PREPRINT

Abstraction and concretizing in information systems and problem domains: Implications for system descriptions and theoretical frameworks

Jens Kaasbøll

Preprint 1995-1

February 1, 1995

INSTITUTT FOR INFORMATIKK
UNIVERSITETET I OSLO

Institutt for Informatikk
Universitetet i Oslo
Postboks 1080 Blindern
0316 Oslo

Tlf:	22 85 24 10
Fax:	22 85 24 01
E-post:	jens.kaasboll@ifi.uio.no
WWW:	http://www.ifi.uio.no/~jensj

Department of Informatics
University of Oslo
P.O.Box 1080 Blindern
N – 0316 Oslo, Norway

Phone:	+47 22 85 24 10
Fax:	+47 22 85 24 01
E-mail:	jens.kaasboll@ifi.uio.no
WWW:	http://www.ifi.uio.no/~jensj

Accepted for publication at

The IFIP 8.1 International Working Conference on Information System Concepts: Towards a Consolidation of Views (ISCO3)

(Marburg, Germany, 28–30 March, 1995)

Abstraction and concretizing in information systems and problem domains: Implications for system descriptions and theoretical frameworks

Jens Kaasbøll

Abstract

“Abstraction” is used both for denoting relations in the problem domain of an information system, and for denoting relations inside software and hardware of a computer. This calls for a clarification of the concept, such that frameworks of information system concepts and techniques for analysis and design can distinguish and compare different types of abstractions.

Abstraction is specialized in the paper as follows: representation, classification, generalization, aggregation, and role-realization. The latter relation occurs often when modelling reality, but it is presented with erroneous direction of abstraction in the literature, and it is not supported by techniques for analysis.

It is also shown that separating abstraction in analysis of problem domains from abstraction when designing information systems clarifies the direction of abstraction.

Abstraction relations in a taxonomy of concepts for information systems science and the FRISCO framework are discussed, and improvements suggested. Jackson System Development, object-oriented analysis and design, and dataflow diagrams can be improved through extensions with the abstraction relations specified in this paper.

Keywords: Modelling, analysis, design, object-orientation, system development methods, techniques

1 Introduction

Descriptions of information systems are often complex. One strategy of reducing complexity is to use abstraction. Abstraction should be supported by methods and learnt by students. Yet, “abstraction” is a diverse issue. In everyday language, *abstraction* is to ignore aspects of a phenomenon and concentrate on other aspects. Thus abstraction can be used to reduce complexity through ignoring the irrelevant aspects while focusing on the relevant ones.

Because of this need to reduce complexity, concepts and mechanisms for abstraction appear in programming languages, system description languages, user interface construction tools, and frameworks for information systems. “Information systems” here include programs, hardware, data, transformations and transportation of data, people interpreting and producing the data. The data, and the information that people have, refer to phenomena in problem domains.

Abstraction is useful both when analysing problem domains and when designing the software of an information system. However, abstraction for analysis and abstraction for design are seldom separated, which may lead to a confusion of the direction of abstraction and concretizing.

A typical illustration of abstraction during analysis of the problem domain is found in (Olle et al, 1991). Olle et al state that “trading partner” is on a higher abstraction level than “supplier” and “customer” (p.58), and that users may comprehend a lower abstraction level easier than they understand a higher level.

Abstraction for design can be illustrated by a history of software abstraction as presented by Wirfs-Brock et al (1990, pp.4–5): the first abstraction was from bits to assembly language, then grouping instructions to macros and naming them, defining machine-independent high-level languages, grouping instructions into procedures, and defining abstract data types; the latter denotes a representation-independent specification of a data type that can be implemented in several ways. The encapsulation in object-oriented programming languages is abstraction in this sense: the specification is accessible from the outside, while the implementation is hidden. Abstraction in Wirfs-Brock et al’s history deals with removing details of the computer from the software tools that are used for implementing application systems. Contrary to the analysis, users would probably understand a high level of abstraction in the design sense, eg, “customer,” easier than a low level, eg, the ASCII code.

A low level of abstraction in analysis of problem domains is therefore not the same as a low level of abstraction during software design. This ambiguous meaning of “abstraction” illustrates the need for clarifying the concept, so that frameworks of information system concepts and techniques for development can distinguish and compare different types of abstractions without mixing them up.

The common type of abstraction used during design is to separate layers of specification and implementation. The analysis of problem domains can benefit from using a similar distinction between roles and realizations. However, this is not commonly done, and the result is that concepts or objects that should have been more abstract than others are modelled as more concrete.

Abstraction relations concerning the formal aspects of information systems have been discussed in the literature (Bergheim et al, 1989). This is useful for design and implementation of software, and it is in line with Wirfs-Brock et al’s history. Abstraction relations of the problem domain have also been discussed (Haugen, 1980; van de Weg and Engmann, 1992).

However, these do not relate abstractions in the problem domain and abstractions in the implementation of the information system, so that the ambiguity remains.

This paper aims to develop an unambiguous approach to abstraction that includes both analysis of problem domains and design of information systems. To achieve this, the “meaning triangle” constitute the basis (Section 2). Two types of abstraction that relate the analysis and design approaches are discussed; representation (Section 3 – 4) and role-realization (Section 8 and 10). In addition to these concepts that are rarely considered in the information systems literature, the traditional relations of classification, generalization, and aggregation are briefly presented in Section 5 – 7. The abstraction relations are summarized in Section 9.

The corresponding view of information system (Section 10) and the resolution of the ambiguity of “abstraction” (Section 11) are presented. Implications of the approach on definitions of “abstraction” in conceptual frameworks (Section 12), and on abstraction mechanisms in object- and flow-oriented techniques for system development will be discussed (Section 13), and improvements proposed.

2 The meaning triangle

In order to cover the problem domain, the information system, and the way the information system represents the problem domain, a simple model of representation of reality is chosen as the point of departure. In the model, an expression expresses a meaning, the meaning refers to an extension, and the expression represents the extension. These relations are depicted as a triangle, according to Charles Kay Ogden (1889–1957), see Figure 1.

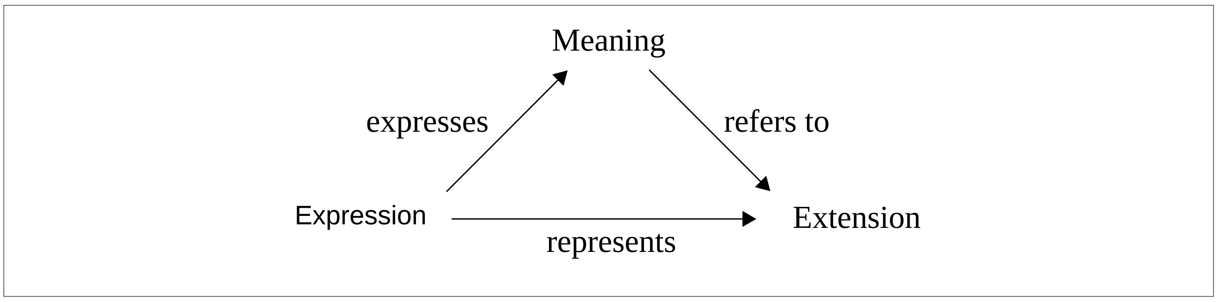

Figure 1: “Ogden’s triangle”: The relation between language and reality

The expressions consist of signs and symbols in the form of printed matter, sounds, contrasts on screens, electronic or magnetic patterns in computers. Extensions may consist of any phenomenon, also including expression and meaning. Hypothetical phenomena and future phenomena may also be included in the extension. Meaning is the relation that persons make between expression and extension. Examples of meaning are the interpretation of an expression and the intention while carrying out a speech act.

When saying for short that an expression e represents a phenomenon p as its extension, it is assumed that the meaning that relates e and p is provided by relevant persons. These may be a language group if the expression is a word, or group of persons communicating if the expression is relevant for the group. We will therefore say for short that an expression represents a phenomenon, implicitly assuming the existence of appropriate persons who provides the relation through their meaning.

The following typography will be used when separation is needed:

“Relation between expression and extension”

“Expression”

Extension

3 Representation

> An expression *represents* an extension if the meaning of the expression refers to the extension.

The relation between expression and extension is the *representation relation,* also for short *representation.*

When using the representation relation, extensions do not have to be present in a speech act, expressions are sufficient. The test to judge whether a relation is a representation relation is to ask: can the expression be separated in time and space from the phenomena which they represent? If the answer is no, there is no representation relation.

Since abstraction is to ignore aspects and concentrate on those aspects that are relevant, representation is abstraction, on the condition that the expressions express the relevant aspects of the extension.

Language is the most developed social means of representation. Since the relation between phenomena in the world and the expression depends on the linguistic conventions of the persons uttering or interpreting the expression, these persons are in the position of determining whether the relation holds.

Information systems process expressions through, eg, transforming and transporting data. The people working in an information system, and those whom the data concern, determine the meaning of the expressions in the system, and hence the extension represented by the expressions. When the expressions in the information system represent an extension, we will also say that the information system represents this extension. The extension of an information system is often called the “problem domain” or the “universe of discourse.” Eg, parts of the expression at a flight ticket represent a flight journey when the persons issuing and controlling the ticket interpret it that way, see Figure 2. Correspondingly, the reservation system also represents the flight journeys, and the flight journeys constitute the problem domain (or the universe of discourse) of the system.

The opposite of “to represent” is *to be represented by.* The journey is represented by the ticket.

Representations is a type of *abstraction relations.* The process starting with an extension, and associating an expression with it, is the *representation process,* which is an *abstraction process,* ie, a way of *abstracting.* The opposite of “an abstraction process” is *a concretizing process*, thus “to be represented by” is to concretize, eg, the journey concretizes the expression at the ticket.

4 Representation relations in the problem domain

Since extensions can be of any kind, there exist representation relations that have expressions or other representation relations as their extension. Assume that the pilots have the procedure knowledge written up as a text, which they follow during safety check of the airplane. Since the written procedure is an expression that represents each of the checks, there is a representation relation between the text and a check. Assume also that the computer system has the procedure

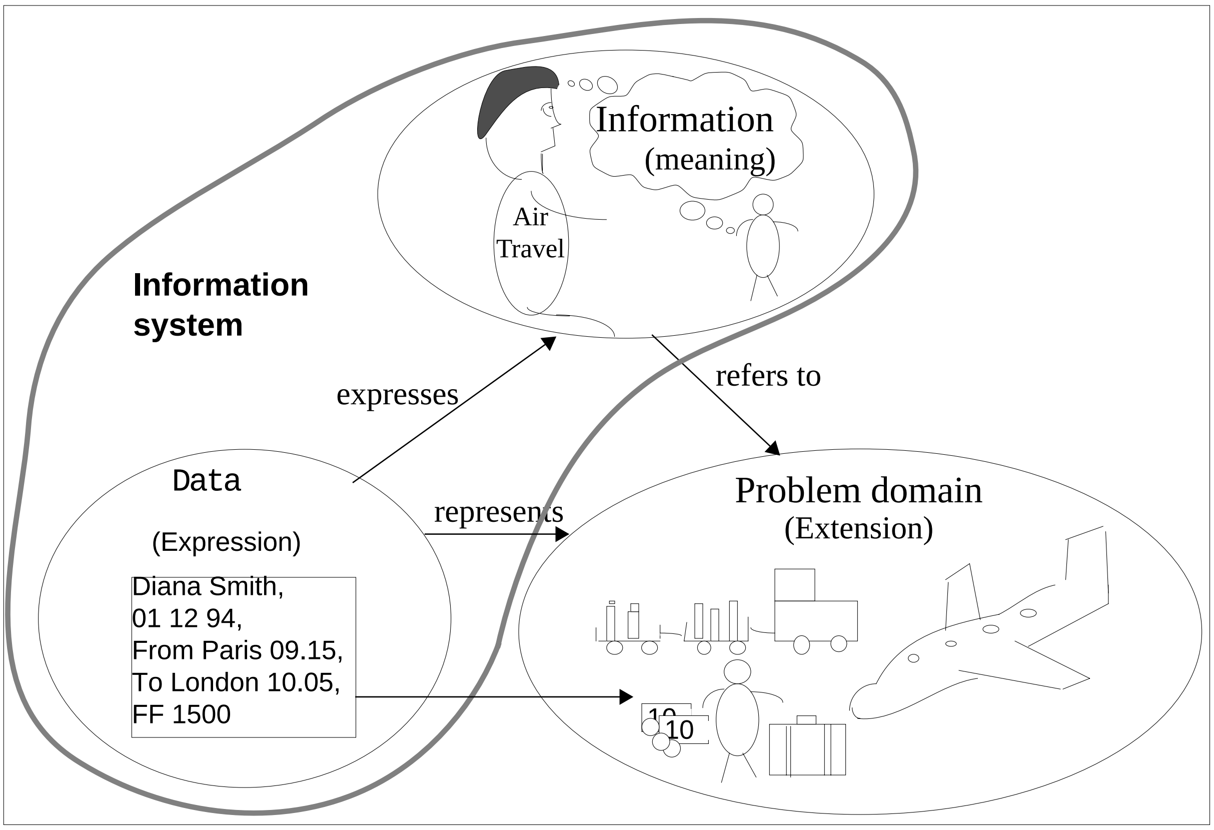

Figure 2: An information system consists of data that express the information of people in the system. The information system represents a problem domain.

in a knowledge base and, that this knowledge is used for control of the safety data. Then there is a representation relation inside the computer system that represents a corresponding representation relation in the problem domain.

In general, assume the following conditions:

Table 1:

	Example
• There exists a phenomenon p	safety check
• There exists an expression pe in the problem domain, and pe represents p	The procedure knowledge written up as a text represents safety checks
• There exists an expression e in the information system, and e represents p	safety check data represents safety checks
• There exists an expression ee in the information system, and ee represents pe	the procedure in the knowledge base represents the procedure knowledge

The representation relation “ee represents pe” *represents* the representation relation “pe represents p.”

A criterion for the existence of a representation relation in the problem domain is that there exists an expression that can be separated in time and space from the phenomenon that the expression is about.

Knowledge based systems and expert systems represent general knowledge from the problem domain. The knowledge representations in the knowledge base is applied to specific data that represent phenomena in the problem domain.

5 Classification

This abstraction relation and the two following ones, generalization and aggregation, are assumed to be well known; thus they are not motivated and discussed. For motivation and discussion, consult textbooks on object-oriented analysis (Hutt, 1994) or articles on abstraction (eg, van de Weg and Engmann, 1992). The reason for presenting the relations here is to define them in the context of the meaning triangle.

For classification, assume the following conditions:

Table 2:

	Example
• There exists phenomena p1, p2, …	Diana Smith’s journey from Paris to London 01 12 94, Tom Jackson’s journey from …, …
• There exists a class P consisting of p1, p2, …	Journeys from Paris to London 01 12 94 consists of Diana Smith’s, Tom Jackson’s, …
• There exist expressions e1, e2, … which have some common parts E	“Diana Smith, 01 12 94, From Paris 09.15, To London 10.05, FF 1500”; “Tom Jackson, 01 12 94, From Paris To London 10.05, FF 1500” have the common part “01 12 94, From Paris To London 10.05, FF 1500”
• c1 is the representation “e1 represents phenomenon p1”; c2 is the relation “e2 represents p2,” …	Diana’s right to go by plane is the representation “Diana Smith, 01 12 94, From Paris 09.15, To London 10.05, FF 1500 represents Diana Smith’s journey from Paris to London 01 12 94”; …
• C is the representation “E represents a class P”	The right to go by plane from Paris to London 01 12 94 is the representation “01 12 94, From Paris To London 10.05, FF 1500 represents journeys from Paris to London 01 12 94”

Then classification is defined:

C is a representation that *classifies* c1, c2, ….

Since only the common parts of e1, e2, … are included in E, there are other parts that are left out, hence classification is abstraction. Since the parts of the expressions that are excluded in classification differ, these parts represent the differences between the phenomena of the extensions. Hence classification disregards differences while focusing on similarities.

The opposite of classification is *instantiation*, and c1, c2, … are *instances* or *examples* of the representation C.

In summary, the expression “classification relation” has the following extension: relations between representations with singular extensions and representations that have classes as extensions.

6 Generalization

> Given two representations C and B, B is a *generalization* of C if all instances of C are also instances of B.

Eg, “the right to travel” is more general than “the right to go by plane,” because all instances of “the right to go by plane” also instantiate “the right to travel.”

The opposite of generalization is *specialization*.

Since B may have instances that are not Cs, the common parts of the expressions of the Cs cannot be common for all B’s. There has to be less common parts of the expressions of C than those of B. Thus some aspects of C are not aspects of B, hence generalization is also abstraction.

“Generalization relation” is thus defined to have the following extension: relations between representations and other representations with larger extensions.

7 Aggregation

Given the representations

r1: the expression e1 represents phenomenon p1,

r2: e2 represents p2, …

and the representation

R: the expression E represents the composition of p1 and p2 and …,

> R *aggregates* r1 and r2 and … into an aggregate representation. Conversely, r1, r2, … *segregates* R into *details*.

Eg, “Diana Smith, 01 12 94, From Paris 09.15, To London 10.05, FF 1500” and “Diana Smith, 01 12 94 From London 11.15, To New York 13.00, FF 3000” aggregates to the aggregate “Diana Smith, 01 12 94, From Paris 09.15, To New York 13.00, FF4500”

Aggregation is commonly associated with real world phenomena and not with concepts. “Aggregation” is given a more general denotation here, also covering the aggregation of representations. Saying that one representation C aggregates c1 and c2, means that each instantiation of C has an extension that is a composition of the extensions of c1 and c2. Eg, saying that “floor,” “walls,” and “ceiling” aggregates into “room,” means that every instantiation of “room” has an extension that is a composition of extensions of “floor,” “walls,” and “ceiling.”

Aggregation differs from classification because the representations that are classified have to be similar in some respect, while similarity is no condition for aggregation, eg, “room.”

The expression “aggregation relation” has the following extension: relations between representations r1, r2, … and another representation with an extension that constitutes the composition of the extensions of r1, r2, …. Since the aggregate ignores the characteristics of its details, aggregation is abstraction.

8 The role-realization relation

Classification, generalization, and aggregation relate representations of any phenomenon in the problem domain. The role-realization relation exists for those phenomena in the problem domain where at least two aspects can be identified. One aspect, the realization, is closer to physical matter than the other aspect, the role. For the role to exist, the realization also has to exist. Assume in general:

Table 3:

	Example
• There exists a phenomenon p-role in the problem domain	Diana Smith's actions and properties related to being a passenger
• Another phenomenon p-realization has also to exist for p-role to exist.	The person Diana Smith
• Ro is the representation "e-role represents p-role"	Diana the passenger is the representation "Diana Smith, reservations, bonus points represents Diana Smith's actions and properties related to being a passenger"
• Re is the representation "e-realization represents p-realization"	The identification of Diana Smith is the representation "Diana Smith, female, 11 Home Road represents the person Diana Smith"

The role-realization relation is defined:

Ro *is a role of* Re. The opposite way: Re *realizes* Ro.

The process of going from realization to role is called role-derivation, while the opposite direction is realization. Whereas separability in time and space is a criterion for identifying representation relations, roles exist in the same time and space as their realizations. Assuming that Diana Smith is also a pilot in the airline, both the pilot and the passenger are roles of the person. When Diana Smith disappears, so do both her roles too.

In addition to persons realizing roles, phenomena that can be interpreted in symbolic ways also realize roles. For example, ink on paper realizes a text, a video cassette realizes a movie. For artificial realizations like a movie cassette, there may exist many copies realizing the same role. One copy is necessary for the movie to exist. The necessity of the existence of the realization has not been considered in literature on role relations (Richardson and Schwarz, 1991; Coad, 1992; van de Weg and Engmann, 1992; Goldstein and Storey, 1994). In this paper, the existence criterion is important for deciding which aspect of a phenomenon is the role, and which is the realization. However, due to that the purpose here is to consider the direction of abstraction, neither the existence criterion will not be discussed further.

When separating roles from realizations, the aspects of the realization are ignored, hence roles are more abstract than their realizations.

An objection may be that the roles carry more information than the realizations, such that more aspects are considered when focusing on the roles rather than on the realizations.

However, when the roles constitute the relevant issues, one can abstract away from the aspects of the realization, even if those are only a few.

Counter to this, one can say that when the realization is in focus, the argument will work the other way. However, since roles are conceptual while the realizations is closer to the physical world that can be sensed, more aspects of realizations can be found more easily than can aspects of roles.

A common way to make object-oriented models of roles, is to mix roles with specializations, see the example from (Odell, 1992) in Figure 3. Since the general class Person in Odell's model is more general than the classes Man, Woman, and Employee; Person is more abstract than Employee and other possible roles of a person. The same direction of abstraction is also suggested by Coad (1992) and van de Weg and Engmann (1992). This direction of abstraction is contrary to the direction that follows from the role-realization relation, as illustrated in Figure 4.

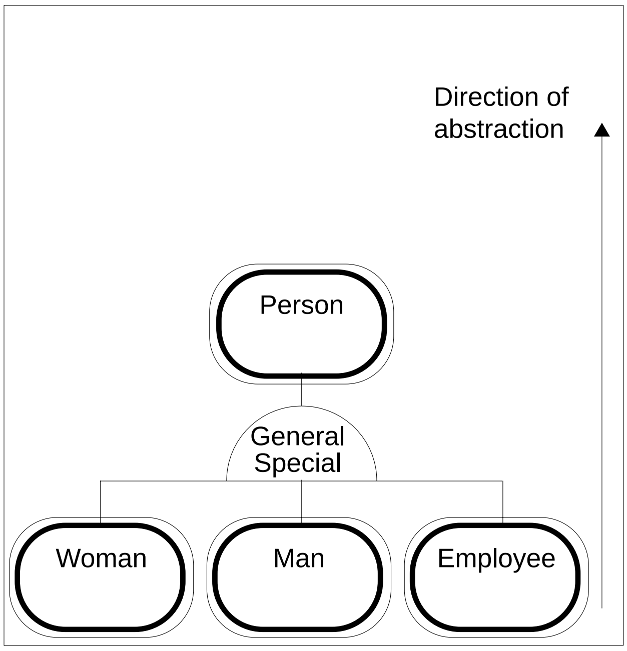

Figure 3: Mixing the role-realization relation with generalization according to Odell (1992)

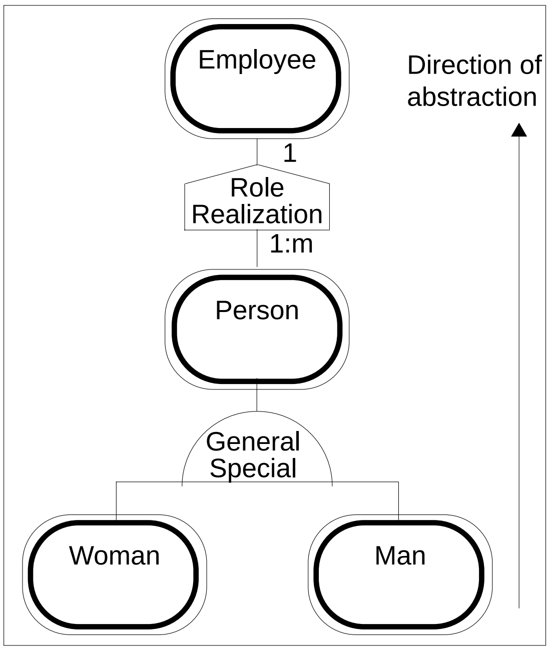

Figure 4: The role-realization relation and generalization with the proper direction of abstraction

Expressions consist of signs and other symbols, which is the subject of computer semiotics (Andersen, 1990). The form is the distinctive properties of the expression, while its substance is a particular way of realizing the form (Andersen, 1992, p.17). The form-substance relation is a specialization of the role-realization relation for expressions. When designing a text processor or other programs having expressions in their problem domain, several layers of form and substance may be needed.

9 Relations between abstraction relations

The expression "abstraction relation" has the following extension: relations between phenomena with aspects and phenomena where only some aspects are included. There is no limitations on what the phenomena may be. "Abstraction relation" is the concept relating the expression to the extension presented in this paragraph.

It has been argued that the representation, generalization, classification, aggregation, and role-derivation relations are abstractions. This means that every instance of these relations also instantiate "abstraction relation." Thus, these relations are specializations of "abstraction relation," and "abstraction relation" is a generalization of these relations, see Figure 5.

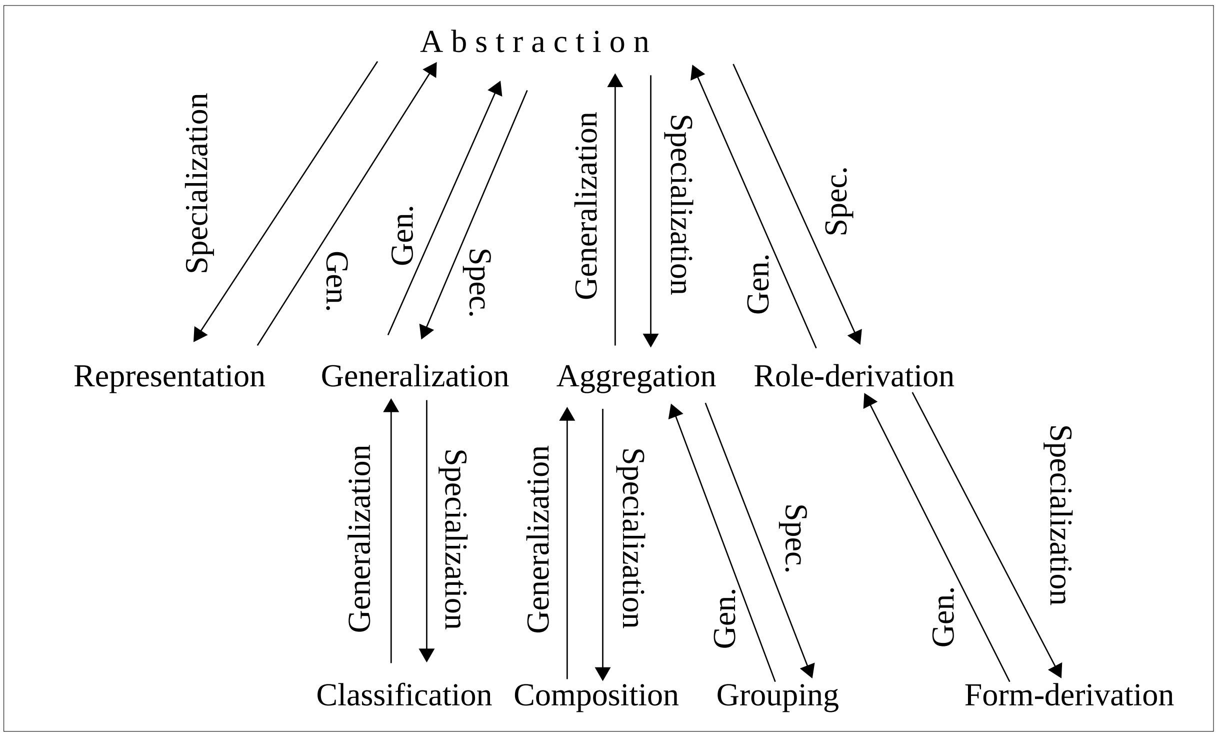

Figure 5: Abstraction relations: Generalization and specialization relations between abstraction relations.

The corresponding concretizing relations are illustrated in Figure 6.

In the following, it will be argued that there is also a generalization relation between classification and generalization.

Generalization relations are between representations and other representations with larger extensions. Classification relations hold between representations with singular extensions and representations that have classes as extensions. Assuming that a common opinion of classes is that classes are larger than a singular entity of the class, classification relations are also relations between representations and other representations with larger extensions. Hence, generalization is more general than classification.

Classification and generalization are often regarded as unrelated mechanisms. As an exception, Tsichritzis and Lochovsky (1982) use the expression "token-type generalization" to denote classification (p.17). However, they neither argue for the relation, nor do they identify it as a generalization relation.

Specializations of aggregation have also been considered in literature (eg, van de Weg and Engmann, 1992). A transitive *composition* of whole from parts can be distinguished from a *grouping* of elements in a set (Motschnig-Pitrik, 1994), see Figure 5 and 6.

10 Role-realization relations in the information system

Since information systems handle expressions, the role-realization relation is useful for

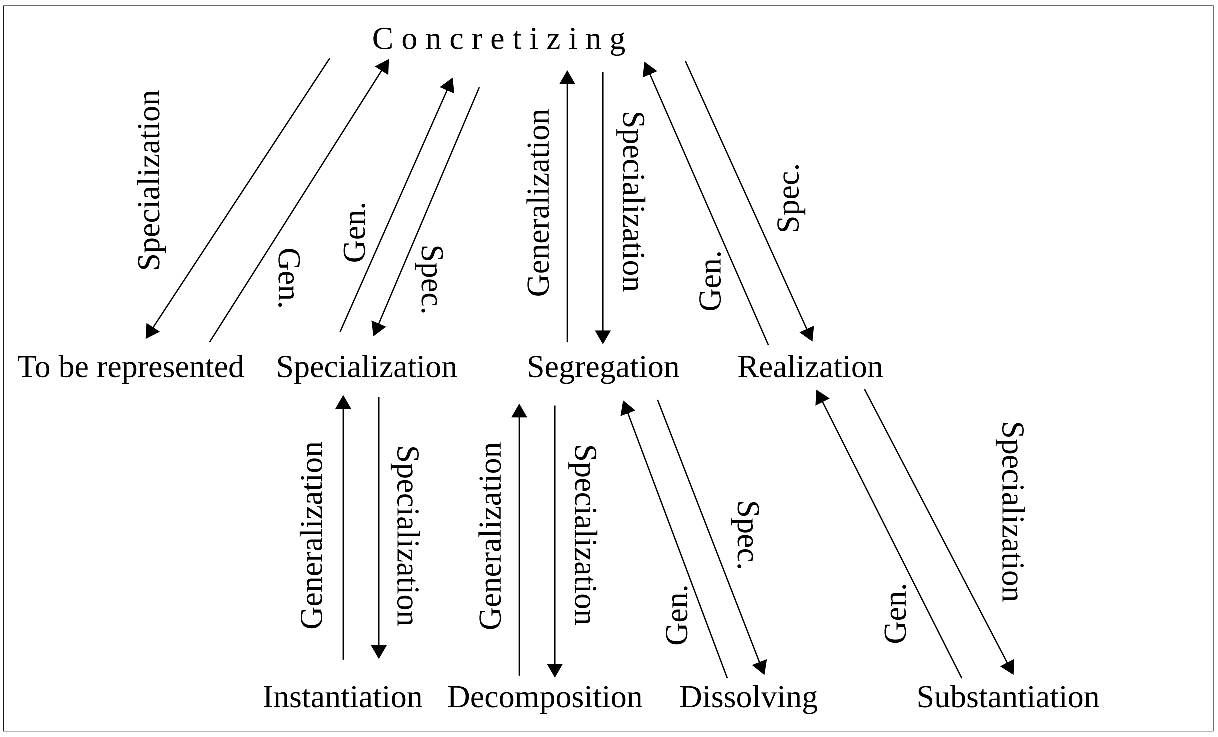

Figure 6: Concretizing relations.

structuring information systems. The expression “Diana Smith, 01 12 94, From Paris 09.15, To London 10.05, FF 1500” can have a computer screen and a printed airline ticket as two different substances. The form is the shape that is invariant and recognized as the same meaning regardless of substance. Layers of realization can often be identified. Eg, black and white contrasts on computer screens can be substantiated with CRT and LCD substance.

Information systems do not only store data, they change data as well. The realization relation between form and substance can be extended to the transformation of data. Eg, when deleting a passenger from the database of scheduled flights, the selection of the passenger and of the text `Delete` that appears in a menu constitutes the form of the transformation. This form may be realized through, eg, pointing and clicking, or through pressing command keys. These realizations constitute substances.

Programs and data are textual or graphic expressions, hence the form-substance relation applies between their form and the substance. The meaning of a program is recognized through its functionality, which includes its data and operations. The functionality is expressed in a specification, which can be implemented in different codes. Since the specification of a piece of software expresses its meaning, the specification constitutes its form, while the code is the substance of the program. Therefore, there *specification* is realized in *code*. This special realization relation is called *implementation*, while the opposite is called *specification-derivation*.

Layers of specifications and code are found in computer systems too. Eg, text is implemented in ASCII code, which in turn is implemented in electrical currents and magnetic fields. Program text is often even more partitioned into layers, because this is assumed to enhance flexibility. These examples concern the computer systems as seen from a programmer’s view. The above example of passenger and menu selection illustrates realizations of the program seen from a user’s point of view.

Assume an implementation relation, known to be correct, between a specification and a code. When the code is executed, it is highly predictable that the corresponding specification is followed, because errors tend not to appear at random.

The users also have to master realization relations. They have to know that to make the computer fulfil a specific function, specific behaviour is required. This is part of the skills required to master a system. However, people's knowledge and behaviour is not predictable to the same extent as computer processing. A user may push a button that triggers an unintended function. Therefore, there is a weaker regularity between the user's knowledge of the function and its implementation than between specification and implementation in the computer. Similarly, the relation between the form of a user's action and the substance of her/his behaviour is not as predictable as the realization relation of computers. Therefore, there exist other specializations of realization in addition to implementation. Three specializations are presented in (Kaasbøll).

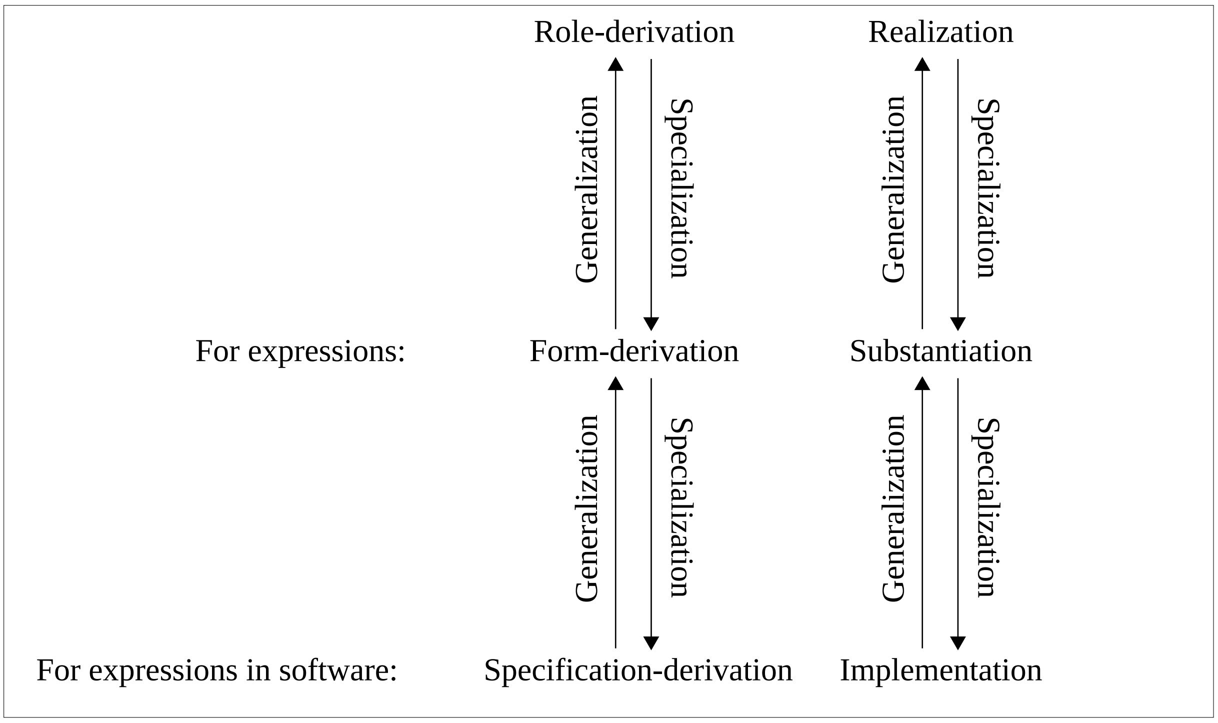

Figure 7: Roles relations and realization relations in information systems

To be able to deal with only one concept, the realization relation is defined in general for both computer processing and people's behaviour to be a regular correspondence between form and substance of signs, data, transformations of data, and categories of these. The collection of all code and the people, machines, documents, etc. that carry the code is called the *ensemble* of the information system. As in the theatre, the ensemble realizes the play. The realization relation in information systems is illustrated in Figure 8.

11 Separating abstraction in problem domains from abstraction in the information system

Since realization in the information system does not interfere with abstraction and concretizing in the problem domain, the ambiguity of direction of abstraction presented in the introduction can be clarified.

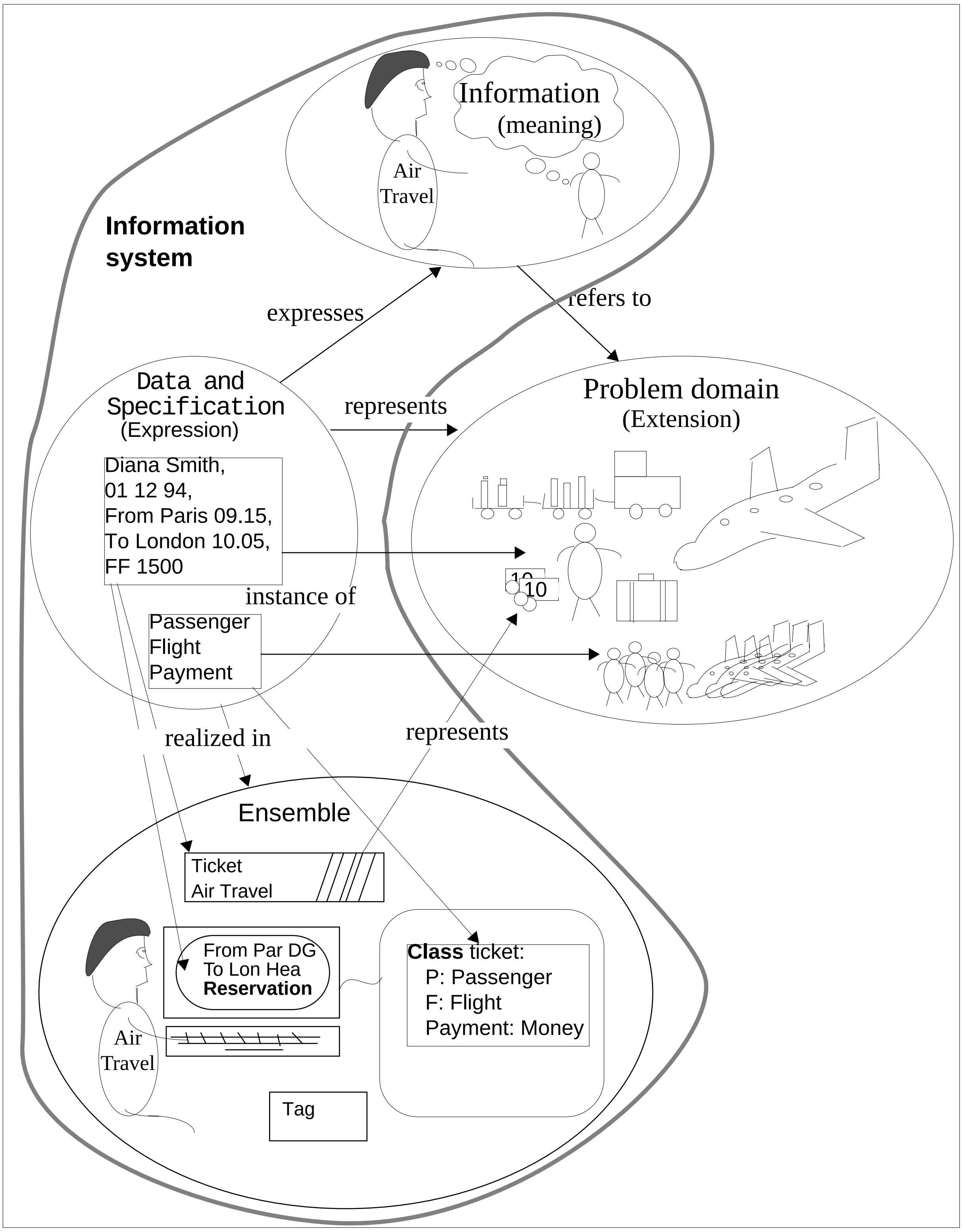

Figure 8: Representations and realizations in information systems

Confused direction of abstraction. Given an information system that represents a problem domain, eg, a flight reservation system. From the travel of Diana Smith from Paris to London, we can abstract "travels from Paris to London," from where we can abstract "travels." Then the question is, will it be more abstract or more concrete to go from "travels" (or from "Diana Smith, 01 12 94, From Paris 09.15, To London 10.05, FF 1500") to program code, to 1011 0101 0011 1001 …, and to electrical currents, light pulses, and magnetic fields? In the sense that abstraction is to ignore some aspects while focusing on others, electrical currents is certainly ignorance of everything concerned with flights, and focus on one issue. On the other side, electrical currents are more concrete than is a general concept like "travel." Thus there is intuitive support for saying both that realization in the information system is abstraction and concretization. Since both choices of direction of abstraction corresponds to some intuition, but runs counter to another, separating abstraction in the problem domain from abstraction in the information system solves the confusion.

Confusion between users' and developers' perspectives. Since both users and developers strive for an accurate representation of the problem domain, they will have to deal with the same abstraction relations. When designing and implementing a solution, the developers will have to consider implementations in the information system. For periods of time, different layers of implementation may constitute the domain of the developers, making it reasonable to consider implementations in the computer as concrete, and specifications of applications as abstract. "Abstract data types" (Guttag and Horning, 1978) are examples. As long as the developers separate their technical sense of abstraction (specification-derivation and implementation) from abstractions in the problem domain, confusion is avoided. But if the developers mix up the two, they also mix their own technical view of an information system with the users' perspective.

The implementation relation is independent of the problem domain. Since implementation has no connection to the problem domain; the same implementations can be used for expressions representing any problem domain. Thus, by defining substantiation and implementation as relations between expressions only, the incoherence between the information system and the problem domain is taken care of.

Parts of the code may also represent. When analysing the information of a ticket, some of the printed text represents a travel. In addition, the material that the ticket is made of also represents phenomena in the problem domain. The specific paper and colours is a contract giving the passenger rights towards the airline company. The text at a photo copy of the ticket still represents the travel, while the copy is not representing the passengers rights. If developing a paperless computer based system for reservation and travel, the representation relations to the passengers rights must be taken care of. The functionality of the system is independent of the layer of realization where the expressions of the current system are found. Even if these representations were expressed by codes in the ensemble of the old system, it may be appropriate to make a textual expression in an electronic version of the system.

If the realization in the information system was mixed up with abstractions in the problem domain, those developing the system might have transferred the relation to the new system, knowing that the abstraction relations that corresponds to the problem domain should be transferred.

12 Related approaches

This section discusses two approaches to concepts for information systems, aiming to suggest ways to improve both these approaches and the current exposition of abstraction. The approaches are selected because they also aim at conceptual clarification.

12.1 A taxonomy of concepts for the information systems science

Bergheim et al (1989) have developed a taxonomy of concepts for formalisms applicable to information systems and their development. The taxonomy separates between abstraction at the "application level" (α-level) and at the level of the languages used for making models and programs (β-level). The α-level corresponds to the expressions in the model of information systems developed in this paper. In our terms, β-level expressions represent α-level expressions.

The general sense of abstraction in their taxonomy, "to suppress irrelevant details in order to emphasize on essential details in some context," is similar to the one adopted in this paper. Several types of abstraction are mentioned: "generalization, composition, procedural abstraction, control abstraction, abstract data types," etc. (Bergheim et al, p.283). We have already stated that "abstract data types" is an implementation relation.

They speak of α-abstraction levels, depending on the amount of details in the α-model. If levels of expression constructed with aggregation, categorization, and generalization (as defined here) were identified, the expressions in the highest levels of abstractions would have less details than those of the lower. Hence, these three types of abstraction can be defined within the α-level of Bergheim et al's taxonomy. This would enrich their formal taxonomy, since they do not consider thoroughly their own suggestions for abstraction relations.

Their "β-models are β-abstraction of each other to cope with the details of the computer" (p.284). The hardware is at the lowest level of β-abstraction. Hence, their β-abstraction seems to correspond with the specification-implementation relation. Since Bergheim et al intend to cover also tools for development (at their β-level) with their taxonomy, the software system is the problem domain of the tools. Then it is in accordance with the information system/problem domain distinction to use abstraction to characterize the internal relations in the computer. The lesson to be transferred to the information system/problem domain distinction is that the models of the computer needed to produce software development tools have to include abstraction relations, while the information system models and programs developed by these tools should include specification-implementation relations between the expressions and the layers of codes.

12.2 FRISCO

The Framework of Information Systems Concepts (Lindgreen, 1990) exemplify "abstraction" like this: "abstraction is applied when … a bee, an elephant, a snake etc. are regarded as animals" (p.73). Instantiation is mentioned as the opposite relation. "Abstraction" in FRISCO thus corresponds roughly to classification. "Generalization" in FRISCO is defined in close accordance with how it is defined here.

FRISCO aims to identify conceptually what “information systems” are. The identification starts from the general concept “system.” “System” is first specialized into “open active system,” which in turn is specialized into “organizational system,” and then to “formalized organization system.” The second step is called “organizational abstraction” (p.30). However, since the class of “organizational systems” is smaller than the class of “open, active systems,” the operation carried out is a specialization. It is thus neither abstraction nor generalization, nor is it in line with the FRISCO definition of “abstraction” as classification referred above. These parts of the framework are therefore in need of clarification.

The way of specializing “systems” to “information systems” carried out in FRISCO may also be an appropriate way to better specify the concept of information system illustrated in Figure 9.

The “formalized organization system” is a system in which all activities are carried out in a reproducible manner. A “formalized organizational system” may consist of all the aspects illustrated in Figure 9. Compared to the view of information systems here, the requirement of reproducibility limits all information to be interpreted according to rules given.

The next step in FRISCO is called “semiotic abstraction.” Semiotic abstraction consists of disregarding all material matter except data and its physical carriers. This corresponds to disregarding the problem domain, and the resulting “information realization system” corresponds roughly to the expressions and the ensemble in Figure 9. Since semiotic abstraction is to disregard the extension while focusing on expressions, semiotic abstraction is representation according to the concepts in this paper.

The final step in FRISCO consists of “infological abstraction” and “datalogical abstraction,” which here is interpreted to mean separating the form of expressions (“infological”) from their substance (“datalogical”).

FRISCO also say that “data represents information.” In the terms used here, data expresses information. The representation relation is found in FRISCO’s triadic “sign relation,” which corresponds to Ogden’s triangle. However, FRISCO does not consider the extension of the data.

13 Techniques for analysis and design

Techniques for systems analysis and design are evaluated regarding their coverage of abstraction in the problem domain and the information system, and the evaluation is summarized in Table 1 and 2. Three methods with distinct approaches are selected. Formal arguments are not provided, since the methods are assumed to be well-known. Suggestions for improvements are made.

13.1 Jackson System Development

Jackson System Development (JSD) was an early approach to modelling according to objects, called “entities” in the method (Jackson, 1983). JSD starts out with making a description of the problem domain of an information system. The method further advises to make a model of the data used to represent that domain, and the relations between the problem domain (“level 0”) and the data (“level 1”). The descriptions may be extended to cover realizations of the data too (“levels 2, …”). JSD does not introduce any conceptual difference between representation and realization. The difference could probably easily be achieved if the two relations were introduced conceptually in the method, and “level 0” was called “problem domain”; “level 1” called

Table 4: Analysing abstractions in the problem domain

	JSD	OOA/D	Dataflow
Representation	No	No	No
Classification	The classification process is supported, but not the relation	The classification process is supported, but not the relation	No.
Generalization	No. Could be introduced	Yes	No. Captured when extended with data models
Aggregation	Of events only. Could be introduced.	Of objects	No. Captured when extended with data dictionaries or data models.
Role-Realization	No	No	No

"expression"; and "level 2" and further levels called "layers of realization".

Classification of objects into classes is supported in the method, but there is no notation for describing a classification relation. However, it may be sufficient to say that all objects are treated as if they were instances of a class.

Unlike the object-oriented methods appearing around 1990, JSD does not support generalization. This could probably be remedied with a "gen/spec" relation similar to Coad and Yourdon's (1991a) technique. Additional rules for specifying how to merge event sequences when such sequences are inherited, would have to be provided. A proposal for such merges is found in (Mathiassen et al, 1992 and 1993).

The method has notation for aggregation of events, covering the common ways of structuring an algorithm. Aggregation of objects or classes is not supported. This could also be introduced, and the merging of event sequences could be carried out as for inheritance.

Table 5: Designing abstractions in the information system

	JSD	OOA/D	Dataflow
Representation of problem domain	Yes. The problem domain and relations to the data that represents. Notation can be improved.	No. The problem domain and the data that represents are described separately, but the representation relation is not captured.	No. Can be achieved if "material flow" and its relation to "logical" diagrams is included
Role-realization	Implementation can be expressed in layers. Notation should be improved to separate realization from representation.	Implementation in one layer. Could be extended to any number of layers.	Implementation in one layer. Can be extended to several layers, provided a suitable visualization.

13.2 Object-oriented analysis and design

Coad and Yourdon's (1991a and b) method for object-oriented analysis and design (OOA/D) contain guidelines for analysis of the problem domain of the information system. In design, the description is to be interpreted as a model of the expressions in the computer. OOA/D therefore describes both the problem domain and the expressions that are going to represent it. However, because the method shift interpretation of the description instead of making separate descriptions, it becomes impossible to capture the representation relations between the problem domain and the expressions in the computer. Keeping the relations as in JSD could be done.

Similar to JSD, OOA advises classification, but the notation gives only room for the class, not for the relations between class and object. Generalization relations can be described through a "gen/spec" relation. Objects can be aggregated by means of a "whole-part" relation. In addition, objects can be aggregated in one level into "subjects."

During design, the model is implemented in user interface and in data base and task management implementations. Additional layers of implementation are not included. There seems to be no good reason why the method does not allow for any number of layers. The proposed realizations into user interface, data base, and task management may be practical solutions in many cases, but these could be specialized suggestions in a general mechanism for realizations.

13.3 Structured analysis

Structured analysis (DeMarco, 1978; Yourdon, 1989) consists of a series of techniques; here only dataflow diagrams are considered, since they constitute the core of the method. Dataflow diagrams describe "logical" and "physical" data processes, while "material processes" should be excluded in the dataflow technique. "Material processes" are what the data is about, ie, the problem domain. The representation relation is thus not included in dataflow diagrams. Through including material processes, the relation could be described, which is done in, eg, activity graphs in ISAC (Lundeberg et al, 1981). Dataflow diagrams are also extended with data dictionaries or data models. Data dictionaries focus on aggregation. Some dialects of data models capture aggregation and generalization.

The "logical" diagrams describe the expressions in the system, while the "physical" diagrams depict the realization in one layer. However, the notation to show the relation between physical and logical is poor. The number of layers could be extended through introducing a "format" layer between the logical and the physical. A notation or visualization of the realization relations for each element in the layers is necessary for working with layers.

A process in a diagram can be expanded to show its interior, which may consist of more processes flows, and stores. The expansion is a mechanism for aggregation of the processes in the information system. However, since this paper is focused on abstraction of representation relations, and the interior of information systems consist of expressions only, a discussion would be necessary to consider abstraction relations between expressions. This is outside the scope of this paper.

13.4 Summary of evaluation of techniques

OOA/D supports generalization, aggregation, and classification in analysis of problem domains. JSD could be extended with similar mechanisms. Dataflow diagrams have poor support for analysis of problem domains. Their abilities to model aggregates of processes may be useful during design.

14 Conclusion

This paper aimed at distinguishing and comparing different types of abstractions in problem domains and information systems. In addition to the well-known classification, generalization, and aggregation, two rarely considered abstractions have been discussed: representation and role-realization.

Roles and realizations often occur in problem domains. When defining the relation properly within the meaning triangle, it has been shown that models of roles and realizations found in literature erroneously indicated that realizations were more abstract than roles.

Expressions acquire a form that is realized in substance. The form-substance relation is a specialization of the role-realization relation for expressions. This relation is further specialized for software where a specification is realized (also called implemented) in code.

When abstraction in the problem domain is mixed up with implementation relations, the direction of abstraction become confused, and users' and developers' perspectives are mixed.

Both conceptual frameworks and techniques for modelling information systems can be improved through clarifying the different abstraction relations.

Implications

System development techniques have been internalized by the developers using the techniques, and CASE tools may strengthen current practice and knowledge. Even if new methods that are more conceptually sound appear, those who are educated in accordance wiht the existing methods will probably stay within the modes of thought of these methods. It may be easier to change the CASE tools, so that they at least could allow for improvements of the methods and techniques. The CASE tools should be constructed to fulfil the following criteria:

- The tool represents the different abstraction relations both for the problem domain and the information system.
- The tool allows for defining notations that realize these relations as modifications and extensions of the techniques.

However, only changing computerized tools give little impact on knowledge and work practice (Sørensen, 1993). Improvements of system developers' knowledge about basic concepts like these relations should also be carried out.

Acknowledgement

Thanks to Kristin Braa, Joan Greenbaum, Renate Motschnig-Pitrik, Jan-Erik Ressem, Ole Smørdal, and the anonymous referees for discussions and comments on the paper, and to Sara Selmark for improving the language.

References

Andersen, Peter Bøgh (1990)*A Theory of Computer Semiotics: Semiotic Approaches to Construction and Assessment of Computer Systems* Cambridge University Press

Andersen, P.B. (1992) “Computer Semiotics” *Scandinavian Journal of Information Systems* Vol.4, pp.3–30

Bergheim, G.; Sandersen, E.; Sølvberg, A. (1989) “A taxonomy of concepts for the science of information systems” In Falkenberg, E. and Lindgreen, P. *Information system concepts: an in-depth analysis* Proceedings of the IFIP TC 8/WG 8.1 Working Conference on Information System Concepts (Namur, Belgium, 18-20 October, 1989) Amsterdam, North-Holland, pp.269–321

Coad, Peter (1992) “Object-Oriented Patterns” *Communications of the ACM* Vol.35, No.9, pp.152–159

Coad, P. and Yourdon, E. (1991a) *Object-Oriented Analysis* Second Edition, Prentice-Hall, NJ

Coad, P. and Yourdon, E. (1991b)*Object-Oriented Design* Prentice-Hall, NJ

DeMarco, Tom (1978)*Structured Analysis and System Specification* Yourdon, New York

Goldstein, R.C. and Storey, V.C. (1994) “Materialization” IEEE Transaction on Knowledge and Data Engineering, Vol.6, No.5, pp.835–842

Guttag, J.V. and Horning, J.J. (1978) “The Algebraic Specification of Abstract Data Types” *Acta Informatica* Vol.10, pp.27–52

Haugen, Øystein (1980) *Concepts of Hierarchies in Programming and System Description* (In Norwegian. Original Title: Hierarkibegreper i programmering og systembeskrivelse) Master Thesis, Department of Informatics, University of Oslo

Hutt, Andrew T.F. (ed.) (1994) *Object Analysis and Design: Description of Methods* Wiley, New York

Jackson, Michael (1983) *System Development* Prentice-Hall, New Jersey

Kaasbøll, J. *Types of explanation and prediction for information systems: Implications for system development techniques* Submitted for publication

Lindgreen, P. (1990) *A Framework of Information Systems Concepts* Interim report from the IFIP WG 8.1 Task Group FRISCO, IR.3-PL, University of Nijmegen

Lundeberg, Mats; Goldkuhl, Göran; Nilsson, Anders (1981) *Information systems development: a systematic approach* Englewood Cliffs, N.J.: Prentice-Hall

Mathiassen, L.; Munk-Madsen, A.; Nielsen, P. A.; and Stage, J. (1992) “Modelling Events in Object-Oriented Analysis” In Bjerknes, Bratteteig & Kautz (eds.) *Precedings of the 15th IRIS* Department of Informatics, University of Oslo, pp.742–757

Mathiassen, L.; Munk-Madsen, A.; Nielsen, P. A.; and Stage, J. (1993) *Object Oriented Analysis* (In Danish) Aalborg, Forlaget Marko

Motschnig-Pitrik, R. (1994) “Analyzing the notions of attribute, aggregate, part, and member in data/knowledge modelling” In Joze Zupancic and Stanislaw Wrycza (eds.) *Proceedings of The Fourth International Conference Information Systems Development — ISD’94 Methods & Tools. Theory & Practice* (Bled, Slovenia, 20–22 Sept., 1994) Moderna Organizacija, Kranj, 1994 pp.31–42

Odell, James (1992) “Managing object complexity, part I: abstraction and generalization” *Journal of Object-Oriented Programming* Vol.5, No.5, pp.19–22

Olle, T. W.; Hagelstein, J.; Macdonald, I.G.; Rolland, C.; Sol, H.; Van Assche, F.J.M.; Verrijn-Stuart, A.A. (1991) *Information Systems Methodologies: A framework for understanding* Second Edition, Addison-Wesley, Wokingham

Richardson, J. and Schwarz, P. (1991) “Aspects: Extending objects to support multiple, independent roles” *SIGMOD Record*, Vol.20, No.2, pp.298–307

Sørensen (1993) “What influences regular CASE use in organizations? — An Empirically Based Model” *Scandinavian Journal of Information Systems* Vol.5, pp.25–50

Tsichritzis, D.C. and Lochovsky, F.H. (1982) *Data models* Prentice-Hall, Englewood Cliffs, NJ

van de Weg, Rob L.W. and Engmann, Rolf (1992) “A framework and Method for Object-Oriented Information Systems Analysis and Design” In E.D. Falkenberg, C. Rolland, and E.N. El-Sayed (eds.) *Information System Concepts: Improving the Understanding* (Alexandria, 13–15 April, 1992) IFIP Transactions A-4, North-Holland, pp.123–146

Yourdon, Edward (1989) *Modern Structured Analysis* Yourdon Press, Englewood Cliffs, NJ