

## **Thoughts on classification / Instantiation and generalisation / Specialisation**

Frank, Ulrich

In: ICB Research Reports - Forschungsberichte des ICB / 2012

This text is provided by DuEPublico, the central repository of the University Duisburg-Essen.

This version of the e-publication may differ from a potential published print or online version.

DOI: <https://doi.org/10.17185/duepublico/47059>

URN: <urn:nbn:de:hbz:464-20180917-135832-0>

Link: <https://duepublico.uni-duisburg-essen.de/servlets/DocumentServlet?id=47059>

License:

As long as not stated otherwise within the content, all rights are reserved by the authors / publishers of the work. Usage only with permission, except applicable rules of german copyright law.

Source: ICB-Research Report No. 53, December 2012



**ICB**

Institut für Informatik und  
Wirtschaftsinformatik

Ulrich Frank



# Thoughts on Classification / Instantiation and Generalisation / Specialisation

**ICB-RESEARCH REPORT**



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.

---

Alle Rechte vorbehalten. Insbesondere die der Übersetzung, des Nachdruckes, des Vortrags, der Entnahme von Abbildungen und Tabellen – auch bei nur auszugsweiser Verwertung.

All rights reserved. No part of this report may be reproduced by any means, or translated.

---

**Author's Address:**

Ulrich Frank

Institut für Informatik und  
Wirtschaftsinformatik (ICB)  
Universität Duisburg-Essen  
Universitätsstr. 9  
D-45141 Essen  
[ulrich.frank@uni-due.de](mailto:ulrich.frank@uni-due.de)

**ICB Research Reports**

**Edited by:**

Prof. Dr. Heimo Adelsberger  
Prof. Dr. Frederik Ahlemann  
Prof. Dr. Klaus Echtele  
Prof. Dr. Stefan Eicker  
Prof. Dr. Ulrich Frank  
Prof. Dr. Michael Goedicke  
Prof. Dr. Volker Gruhn  
PD Dr. Christina Klüver  
Prof. Dr. Tobias Kollmann  
Prof. Dr. Klaus Pohl  
Prof. Dr. Erwin P. Rathgeb  
Prof. Dr. Rainer Unland  
Prof. Dr. Stephan Zelewski

---

**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](mailto:icb@uni-duisburg-essen.de)

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



“Die Beziehung eines Gegenstandes zu einem Begriffe erster Stufe, unter den er fällt, ist verschieden von der allerdings ähnlichen eines Begriffes erster Stufe zu einem Begriffe zweiter Stufe.”<sup>1</sup>

Gottlob Frege

---

<sup>1</sup> “The relationship of an object to a first-level concept, it is subsumed to, is different from the, however, similar relationship of a first-level concept to a second-level concept.” (translation by U.F.)



## Abstract

Classification and generalisation are arguably the most important abstractions used in conceptual modelling. From a software developer's perspective it may seem that there is a clear difference between classification and generalisation. However, this impression is deceptive. It is indeed relatively easy to decide whether classification or generalisation – and instantiation or specialisation respectively – is the better choice as long as models are considered on the  $M_1$  layer only. However, as soon as higher levels of abstractions are included, as it is characteristic for meta modelling, this decision can become very demanding. Starting with focussing on obvious criteria to distinguish between classification/instantiation and generalisation/specialisation, this report will first illustrate why these criteria become blurred on higher levels of abstraction. Also, it will be shown that often neither generalisation/specialisation nor classification/instantiation is sufficient. Instead, there is need for further abstractions. Against that background an approach to structure the problem is presented. While it does not allow to strictly determine the decision between the two abstractions, it provides guidelines which support the decision in a particular modelling context.



# Table of Contents

<b>FIGURES</b> .....	<b>III</b>
<b>TABLES</b> .....	<b>IV</b>
<b>1 INTRODUCTION</b> .....	<b>1</b>
<b>2 OSTENSIBLE DIFFERENCES</b> .....	<b>1</b>
2.1 BACKGROUND: CONCEPTS OF CLASS .....	5
2.2 DELEGATION .....	8
<b>3 RAISING THE LEVEL OF ABSTRACTION</b> .....	<b>8</b>
3.1 "FAKE" INSTANTIATIONS .....	9
3.2 MULTI-LEVEL INSTANTIATION/SPECIALISATION .....	10
<b>4 AN ATTEMPT TO STRUCTURE THE PROBLEM</b> .....	<b>14</b>
4.1 PROPERTIES AND FEATURES .....	14
4.2 AN EXAMPLE .....	15
<b>5 PRELIMINARY CONCLUSIONS</b> .....	<b>21</b>
<b>REFERENCES</b> .....	<b>23</b>

# Figures

FIGURE 1: EXTENSION OF A CLASS.....	5
FIGURE 2: ILLUSTRATION OF INTENTIONAL CONCEPT OF CLASS.....	7
FIGURE 3: ILLUSTRATION WITH SETS.....	7
FIGURE 4: FOSTERING COMPREHENSIBILITY AND INTEGRITY THROUGH SPECIALISATION AND DELEGATION.....	8
FIGURE 5: EXAMPLE OF "FAKE" INSTANTIATION.....	10
FIGURE 6: MULTIPLE LEVELS OF ABSTRACTION.....	11
FIGURE 7: POSSIBLE REPRESENTATION OF SELECTED CONCEPTS.....	11
FIGURE 8: ALTERNATIVE REPRESENTATION.....	12
FIGURE 9: IMPLICIT INSTANTIATION.....	12
FIGURE 10: ILLUSTRATION OF "AMBIGUOUS CLASSIFICATION" – ADAPTED FROM ((KÜHNE 2006), P. 378).....	13
FIGURE 11: "ONTOLOGICAL" VS. "LINGUISTIC" INSTANTIATION (ADAPTED FROM (KÜHNE 2006), P. 10).....	14
FIGURE 12: PARTIAL RECONSTRUCTION OF MULTI-LEVEL HIERARCHY.....	16
FIGURE 13: COMPREHENSIVE REPRESENTATION OF EXAMPLE USING INTRINSIC FEATURES.....	18
FIGURE 14: EMPHASIS ON INSTANTIATION.....	19
FIGURE 15: EXAMPLES OF LOW QUALITY ABSTRACTIONS.....	20

# Tables

TABLE 1: DECISION BETWEEN INSTANTIATION AND SPECIALISATION.....	2
TABLE 2: CHARACTERISTIC CONSTRAINTS OF SPECIALISATION AND INSTANTIATION .....	3
TABLE 3: APPLYING CRITERIA TO DISTINGUISH SPECIALISATION AND INSTANTIATION .....	4
TABLE 4: CRITERIA FOR DECIDING BETWEEN INSTANTIATION AND SPECIALISATION .....	20

## 1 Introduction

The creation of conceptual models as well as the implementation of software systems recommend the use of abstraction concepts. It is directly related to the idea of conceptual modelling: Instead of representing particular objects (instances), it is aimed at concepts, i.e. abstractions that cover a range of objects of the same kind. This contributes to reusing models for varying instance populations. At the same time it makes models robust against ever occurring changes on the instance level. Defining concepts as abstractions over congenerous objects corresponds to *classification*. Concepts can be built through *generalisation*, too. In this case, the focus is on the commonalities of a set of classes. Classification and generalisation are arguably the most important abstractions used in conceptual modelling. From a software developer's perspective it may seem that there is a clear difference between classification and generalisation. Such an assessment is supported by the fact that object-oriented programming languages provide distinct operations for defining classes, superclasses, specialized classes and for instantiating objects. However, this kind of differentiation represents only one side of the coin. Before one applies the operations offered by a programming language, one needs to know which operation is most appropriate for a certain purpose, i.e. one needs to be clear about the conceptualisation of the system to be modelled. As we shall see, respective decisions can be very demanding. First, distinguishing instantiation and specialisation from a conceptual point of view is compromised by the fact that both, the relationship of an instance to its class and the relationship of a class to its superclass, are usually referred to as "is a". This ambiguity of the predicate "is a" in natural language will often prevent a straightforward distinction based on a simple linguistic analysis. Second, the conception of a class in prevalent programming languages differs from the meaning we normally associate with the concept of a class. This difference results in subtle, but substantial effects on the semantics of instantiation and specialisation. Third, the distinction between both concepts is further complicated, if a model is not restricted to the dichotomy of instance and class, but allows for further levels of abstraction such as meta classes or meta types. The elaborate analysis of classification/instantiation and generalisation/specialisation extends an earlier report where the consideration was mainly restricted to two levels of abstraction (Frank 2011b).

## 2 Ostensible Differences

The following propositions illustrate the ambivalent use of the predicate "is a":

"John is a student."

"A student is a person."

To adequately represent the corresponding relationships in a conceptual model, it is required to decide in each case whether to represent it as instantiation or as specialisation. With re-

spect to system modelling, it may be required to distinguish between the concept as it is used in the relevant domain of discourse and its representation in a system. At first, we will restrict our consideration to prevalent system architectures that are limited to two levels of abstraction, i.e. classes and instances. That means that every concept has to be mapped either to a class or an instance. If, in this case, we consider the proposition “A is a B”, we only need to decide for A and B whether to represent them as a class or an instance. Then, we can determine, whether specialisation or instantiation are feasible. Table 1 shows the corresponding decision table.

		A	
		instance	class
B	instance	none	none
	class	Instantiation	Specialisation

**Table 1: Decision between Instantiation and Specialisation**

Unfortunately, this kind of decision is not satisfactory. On the one hand, it is restricted to formal feasibility and does not account for conceptual adequacy. On the other hand, it implies deciding beforehand whether a certain object should be represented as class or as instance. As far as the second aspect is concerned, one may argue that it will usually be clear how to represent an object: If it is a particular occurrence of something, it should be represented as an instance. If it serves as an abstraction over a set of further objects, it should be represented as a type. However, there are cases, where one needs to represent concepts as instances – as a consequence of the fact that no more than two levels of abstraction are available. Consider the following example: “A lion is a mammal”. Even though the term “lion” does not represent a particular animal here, it may make sense to represent it as an instance – in order to model it as an instance of mammal. Hence, the decision about modelling something as an instance or a class may depend on the decision for or against using instantiation. Therefore, further criteria are required to support the original decision between instantiation and specialisation. One characteristic feature of specialisation is monotonic extension. It implies that all propositions that hold true for a superclass hold true for its subclasses, too.<sup>1</sup> It also implies substitutability: An instance of a superclass can be replaced by an instance of a

---

<sup>1</sup> Note that there are exceptions to this rule such as propositions that refer to the relationship between superclass and subclass (“A is superclass of B”) or that refer to the state of instance populations (“A has 138 instances”).

subclass without notice, i.e. without causing harm. Instantiation does not result in monotonic extension. Propositions about a class do not have to apply to its instances. For example: The proposition “Customer represents a class of customers” does not hold true, if “Customer” is replaced by one of its instances. Substitutability is not an issue in case of systems with classes and instances only, because instances cannot be further instantiated.

The natural language concept of instantiation and specialisation corresponds to respective concepts in logic, where it is common to speak of “subordination” instead of specialisation and of “subsumption” instead of instantiation (Wolters 1996). In both cases, the definition refers to the concept of class. A class B is a subordination of another class A, if the elements of B represent a true subset of the elements of A. For instance: The class “Student” is subordinated to the class “Person”, if the set of instances of “Student” is part of the set of instances of “Person”. Subsumption on the other hand describes a relation between an element and its class: An element is subsumed to a class, if it is an element of the set represented by the class. Similarly, in his analysis of concepts, Frege characterized subsumption as a relation between a concept („Begriff“) and an object („Gegenstand“). A concept is defined by properties („Merkmale“) that allow distinguishing it from other concepts. Objects that are subsumed under a concept are characterized by particular occurrences of the properties (“Eigenschaften“) the corresponding concept is specified with ((Frege 1892), p. 201). Subordination on the other hand is a relation between two concepts  $c_a$  and  $c_b$  that satisfies two constraints. First, all properties that are characteristic for  $c_a$  are characteristic for  $c_b$ , too. But not all properties of  $c_b$  are necessarily properties of  $c_a$  (ibid, p. 202). Apparently, this aspect corresponds to inheritance. Second, every object that is subsumed under  $c_b$  is subsumed under  $c_a$ , too. This aspect of Frege’s conception is similar to substitutability.











For instantiation it is characteristic that the set of all possible instances a class may have is specified with the class (see 2.1). Specialisation, on the other hand, allows for adding an unlimited number of arbitrary properties to a given set of properties. In Table 2, respective characteristics of specialisation are opposed to those of instantiation. For this purpose, we assume that B is specialised from A and D is an instance of C.

Specialisation	Instantiation
Both, A and B are classes.	C is a class. D is an instance (i.e. not a class).
Substitutability: Every instance of A can be replaced by an instance of B without notice, i.e. without jeopardizing system integrity.	Instances of D are not possible (because D is represented as an instance).
A specifies only a part of the properties of B.	All possible features of D are specified with the properties defined for C.

**Table 2: Characteristic constraints of specialisation and instantiation**

## Ostensible Differences

Applying these constraints to the above example propositions produces clear results (see Table 3).

<i>"John is a student."</i>	Instantiation	Specialisation
"John" represents an instance. "Student" represents a class.		
Substitutability: There are no instances of "John".	-	
All possible features of "John" are specified with "Student".		
<i>"A student is a person."</i>	Instantiation	Specialisation
Both, "student" and "person" represent classes.		
Every instance of "Student" qualifies as an instance of "Person". Hence, instances of "Person" can be replaced by instances of "Student" without notice.	-	
"Person" specifies only a part of the properties that define "Student".		

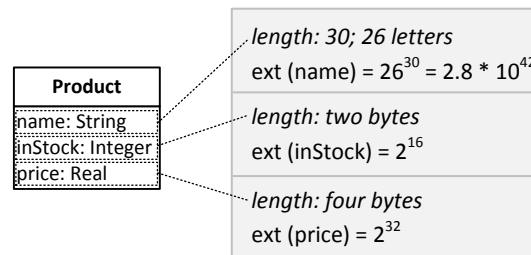
**Table 3: Applying criteria to distinguish specialisation and instantiation**

For those who are familiar with the use of an object-oriented programming language, the above criteria may seem trivial. As long as we restrict the consideration to two levels of abstraction, instance and class, only, the distinction is indeed not much of a challenge – even though the specific semantics is not always easy to determine as we shall see in the next paragraph. However, even with only two given levels of abstraction, modelling a particular domain can be more demanding. Different from the above example, there are cases where it is not clear whether to represent a concept as a class or an instance – with respective consequences on the decision between specialisation and instantiation. For example: "Business report" is meant to represent a class – not a particular report. This can be the case for "document", too. As a consequence, the proposition "business report is a document" could be represented as a specialisation relationship. However, it is also conceivable that "document" is actually intended to represent a class of document classes, i.e. a meta class (see below). In an environment that does not include meta classes, it may be an appropriate option to model "document" as a class (i.e. to represent a meta class with a class) and "business report" as an instance.

## 2.1 Background: Concepts of Class

With these definitions at hand, we could try to resolve the ambiguities of “is a”. However, that would be premature for two reasons: First, the concept of class in prevalent programming languages is different from the concept of class in logic (and in natural language as well) – which has a substantial effect on the semantics of specialisation and instantiation. Second, there is need to clarify how to interpret the term “element” or “object” respectively.

In logic, a class is defined extensionally, i.e. as the extension (set) of its potential instances. Such a set-theoretic view has two implications. First, it suggests regarding the process of instantiation as *selecting* an element of a given set – rather than creating it. Hence, regarding instantiation as selecting an element from a given set would implicitly include the initialization of the element, i.e. the assignment of a particular state. Second, it implies that an element of a class can be an element of other classes at the same time. In programming languages, a class is based on an intentional definition<sup>2</sup>. In this sense, a class serves to specify the properties intended for all its instances. It includes an explicit instantiation procedure that allows for creating particular instances from the class description. This conception can be illustrated by thinking of a class as a template that allows *creating* instances by “punching” them out – even though this would be an all too simplified view as we shall see. At first sight, it may seem that the intentional and the extensional notion of class are the same and mark only different perspectives. They are, indeed, very similar. The intentional specification of a class includes an implicit definition of its extension: The extension of a class with properties  $p_1, p_2, \dots, p_n$  is the product of the extensions of the properties:  $\text{ext}(p_1) * \text{ext}(p_2) \dots * \text{ext}(p_n)$ . Often, the extension of a class will be huge – and much bigger than the number of meaningful instances, as the example of a simple class in Figure 1 illustrates:



$$\text{ext}(\text{Product}) = 2.8 * 10^{42} * 2^{16} * 2^{32} \approx 10^{60}$$

**Figure 1: Extension of a class**

Also, as the example shows, an intentional definition may include extensional elements, too – such as the specification of attributes. Nevertheless, there are two important differences. The first relates to the differences of the instantiation process: While the intentional concept

<sup>2</sup> Note that this use of the predicate “intentional” is not related to human intention. Also, it is extensional in the sense that it implies the explicit specification of all properties that constitute the semantics of a class.



implies a classification that is based on common properties and therefore allows for creating instances according to this common schema, the extensional concept requires specifying a set of instances. While these should share common properties, it would be sufficient for them to satisfy the predicate that they are element of the same class. That makes the extensional concept more flexible, may however compromise the quality of the abstraction that goes along with classification. For example: A software system to support human resource management requires an assessment of employees' expertise using four predicates. An intentional specification would require specifying a class "expertise" with properties that are characteristic for all expertise levels and that allow for instantiating and initialising a particular kind of expertise. For an extensional specification it would be sufficient to define the set of strings that represent possible kinds of expertise: ["outstanding", "very good", "fair", "poor"]. The second difference between the intentional and extensional concept has more severe implications. While an instance of a class that corresponds to the intentional concept – as it is used in prevalent programming languages – is instance of one class only, an instance of a class that corresponds to the extensional notion can be instance of many classes at the same time – like an element of a set can be element of further sets. The specialisation hierarchy in Figure 2 shows a model that seems to be intuitive: The class "Person" is specialized into the classes "Student" and "Employee". The consequences of the intentional concept of class are illustrated by showing corresponding instances. Apparently, the intentional concept of class is not the same that we use in natural language. We would regard it as bizarre, if somebody who is a student would not be a person anymore. Figure 3 illustrates the difference with a representation of the respective classes as sets: The extensional conception of class implies that "Student" and "Employee" are true subsets of "Person", which may overlap. Hence, a particular element can be "Student", "Employee" and "Person" at the same time. The intentional conception, on the other side, implies three distinct sets – this corresponds to the image of a template for "punching" out instances. Therefore, representing somebody both as a person and a student would require two different objects – resulting in redundancy that would jeopardize system integrity. As an alternative, creating an instance of "Student" could go along with deleting the corresponding instance of "Person". In that case, it would not only be required to transfer the state of the "Person" object to the "Student" object, but also to redirect all references to the "Person" object – again causing a threat to system integrity.

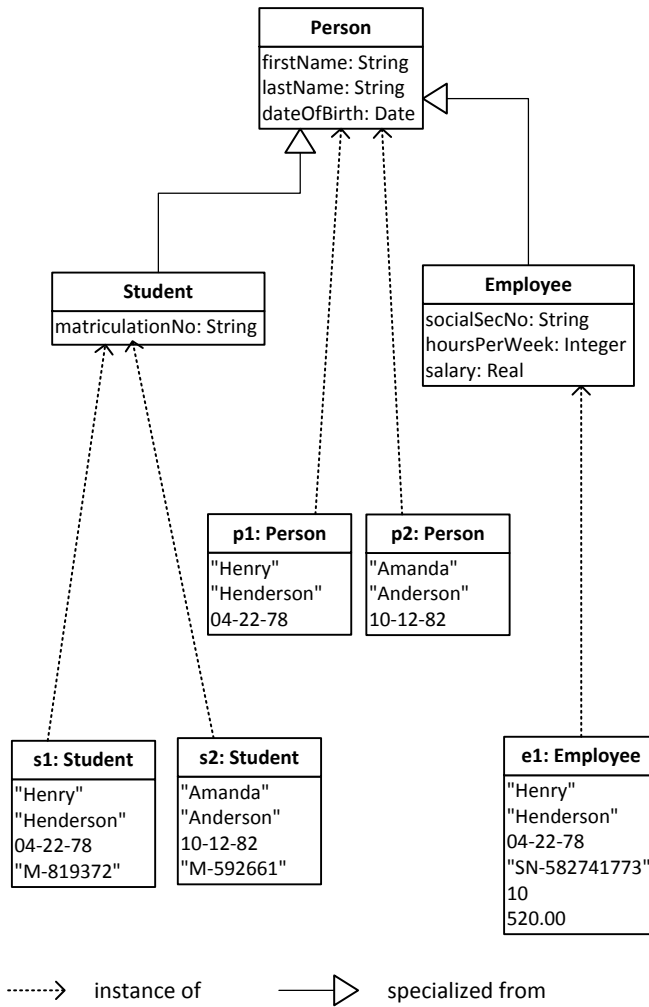


Figure 2: Illustration of intentional concept of class

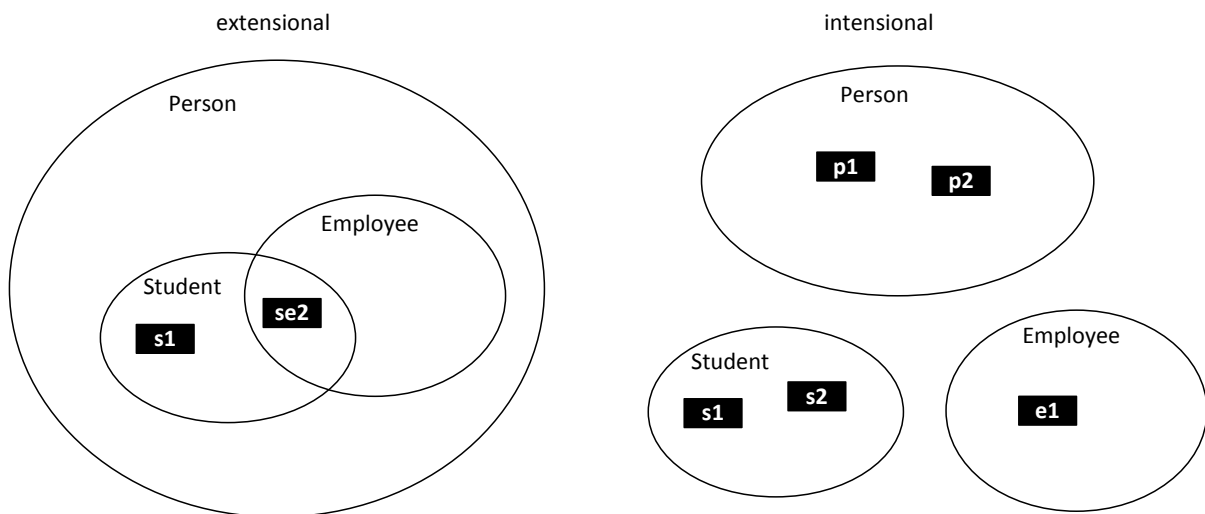


Figure 3: Illustration with sets

## 2.2 Delegation

The intentional concept contributes to confusion and jeopardizes adaptability and integrity. Therefore, the extensional concept seems a clearly better choice. However, when it comes to building software, the intentional concept has the advantage that there is no mismatch between a conceptual model and the implementation. To reduce the negative impact of the intentional concept of class, it is important that modellers and software developers are aware of its counter-intuitive semantics. Delegation is a useful approach to address the limitations of specialisation that result from an intentional concept of a class ((Lieberman 1986), (Frank 2000)). In a particular context, an object that serves as a role filler delegates its responsibilities to an object that serves as one of its roles. For example: A student may be regarded as a role of a person. Whenever an object of the role class “Student” receives a message that is not part of its protocol, it would dispatch the message to its role filler. As a consequence, it would not only “inherit” the properties of the class “Person”, but also the state of the corresponding role filler object. The example in Figure 4 illustrates how to combine specialisation and delegation.

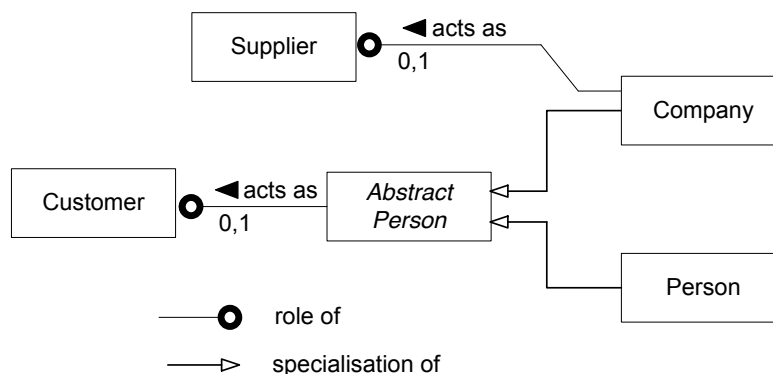


Figure 4: Fostering comprehensibility and integrity through specialisation and delegation

## 3 Raising the Level of Abstraction

Classification allows for abstracting over a range of instances. To take advantage of similarities between classes (or types), it can be useful to introduce a higher level of abstraction, i.e. to define meta classes that specify the properties shared by a range of classes. A similar approach can be applied to the specification of a class of models of the same kind through meta models. In principle, raising the level of classification is not limited to one meta level. Hence, we need to account for meta meta classes, meta meta models etc., too. While these additional meta layers can be very beneficial with respect to reuse and integrity, they create additional complexity at first – and may make the decision between instantiation and specialisation more demanding. So far, our consideration was mainly focussed on classes. Now, we will explicitly include types and meta types, too, because for the specification of DSML one will usually focus on types instead of classes. Nevertheless, we will use both abstractions widely

synonymously – with the main difference that classes other than types allow for user-defined operations. In Logic, the notion of a type is coined by type theory which was proposed by Russel and Whitehead to overcome the antinomy that Russel had discovered in Frege’s axiomatic foundation of set theory. In programming languages, *type systems* are essential for checking the formal integrity of programs. In object-oriented programming languages, a class is defined by its internal structure and a set of operations. The interface of a class defines its type. Therefore, it is possible that a class may satisfy more than one type. While the peculiarities of type theory and type systems are important for understanding the concept of a class as it is used in software engineering and in conceptual modelling, differentiating types and classes is of no specific value for our course of investigation.

### 3.1 “Fake” Instantiations

While there seems to be a wide consensus that a meta model is a model of a class of models, the relationship between a meta model and corresponding models can be conceptualized in different ways. Often, a model is regarded as an instance of a corresponding meta model. According to this view, a meta model is instantiated into models. From a formal point of view that means that all models that are syntactically and semantically correct can be generated from a meta model. According to Bézivin it is confusing to regard a meta model as a model of other models: A model of something should provide a representation that provides a reduction of complexity ((Bézivin 2003), p. 9 f.). Apparently, this not the case for a meta model. While such an objection may seem reasonable at first sight, it is not convincing in the end: A meta model is not a model of a particular model, but of an entire set of models. Hence, it can very well reduce the complexity of dealing with the whole range of models. Nevertheless, the idea that a meta model is instantiated into its models, i.e. that a model is an instance of a corresponding meta model, is indeed not totally convincing. On the one hand, it is somewhat misleading, because it suggests that creating a meaningful model from a meta model is a well defined process that can be performed by a machine. However, this is usually not the case. Even for trivial meta models the number of possible models can be extremely high, so that generating the entire range is no option. In addition to that, identifying meaningful models will be much more demanding than it will often be for instances of a particular class. Instead, instantiation in the case of modelling is rather a process of construction in conformance with the rules defined in the meta model. On the other hand, it happens on a regular base that the meta types used in meta models are based on formally incorrect classifications – which go along with “fake” instantiations. According to the notion of instantiation that we used so far, instantiating a class means to create an object with features that are instances of the properties defined for the corresponding class. However, frequently meta types are used that include properties which are incorrect classifications of the respective type features. Figure 5 shows two different instantiations of the type “Person” from the meta types of a corresponding meta model. The example on the right shows a formally correct instantiation, which, however, represents an instance that does not make much sense. The

example on the left shows an instance that represents a valid type, which, however, results from a formally incorrect instantiation. To justify the abstraction used in the meta model, one could apply a more relaxed interpretation. In this case, one would regard “String” as an abstraction over the set of allowable types (or classes), i.e. as a placeholder for an extensional representation such as “Character”, “Date”, “Integer” etc., each of which would serve as a unique identifier of a certain type. To enable a formally correct instantiation of types, there would be need for specifying a meta type that represents a classification of all intended types.

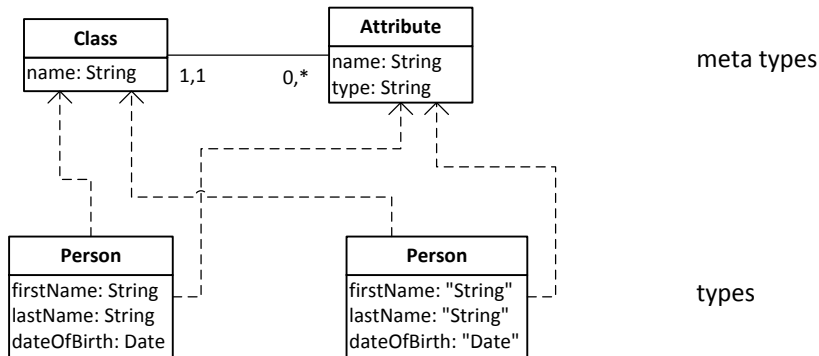


Figure 5: Example of "fake" instantiation

As we shall see in the next section, there is a further reason why it may be inappropriate to regard a model as an instance of a meta model.

### 3.2 Multi-Level Instantiation/Specialisation

So far, our consideration of instantiation and specialisation was restricted to two levels of abstraction, i.e. to instances and classes or types respectively. Although the distinction between the two concepts may be confusing sometimes even on this level, there are clear rules that support the decision. If we also account for concepts that describe a range of concepts, i.e. meta concepts, those rules are not as clear anymore. For Frege the distinction of object (“Gegenstand”) and concept (“Begriff”) was essential to structure the use of language. Thereby an object is regarded as falling within a concept and being characterized by a proper name – in other word: something that exists and that is not an abstraction over other existing entities. That does not mean, however, that the distinction between object and concept is trivial (for a discussion on the difficulties in identifying the “being” see (Quine 1953)). In advanced object-oriented languages, there is no clear distinction by definition: An object is an instance of a class. At the same time, a class is regarded as an object, too, i.e. it has specific attributes and methods that are not available on the instance-level.

The example shown in Figure 6 illustrates that with more than two levels of abstraction the decision between instantiation and specialisation can become very demanding. The directed edges represent only that one concept is regarded to be more specific than the one above.

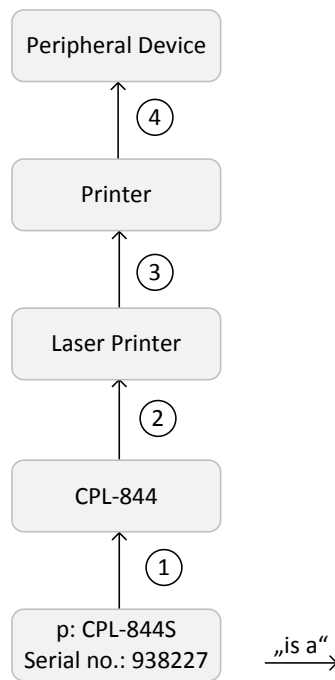


Figure 6: Multiple levels of abstraction

To analyse the question whether specialisation or instantiation is more appropriate – or whether there is even need for further concepts, we consider possible representations of “Device” and “Printer” (see Figure 7).

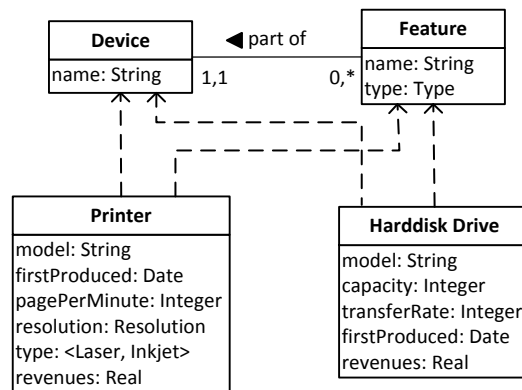


Figure 7: Possible representation of selected concepts

Apparently, “Printer” and “Harddisk Drive” are instances of “Device” and “Feature” respectively: Each attribute of both concepts is instantiated from attributes of the superior classes. However, an alternative representation is possible, too (see Figure 8).

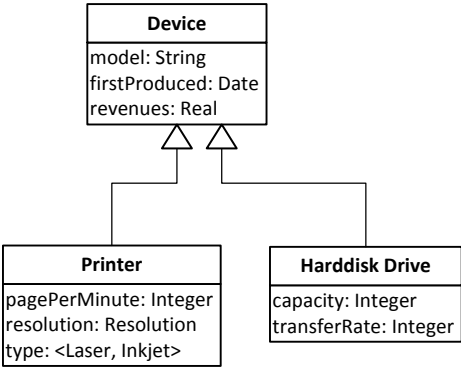


Figure 8: Alternative representation

In this case, “Printer” and “Harddisk Drive” would obviously qualify as specialisations of “Device”. Hence, this example illustrates that the same result can be produced through instantiation and specialisation, if the superior class is adapted accordingly. At the same time, it stresses the question, which alternative is more appropriate. To add to the confusion, we will first show each specialisation includes an implicit instantiation, too. If a class is specialised from an existing class, this will result in creating a new class. In the above example, specialisation results in two new classes, namely “Printer” and “Harddisk Drive”. At the same time, both new classes were created through an act of instantiation: They were instantiated from one or more meta classes. In the example shown in Figure 9 this is illustrated for the class “Printer” only, which is instantiated from the meta classes “Class” and “Property”.

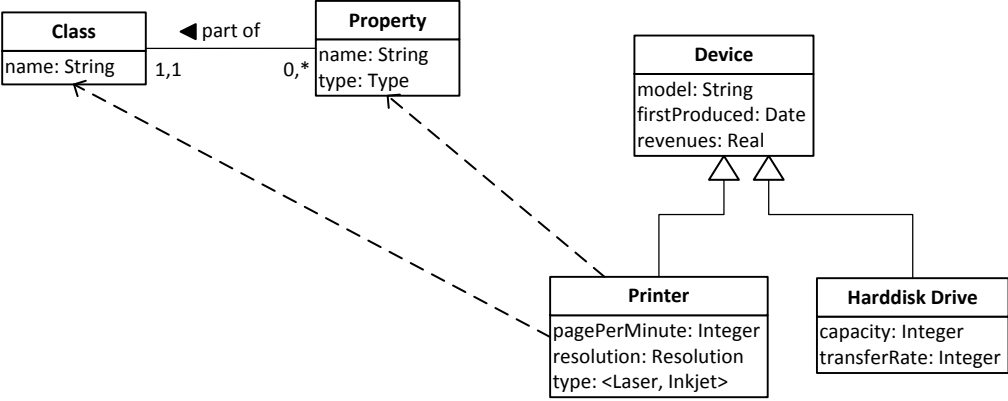
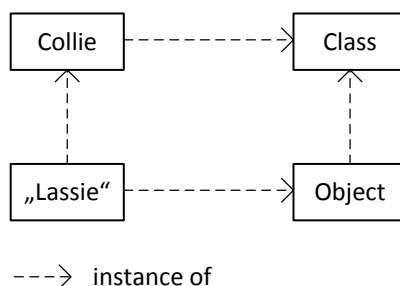


Figure 9: Implicit instantiation

Using a similar perspective, however focussing on instantiation alone, Atkinson and Kühne consider a constellation they refer to as “ambiguous classification” ((Atkinson and Kühne 2001), p. 21). It seems that there are two alternative classifications for each class and each instance. In a later publication Kühne gives the example shown in Figure 11 to illustrate the problem.



**Figure 10: Illustration of "ambiguous classification" – adapted from ((Kühne 2006), p. 378)**

It seems confusing that "Lassie" is an instance both of the classifier "Collie" and the classifier "Object". To dissolve the confusion, Kühne suggests differentiating two kinds of instantiation. "Ontological instantiation between two elements or models is therefore based on the relationship between them in terms of their meaning." ((Kühne 2006), p. 377) Apparently, Kühne thinks of meaning being created by references to "the original system". In other words: An ontological instance should refer to an element that is "in the extension of the concept referenced by the type." The concept is supposed to be defined intentionally. "Linguistic" instantiation on the other hand is based on an extensional concept of class, which is to express that the concept does not express the meaning of the corresponding instances: ".. linguistic instantiation concerns the form of elements themselves, as opposed to their content (and meaning respectively) as is the case with ontological instantiation." (p. 377). An example would be instantiating the concept "Collie" from "Class", or "Lassie" from "Object".

While pointing at what he calls "ambiguous classification" is important with respect to clarify potential confusion, Kühne's conclusions are not entirely convincing. First, the differentiation of "ontological" and "linguistic" is misleading for various reasons. On the one hand, both are orthogonal. Every "ontological" concept requires a linguistic representation. Otherwise we cannot express it – independent from the philosophical question whether ontological categories can exist without language. A linguistic expression may or may not represent an ontological category. On the other hand, terms such as "class" or "object" that Kühne assigns to "linguistic instantiation" are typical examples for philosophical ontologies (as the one suggested by Bunge). Also, a term like "Collie" represents a linguistic concept, i.e. it is part of a language. Second, and more important, the differentiation seems to be based on a misconception. If we look at a model where "Lassie" is represented as an instance of "Collie", then we deal with classes and objects only. "Collie" is a particular class – not instantiated from a class (and an object at the same time!) – with the "name" property instantiated to "Collie" – i.e. it is instantiated from a metaclass. This particular class is then instantiated into an object with the name property instantiated to "Lassie". Hence, the terms as "class" and "object" are terms of a meta language that enable us to speak about language – and to classify language elements.



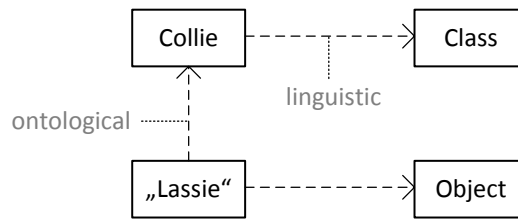


Figure 11: "Ontological" vs. "linguistic" instantiation (adapted from (Kühne 2006), p. 10)

Kühne's example nevertheless shows that there is need to distinguish between language and language application. While this distinction may contribute to a more elaborate understanding of instantiation – and specialisation respectively, it does not help much with the decision between instantiation and specialisation.

## 4 An Attempt to Structure the Problem

Our further considerations to clarify this problem focus on properties and features of classes and meta-level classes. They also account for the peculiarities of relationships between concepts on different levels of abstraction. Frege makes a difference between first-order concepts and second-order concepts – and emphasizes that the relationship between an object and a first-order concept is similar, but different from the relationship between a first-order and a second-order concept. In the first case, he speaks of an object falling under a concept ("ein Gegenstand falle unter einen Begriff", (Frege 1892), p. 201). In the second case, he uses the phrase of a concept falling into a second-order concept ("ein Begriff falle in einen Begriff zweiter Stufe", *ibid*). Unfortunately, Frege's distinction is of limited use for our investigation only, since it is based on a clear differentiation of objects and classes. However, for our purpose, classes are regarded as objects, too.

### 4.1 Properties and Features

To develop criteria that support a respective decision, we consider two classes (or meta classes respectively)  $C_a$  and  $C_b$ . We assume that  $C_a$  is regarded to represent a higher level of abstraction than  $C_b$ , where it is not clear at first whether it is a meta class or a superclass of  $C_b$ . To analyse this question in more detail, it makes sense to first define all properties that  $C_a$  should have for the intended purpose. In a next step, one would check for each property whether  $C_b$  should inherit them, i.e. should have exactly the same properties or whether intended properties of  $C_b$  should be instantiated from respective properties of  $C_a$ . To develop a heuristic to support a respective decision, we distinguish between different kinds of class properties and features – inspired by Frege's distinction of "Merkmal" (property) and "Eigenschaft" (feature). A feature is an instantiation of a corresponding property. For this pur-

pose we consider classes as objects (i.e. not just as sets) that have features and properties. There are three categories of class features and properties:

*Class features:* These are properties that serve to characterise a class and cannot be applied to particular instances of a class.<sup>3</sup> They are based on the conception of a class as an object with a certain state. There are two kinds of class features. Note that class features are instantiated from class properties defined on a higher level of abstraction.

- a) *Life-cycle features:* Features that are characteristic for the class only and do not apply to instances. For example: time when a class was created; times when modifications took place. The class name falls into this category, too.
- b) *Derivable features:* Features that apply to the class only, but depend on the corresponding instance population. Examples include the number of instances at a certain point in time, the average number of instances within a certain time frame or the average value of certain instance properties. In addition to the direct instances of a class, the relevant instance population may also include instances of subclasses. To avoid redundancy, these features would usually be implemented with methods.

Since class features are characterised by class-specific *values* they can neither be inherited to other classes nor instantiated. At the same time, they cannot be inherited from an upper-level class. Instead, they can be instantiated from properties in an upper-level class only.

*Class properties* serve to characterize and differentiate specific instances. Properties are used for the intentional definition of a class. A class property corresponds to a feature of corresponding instances. Since an instance may be a class, a class feature may correspond to a property of the respective meta class. For instance: The meta class “Class” may include the property “created”, which is instantiated with each of its instances into a class feature. Class properties can be inherited through specialisation.

*Class invariants:* These are constraints that apply to class properties. A class invariant is valid for all instances, hence does not allow differentiating between instances. For example: the restriction of an attribute’s value range, e.g. for representing employees’ salaries, the value of a certain property, i.e. the power consumption of an electric appliance, or the dependence of one property on another one – e.g. the sales price of a product has to be larger than the retail price. Like class properties class invariants can be inherited through specialisation – which is an implication of the substitutability constraint. They cannot be further instantiated.

## 4.2 An Example

In the following we apply the above categories to classes that correspond to the example in Figure 6. We start by focussing on three levels only. We make the assumption that instantia-

---

<sup>3</sup> This statement corresponds to Frege’s determination that – in a strict sense – propositions about concepts do not apply to corresponding objects (ibid, p. 201).

## An Attempt to Structure the Problem

tion and specialisation are mutually exclusive. Hence, if there are conflicting requirements, a decision for one of the two has to be made.

For deciding whether  $C_b$  should be modelled as an instance of  $C_a$  or rather as a subclass, one would first categorize all relevant features and properties of  $C_a$ . Note that assigning a feature or property to one of these categories may require analysing the intended purpose of a concept. In the next step, one needs to decide for properties of each category whether and how they are expected to be used in  $C_b$ . The left part of Figure 12 shows a part of the concepts presented in Figure 6. The attributes are assigned to the categories introduced above. The diagram on the right shows a possible reconstruction using a specialisation and an instantiation relationship. They cover the class ( $M_1$ ) and meta class layer ( $M_2$ ). Class features are marked through a grey background.

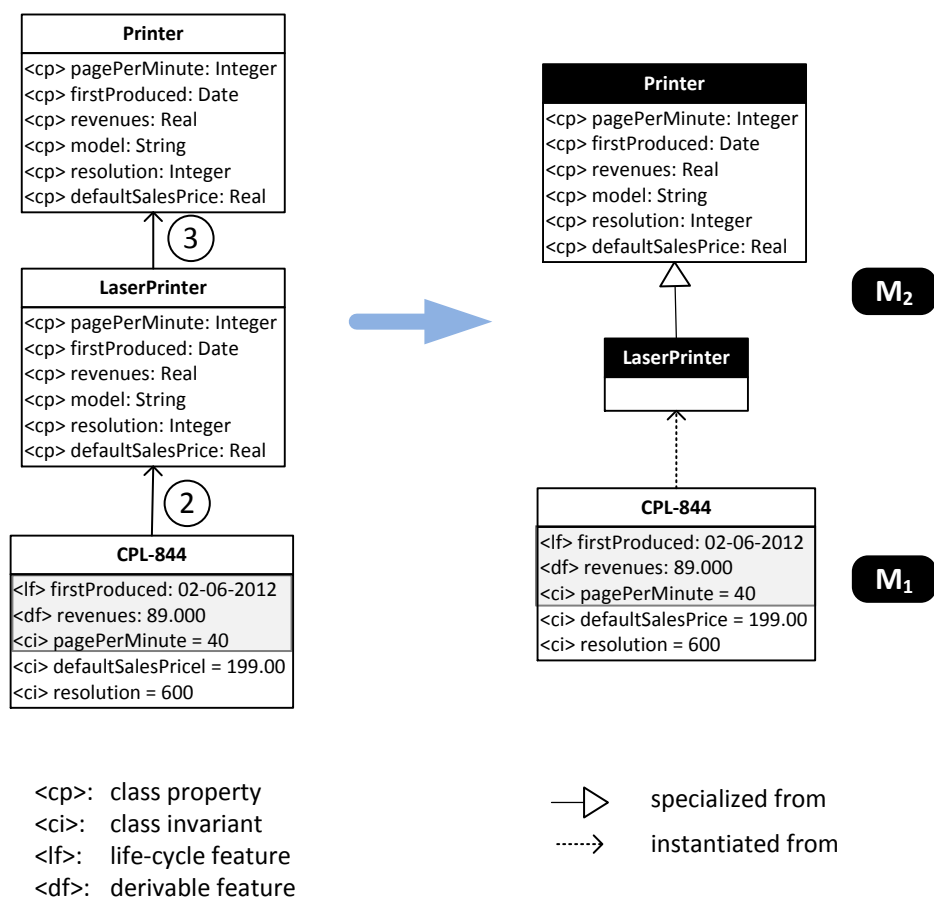


Figure 12: Partial reconstruction of multi-level hierarchy

While the model on the right seems to be appropriate for various purposes, it does not account for class features on the  $M_2$  layer. In case these are important, it has to be clarified where the corresponding properties would be defined. Furthermore, the model does not account for representing particular instances. In many cases a corresponding information system would have to keep track of particular devices. However, the class "CPL-844" does not allow for further instantiations. Figure 13 shows an extension of the previous model that

addresses both aspects. To allow for class features on the  $M_2$  layer, the corresponding meta classes are instantiated from meta meta classes on an additional  $M_3$  layer, in this case from “PeripheralDevice” and “Feature”. Enabling class properties for “CPL-844” which could be instantiated into features of corresponding instances, such as a specific serial number, is more demanding. It requires an additional (meta) concept. While “serial no” could be defined with “Printer” (as an instantiation of “Feature”) or with “LaserPrinter” (as an extension of inherited properties or – equivalently – as an instantiation of “Feature”), it would have to be instantiated to a class feature within “CPL-844”, since “CPL-844” is modelled as an instance of “LaserPrinter”. There are a few proposals that address this challenge such as “power types” (Odell 1998), “clabjects” (Atkinson and Kühne 2008) or “intrinsic features” (Frank 2011a). Intrinsic features are based on the assumption that conceptualising a meta meta class does not only imply an imagination of its immediate instances, i.e. classes, but may also imply an imagination of those classes’ instances. Hence, when conceptualising a meta class it may be a good idea to already specify class properties that apply to lower level instances only. The corresponding features would then be intrinsic to the concept of a meta class. In the example shown in Figure 13 the concept of an intrinsic feature allows defining the property “serial no” within “Printer” on the  $M_2$  layer. Power types and clabjects allow for representing this, too. However, they come with a slightly different ontological foundation. Also, clabjects are more flexible, since they allow for defining a “potency” to specify on what level a property is supposed to be instantiated. Further approaches to address the above challenge include “m-objects” (Neumayr, Grün et al. 2009) and “materialization” (Olivé 2007) (for an overview see: (Neumayr, Schrefl et al. 2011)).

The example in Figure 13 shows a comprehensive representation of the concepts in Figure 7. Note that “LaserPrinter” does not inherit the values of the class features assigned to “Printer”. Instead, these features are instantiated from its corresponding meta class, i.e. “PeripheralDevice”. To evaluate this model one needs to account for its purpose. Let us assume we want to build a system for supporting order management within a computer hardware dealership. In case, order management is restricted to simply selling devices without the need for giving technical advice, this model may be sufficient. One would have to evaluate whether the meta class “LaserPrinter” is required, since it is not different from its superclass – except for its state as an object. However, if the system should also include technical advice and maybe the configuration of devices, the model is certainly not appropriate. In that case, the concept of a printer should be described in more detail, allowing for clearly distinguishing different kinds of printers.

## An Attempt to Structure the Problem

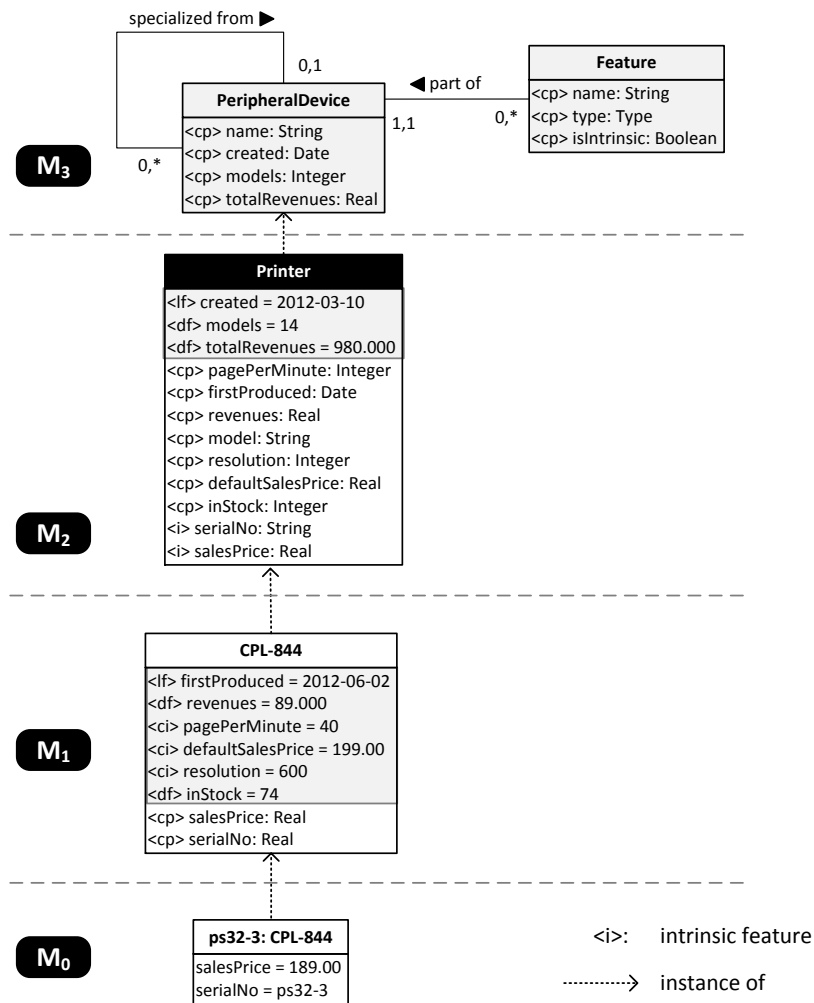


Figure 13: Comprehensive representation of example using intrinsic features

The model shown in Figure 14 illustrates a corresponding representation using instantiation only. Different from the previous model, it includes a conceptualisation of “Printer” that is more elaborate in the sense that it describes what a concept called “printer” is composed of. That does not mean, however, that classification in general results in more elaborate (meta) concepts than generalisation.

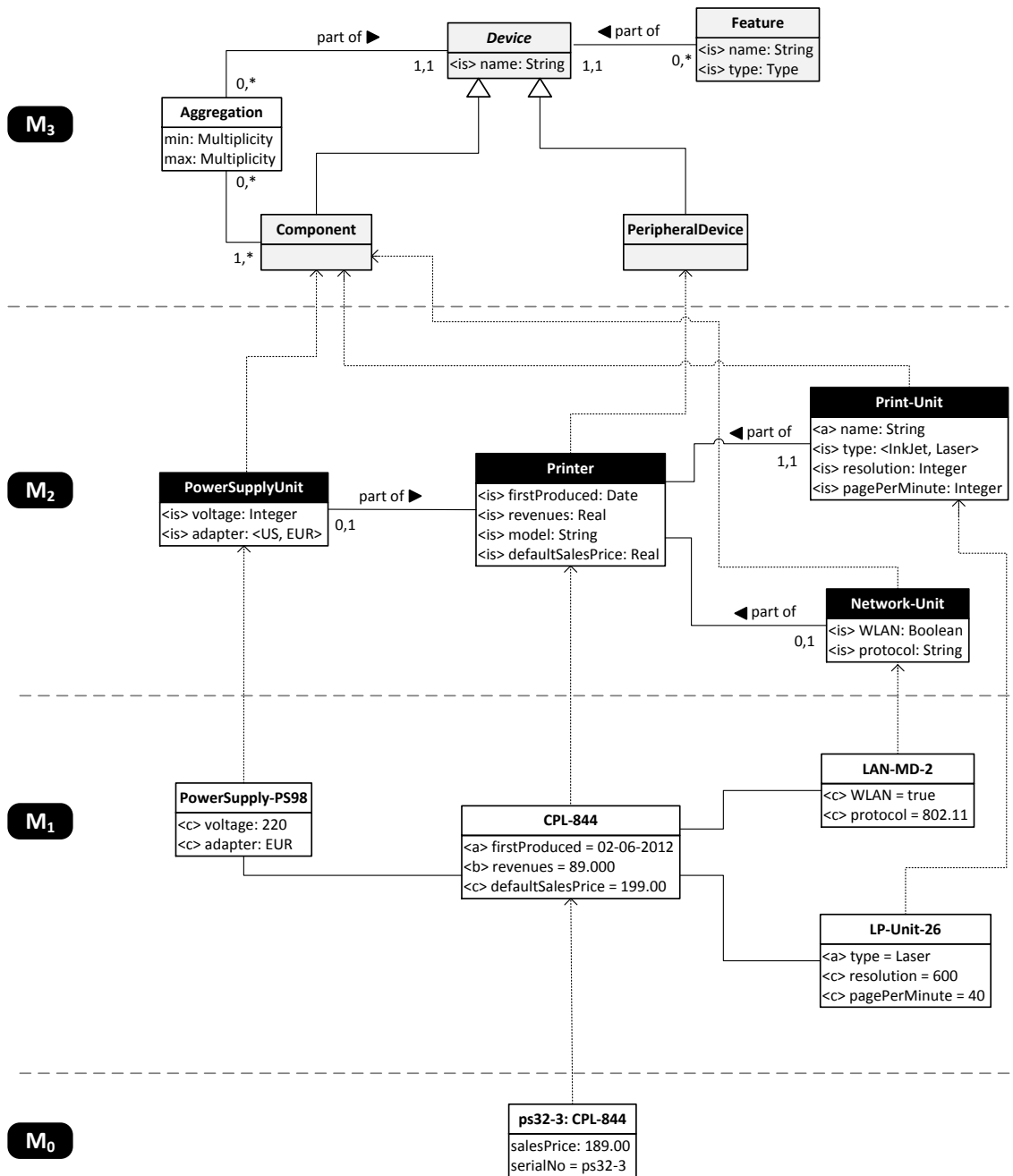


Figure 14: Emphasis on instantiation

The difference between instantiation and specialisation is related to two aspects:

- How can characteristics of the abstraction be (re-) used in a concretisation?
- How can specific characteristics of the concretisation be defined?

Ad a): In the case of specialisation, all properties of the superior class are inherited, i.e. used as they are defined in the superior class. This is different with instantiation where it is not possible to use concepts defined in the superior class as they are.

## An Attempt to Structure the Problem

Ad b): Specialisation allows only for specifying additional class properties. Instantiation allows for instantiating class features and (additional) class properties.

The quality of classification versus generalisation depends chiefly on the semantics of the respective meta class or superclass. The more useless concretisations (either specialisation or instantiation) are possible, the lower the level of semantics. In this respect, both specialisation and instantiation have specific strengths and shortcomings. While specialisation does not allow excluding arbitrary extensions, classifications can restrict the number of possible instantiations, i.e. classification allows for defining the permissible characteristics of all possible instantiations. The higher the level of semantics specified for a classification, i.e. the lower the range of possible, however, useless instantiations, the higher is the quality of the corresponding abstraction. On the other hand, the quality of generalisation depends on its semantics, too. The semantics of a generalised class depends on the semantics of its property types and the number of its properties. The more semantics a generalisation represents as invariant commonality of the corresponding specialisation, the higher its contribution to reuse and flexibility. Figure 15 shows the examples of a meta class and a superclass that incorporate very little semantics only, i.e. there is only little value in the corresponding abstraction: anything goes. Both cases result in widely the same set of possible concretisations.

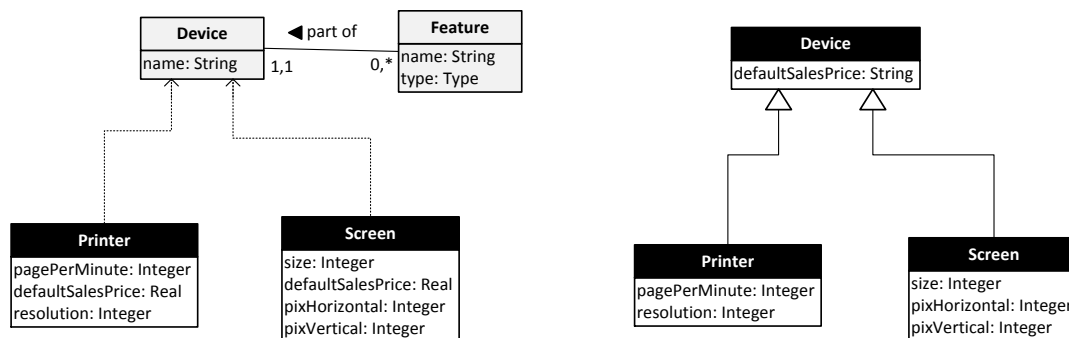


Figure 15: Examples of low quality abstractions

To illustrate the criteria that are to be accounted for when deciding between instantiation and specialisation we consider a concretisation (either specialized or instantiated class) and differentiate its properties/features according to the above proposal.

Request for concretisation		Implication
1	representation of class features (lf, df)	instantiation only
2	reuse of properties as they are	specialisation only
3	specification of further characteristics	
	a) arbitrary	both
	b) restricted	instantiation only

Table 4: Criteria for deciding between instantiation and specialisation

As the examples from Figure 12 to Figure 14 show, there are cases that demand for both instantiation and specialisation. In the worst case, i.e. if both specialisation and instantiation seem mandatory, the resulting conflict creates a dilemma.

## 5 Preliminary Conclusions

Our previous considerations lead to a number of conjectures some of which recommend revising wide-spread assumptions.

*The decision between instantiation and specialisation can be a serious challenge.* As long as one deals with the  $M_0$  and  $M_1$  layer only, the decision between instantiation and specialisation is fairly easy. However, if more levels of abstraction need to be accounted for, this decision may face the dilemma that there are good reasons for both choosing instantiation and specialisation. The problem is attributed to treating classes as objects of corresponding meta classes. As a consequence, a class may have both instantiated features (class as object) and properties that are instantiated only with its instances.

*The distinction between instantiation and specialisation should not be abandoned.* The fact that in some cases a clear *decision* for one of the two is not possible could be used as an argument for giving up the *distinction* between instantiation and specialisation. The difficulties in differentiating both concepts are rather an argument for aiming at a clear separation than against it. If we gave up the differentiation, which is arguable one of the major accomplishments of modern Logic, this would be a step back resulting in a loss of analytic power (Kühne 2009).

*There is urgent need for additional abstractions to overcome the respective decision conflict.* It is widely impossible to create meta models that represent the intended semantics, if there is no chance to somehow combine instantiation and specialisation. Therefore, abstractions that address this issue, such as clajects, powertypes, intrinsic features, materialization or m-objects are mandatory for meta modelling languages.

*A strict separation of abstraction levels is often not possible.* The language hierarchy referred to as  $M_0$  to  $M_3$  as it became popular mainly by the OMG suggests that there is a clear difference between linguistic levels of abstraction, which is defined by classification/instantiation relationships. By regarding (meta) classes as objects and through the use of additional abstractions such as clajects, the distinction of abstraction layers becomes blurry. However, that does not mean that differentiating levels of linguistic abstraction is in general inappropriate. It is still important as an analytical measure to provide an orientation for interpreting models. Only, one should be aware of possible intrinsic overloading.

*Four levels of abstraction are not necessarily sufficient.* Modelling languages and corresponding tools require a language architecture. It includes the meta models that define modelling languages and it may also include the specification of the corresponding meta modelling languages through a meta meta model. Therefore, language architectures often distinguish four



## Preliminary Conclusions

levels of abstraction starting with  $M_0$  – which is usually not relevant for conceptual modeling, but provides a common ground, and going up to  $M_3$ , which represents the meta meta model. The examples in Figure 13 and Figure 14 indicate that it can be useful to introduce further levels of abstraction.

## References

- Atkinson, C. and T. Kühne (2001). The Essence of Multilevel Metamodeling. «UML» 2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools, Springer: 19-33.
- Atkinson, C. and T. Kühne (2008). "Reducing accidental complexity in domain models." *Software and Systems Modeling*: 345-359
- Bézivin, J. (2003). On the Unification Power of Models. Nantes.
- Frank, U. (2000). "Delegation: An Important Concept for the Appropriate Design of Object Models." *Journal of Object-Oriented Programming* 13(3): 13-18.
- Frank, U. (2011a). The MEMO Meta Modelling Language (MML) and Language Architecture. ICB Research Report. Essen, University Duisburg-Essen.
- Frank, U. (2011b). Multi-Perspective Enterprise Modelling: Background and Terminological Foundation. ICB Research Report, University Duisburg-Essen.
- Frege, G. (1892). "Über Begriff und Gegenstand." *Vierteljahresschrift für wissenschaftliche Philosophie* 16: 192-205.
- Kühne, T. (2006). "Matters of (Meta-) Modeling." 5(4): 369-385.
- Kühne, T. (2009). Contrasting Classification with Generalisation. *APCCM 2009*: 71-78.
- Lieberman, H. (1986). Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems. Proceedings of First ACM Conference on Object-Oriented Programming Systems, Languages and Applications. Portland: 214-223.
- Neumayr, B., K. Grün, et al. (2009). Multi-Level Domain Modeling with M-Objects and M-Relationships. Proceedings of the 6th Asia-Pacific Conference on Conceptual Modeling (APCCM). S. Link and M. Kirchberg. Wellington, Australian Computer Society: 107-116.
- Neumayr, B., M. Schrefl, et al. (2011). Modeling Techniques for Multi-level Abstraction. The Evolution of Conceptual Modeling. From a Historical Perspective towards the Future of Conceptual Modeling. R. Kaschek and L. Delcambre. Berlin, Heidelberg, Springer: 68-92.
- Odell, J. (1998). Power Types. *Advanced Object-Oriented Analysis and Design Using UML*. J. Odell. Cambridge, Cambridge University Press: 23-33.
- Olivé, A. (2007). *Conceptual Modeling of Information Systemes*. Berlin, Heidelberg, New York, Springer.
- Quine, W. V. O. (1953). *On What There Is. From a Logical Point of View*. W. V. O. Quine. Boston, Harvard University Press: 1-19.
- Wolters, G. (1996). Subordination. *Enzyklopädie Philosophie und Wissenschaftstheorie*. J. Mittelstraß. Stuttgart, Weimar, Metzler. 4: 132.

## Previously published ICB - Research Reports

### 2012

No 52 (July 2012)

*Berntsson Svennson, Richard; Berry, Daniel; Daneva, Maya; Dörr, Jörg; Frickler, Samuel A.; Herrmann, Andrea; Herzwurm, Georg; Kauppinen, Marjo; Madhavji, Nazim H.; Mahaux, Martin; Paech, Barbara; Penzenstadler, Birgit; Pietsch, Wolfram; Salinesi, Camile; Schneider, Kurt; Seyff, Norbert; van de Weerd, Inge (Eds): "18th International Working Conference on Requirements Engineering: Foundation for Software Quality. Proceedings of the Workshops Re4SuSy, REEW, CreaRE, RePriCo, IWSPM and the Conference Related Empirical Study, Empirical Fair and Doctoral Symposium"*

No 51 (May)

*Frank, Ulrich: "Specialisation in Business Process Modelling: Motivation, Approaches and Limitations"*

No 50 (March)

*Adelsberger, Heimo; Drechsler, Andreas; Herzig, Eric; Michaelis, Alexander; Schulz, Philipp; Schütz, Stefan; Ulrich, Udo: "Qualitative und quantitative Analyse von SOA-Studien – Eine Metastudie zu serviceorientierten Architekturen"*

### 2011

No 49 (December 2011)

*Frank, Ulrich: "MEMO Organisation Modelling Language (2): Focus on Business Processes"*

No 48 (December 2011)

*Frank, Ulrich: "MEMO Organisation Modelling Language (1): Focus on Organisational Structure"*

No 47 (December 2011)

*Frank, Ulrich: "MEMO Organisation Modelling Language (OrgML): Requirements and Core Diagram Types"*

No 46 (December 2011)

*Frank, Ulrich: "Multi-Perspective Enterprise Modelling: Background and Terminological Foundation"*

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; Geroasi, 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 Language 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; Rüdinger, 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
<b>Prof. Dr. H. H. Adelsberger</b> Information Systems for Production and Operations Management	E-Learning, Knowledge Management, Skill-Management, Simulation, Artificial Intelligence
<b>Prof. Dr. F. Ahlemann</b> Information Systems and Strategic Management	Strategic planning of IS, Enterprise Architecture Management, IT Vendor Management, Project Portfolio Management, IT Governance, Strategic IT Benchmarking
<b>Prof. Dr. P. Chamoni</b> MIS and Management Science / Operations Research	Information Systems and Operations Research, Business Intelligence, Data Warehousing
<b>Prof. Dr. K. Echtle</b> Dependability of Computing Systems	Dependability of Computing Systems
<b>Prof. Dr. S. Eicker</b> Information Systems and Software Engineering	Process Models, Software-Architectures
<b>Prof. Dr. U. Frank</b> Information Systems and Enterprise Modelling	Enterprise Modelling, Enterprise Application Integration, IT Management, Knowledge Management
<b>Prof. Dr. M. Goedicke</b> Specification of Software Systems	Distributed Systems, Software Components, CSCW
<b>Prof. Dr. V. Gruhn</b> Software Engineering	Design of Software Processes, Software Architecture, Usability, Mobile Applications, Component-based and Generative Software Development
<b>PD Dr. C. Klüver</b> Computer Based Analysis of Social Complexity	Soft Computing, Modeling of Social, Cognitive, and Economic Processes, Development of Algorithms
<b>Prof. Dr. T. Kollmann</b> E-Business and E-Entrepreneurship	E-Business and Information Management, E-Entrepreneurship/E-Venture, Virtual Marketplaces and Mobile Commerce, Online-Marketing
<b>Prof. Dr. K. Pohl</b> Software Systems Engineering	Requirements Engineering, Software Quality Assurance, Software-Architectures, Evaluation of COTS/Open Source-Components
<b>Prof. Dr. R. Unland</b> Data Management Systems and Knowledge Representation	Data Management, Artificial Intelligence, Software Engineering, Internet Based Teaching
<b>Prof. Dr. S. Zelewski</b> Institute of Production and Industrial Information Management	Industrial Business Processes, Innovation Management, Information Management, Economic Analyses