Validation of conceptual models through analysis of semantic content and explanation generation

Maria Bergholtz, Paul Johannesson

Department of Computer and Systems Sciences
Stockholm University and the Royal Institute of Technology
Forum 100, S-164 40 Kista, Sweden
Email: {maria, pajo}@dsv.su.se

1 INTRODUCTION AND BACKGROUND

This paper addresses the interpretation of conceptual models. These models are based on the underlying notion that the world can be represented, or rather viewed, as consisting of objects. These objects can then be grouped together into classes, they are associated with each other via different types of relationships, and they possess distinct properties. Conceptual modelling is the activity of *identifying* the concepts and phenomena that exist in some *part* or *aspect* of the world in order to *represent* them as the aforementioned building blocks of a conceptual model. To view the world in this way provides a powerful tool for representing systems in a structured and easily understandable way [Boman et al. 97]. Conceptual modelling has therefore been used in many different aspects. It has typically been carried out in an organisational context with the aim of analysing, designing and implementing an *information system* to support the activities of the organisation. Conceptual models have also been used for enterprise engineering, e.g. for clarifying and developing the mission and goals of an enterprise. A third example of the usage of conceptual modelling is reverse modelling of existing systems as a step in integration of legacy systems [Johannesson93] on many different levels such as view integration and database integration.

The conceptual model, in figure 1 represented by a very small fragment of an UML class diagram, and the choice of conceptual modelling language constitute one of the most important parts of the various methods deployed during a systems development process. The model is not only base and input to later stages in systems development such as database implementation or design of user interfaces, Figure 1. It is also, and equally important, the key part/means for communication between on one hand the systems analyst and on the other hand the domain experts and users. The abstraction level of the modelling language must thus serve two, often contrary, purposes:

- It must be able to offer the systems analyst a way to capture all essential features of the Universe of Discourse in a complete and consistent manner
- It must be comprehensible enough to serve as a road map for experts knowledgeable in the domain but not necessarily in the modelling language.

The contradiction between these two desirable purposes can intuitively be looked upon as a language barrier, i.e. that domain knowledgeable persons does not understand the language used by the systems experts. In as systems development process the first, and perhaps most obvious, risk here is that the constructed model will not match what the different stakeholders require in terms of functionality of the system.

It is essential that the constructed model of the system to be built correctly represent the reality under consideration and the user requirements. To define what the term “correct” stands for in this respect is however a difficult task. Often the systems analyst have to compare and choose among requirements that are all correct in some sense, and the choices may lead to different information systems [Gulla96]. Accordingly the importance of requirements engineering , i.e. the process where the properties of the desired information system are discussed and recorded, have since long been recognised as a crucial part of systems development [Lubars92]. The conceptual models used during requirements engineering will and must always be a simplification of the reality they are supposed to represent. This proposition does not only imply that the number of concepts in a model must always be a subset of the concepts present in reality, but also that the choice of concepts present in the model must be the *right* one. What the user of a constructed system wants is a system with high *effectiveness*, i.e.

a system that *does the right thing* (-s) as opposed to a system that possess high *efficiency* only, i.e. a system that only *does things in the right way*. The difference is fundamental, where the significance of effectiveness is by far the most important of the two. Models of what functionality a system shall possess are typically constructed during the *early* stages in systems development. This state of affairs is by no means a coincidence. All deviations in the model from what is "correct" with respect to the reality to be represented and/or the user requirements will cascade to later stages in the development of the system. To correct an already implemented malfunctioning or insufficient system is by far more difficult and costly than to improve the system in an early modelling stage. The effectiveness of the system is therefore mainly dependent of the outcome of the systems analysis phase, the conceptual model, while the efficiency of the system is mainly dealt with in the later stages of systems development, i.e. in the design and implementation stages.

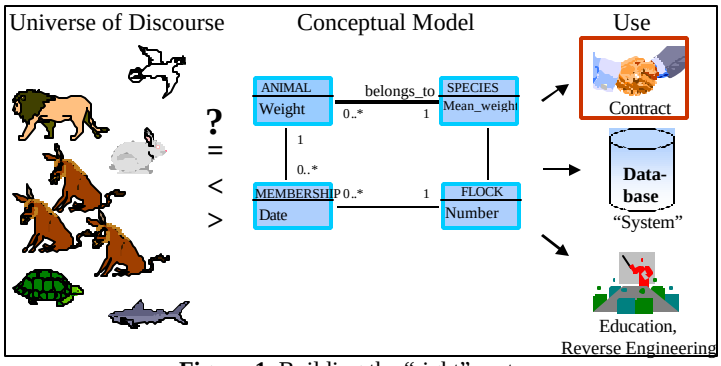


Figure 1. Building the "right" system

In order to achieve the desired correctness of the conceptual model it is an important issue to bridge the gap between formal language, in this respect different modelling notations, and the languages used by people who understand the domain. To map concepts in conceptual models onto natural language concepts is, however, a difficult task. One fundamental problem is that a Universe of Discourse can be modelled in many different ways. The same phenomenon may be seen from different levels of abstraction or be represented using different properties. Different terms can denote the same concept, and different modelling structures can represent the same reality. In order to do so it is necessary to investigate and identify what concepts in the models are appropriate to explain and in what context explanations should be formed.

In this paper we will address the topic of schema validation through explanation generation in the context of data abstractions. The paper is based on [Bergholtz00] and is organised follows. Section 2 introduces the modelling formalism and notation used in the rest of the paper. Section 3 provides the theoretical background for validation, focusing on generations of explanations in natural language. Section 4 outlines a number of general problems natural language generation (NLG) focusing on the problem of determining the context in which to generate natural language explanations. In section 5 we propose to use analysis patterns and other data abstractions as a context for explanation generation. Section 6 concludes the paper and gives directions for further research and relevance for Lyee.

2 CONCEPTUAL MODELS AND CONCEPTUAL MODELLING LANGUAGES

According to [Boman et al. 97] a conceptual model consists of a conceptual schema and a corresponding information base i. e. the instances corresponding to the types in the conceptual schema. The conceptual schema, in turn, can be viewed as a language (i. e. enumerations of entitytypes, attributes, relations) to describe the phenomena in the system to be modelled , a set of derivation rules and integrity constraints and a set of event-rules describing the behaviour of the object system. The most common graphical modeling notation used in conceptual modelling is the Entity-Relationship diagram introduced by Peter Chen [Chen76].

For the purpose of explaining conceptual models the Unified Modelling Language (UML) was chosen as the modelling notation. We will briefly describe some of the features of UML (class diagrams and object diagrams) together with instances of the same features which will be used as examples throughout the rest of this paper.

UML [UML97] is a visual modelling language designed by Grady Booch, Ivar Jacobsen, and James Rumbaugh to standardize already existing modelling languages for the various methodologies used in object-oriented systems analysis and design. For the purpose of this paper we will employ only parts of UML especially the class-diagram, object diagram and rule-language OCL

2.1 UML Class Diagram

A UML class diagram describes the types of objects in the system and the various kinds of static relationships that exists among them.

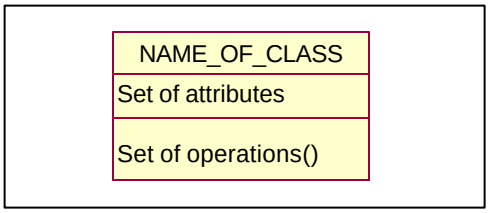


Figure 2. A UML class

A UML class is graphically denoted by a rectangle divided into three parts: the name of the class (reflecting the name of the real-world phenomena-type to be modelled), an enumeration of attributes describing essential properties of (the objects in) the class and an enumeration of operations describing the behaviour of the (objects instantiating the) class.

Relations between objects are denoted graphically different depending of the type of the relationship. We will discuss the graphical notation for one specific relationship type only. An example of an *association-relationship* is given in Fig. 1 where the class ANIMAL is related to the class SPECIES by means of an association (graphically denoted by a line between the to classes). An association has two *roles*, each role is the direction on the association. The roles may be given a label reflecting the semantics of the (direction of the) association. In Fig. 1 only the role from ANIMAL to SPECIES has received a label "belongs_to", the role from SPECIES to ANIMAL is left unlabeled. Each role is subject to a cardinality constraint, depicted in UML by minimum..maxiumum sybols. The most common multiplicities are '1' (exactly one), 0..1 (zero to one) and '*' (this symbol in fact doesn't inflict any cardinality constraints on the role; every multiplicity is allowed).

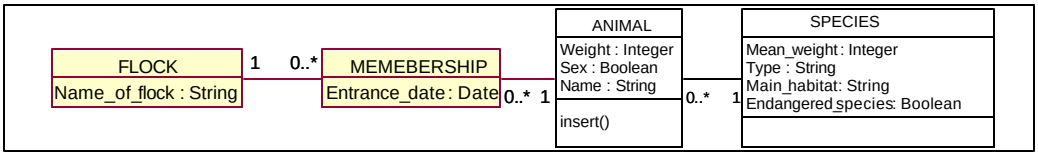


Figure 3. A UML Class Diagram

2.2 Specifying constraints and the Object Constraint Language

In addition to the structural aspects of the Universe of Discourse to be modelled the graphical notation of an UML class diagram include several static *constraints*. A constraint is a restriction on one or more values of part of an object oriented model or system [WarmerKleppe99]. In the class diagram of Fig1 several constraints are graphically represented, for example the multiplicity symbol of each role in an association constitute a constraint on the number of objects that can be related. [Meyer85] uses the term *assertion* for grouping together three types of constraints: *preconditions*, *postconditions* and *invariants*. An invariant stipulate certain conditions which must always be met by every instance of type, class or interface. The principles of pre- and post conditions is often referred to as the *design by contract* principle [Meyer91]. A contract is a specification of the interface of an object, that is the operations a certain object can perform. For each operation, the rights of the object that offers the contract are defined by means of preconditions. A precondition specifies a number of conditions that must be true in order for a certain operation to execute. Postconditions state the obligations of the object offering the contract, i.e. the postconditions must hold when the operation has just ended its execution.

Within UML the Object Constraint Language (OCL) is the standard for defining invariants, pre- and post conditions as well as other kinds of constraints.
A valid *OCL expression* has a type, a result and a context. The result is the value of evaluating the expression and the type is given by the type of the result. Types in OCL can be devided into predefined basic types (Integer,

Real, String, Boolean and different collection types) and user-defined model types defined in the UML class diagrams (se Figure 4). Every class, interface and type in any kind of UML model is a type in OCL. OCL discriminates between value types and object types where value types define instances that never change their value (for instance the integer 1) while object types define instances that can change their values. In Fig1 an instance of the model type ANIMAL may well change the value of the individual properties belonging to the type. The context, finally, of an OCL expression is always an element of a UML model (exempel behövs).

An *OCL constraint* is a valid OCL expression of type Boolean. The context of the OCL constraint depends of the kind of the constraint, invariant, pre- or post condition. The context of an invariant is always a class, interface or type. Defining an OCL invariant means declaring the contextual type followed by an enumeration of valid OCL expressions of type Boolean. In Fig2 an example of a constraint used as an invariant is given. The contextual type is SPECIES and the invariant states that for all legal instances of SPECIES the length of the value of the attribute 'main_habitat' must be greater than zero (empty strings are thus not allowed). The invariant makes use of the operation 'length' defined in the predefined basic type 'String' which is the data type of the attribute 'main_habitat' of class SPECIES.

```
SPECIES

main_habitat.length > 0
```

Figure 4. An OCL-invariant

The context of OCL pre- and postconditions is always an operation declared in the contextual type. The definition consists of the contextual type followed by the name of the operation, potential parameters and their type, the return type value of the operation followed by the pre and post condition expressions which both constitute valid OCL expressions of type Boolean. The syntax of a declaration of pre- and post conditions as well as an example declaration of pre- and post conditions for operation 'name()' of contextual type ANIMAL can be found in Figure 5.

```
Context Typename::operationName(parameter1: Type1, ...): ReturnType
Pre: parameter1 = true XOR ...
Post: result = ...

Context ANIMAL::name(): String
Pre: name.String.length <> 0
Post: result = name
```

Figure 5. OCL pre- and post conditions

The precondition stipulate that every legal instance of class ANIMAL must have a name (note that this precondition in fact is redundant since the cardinality constraint depicted in the visual model already demand that every animal should have a name). The reserved word 'result' in the post condition hold the returned value of the operation.

2.3 UML Object diagrams

A UML class diagram can have instantiations, depicted in an *object diagram*, describing a number of classes and their properties and relations at a particular point in time. The visual notation for object diagram resemble the notation of the class diagram the difference being that all instances in a relationship are shown with the names of each object underlined. An example of object diagrams, which will be used in later sections is given below.

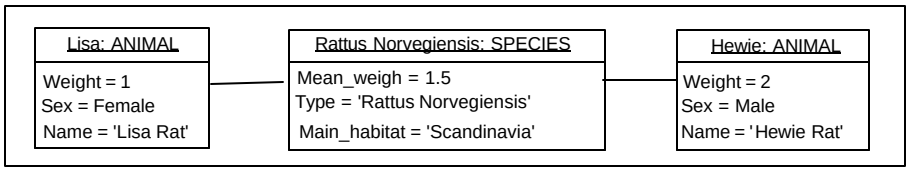


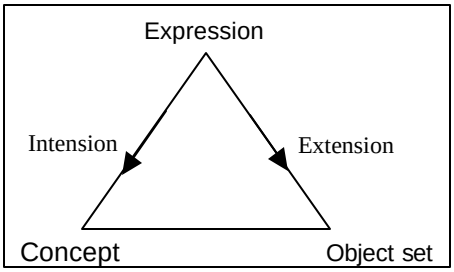
Figure 6. Part of a UML Object Diagram corresponding to the class diagram of **Figure 1**.

3 THEORETICAL BACKGROUND ON VALIDATION

The process of ensuring that a model properly represents the reality under consideration and the requirements of the user is called *validation*. While verification deal with the formal, syntactic properties of the model, validation is mainly targeting the issue of *semantics*, i.e. the relationship between the modelling constructs and reality. Returning to the issues of effectiveness and efficiency, validation deals with the degree of effectiveness, i.e. an assessment that the constructed model is the right one, rather than an assessment of how well some focal point (-s) in an Universe of Discourse is translated into a model.

Common to many approaches in semantics is the idea that the meaning of construct is completely determined by the meanings of its constituents. This idea is commonly called the principle of compositionally [Boman et al. 97]. The principle states that the meaning of a complex construct can be obtained by some operation on the meaning of its parts. The main drawback of this principle is that it does not take into account the context in which a construct appears. The meaning of a composite construct is hence determined both by its constituents and its context. Since models are generally not small and in-complex, the sheer size and complexity make the principle of compositionally inadequate in a validation process.

Another approach to understanding the meaning of individual terms and expressions is to distinguish between the extension and intension of the terms. The *extension* of a term means the object or set of objects in the real world to which the expression refers. The extension of the term 'rat' is the set of all rats, and the extension of the expression 'Heaviest rat in the system' is 'Lisa' if 'Lisa is indeed the rat with the highest weight in the data base. Different terms or expressions may have the same extension, for instance the planet Venus is referred to both with the term 'Evening Star' and 'Morning Star'. The *intension* of a term is its sense; that which a person usually understands by the term. The intension of a rat might be "medium sized, hairy, grey or brown, mammal with four legs and a long tail". The relationship between linguistic terms or expressions and their extensions and intensions can be depicted graphically in a figure called Ogden's triangle.



Figur 7. Ogden's triangle

The validation process shall detect flaws and give suggestions for corrections. Validation is a joint venture between the people who know the domain in question and the analysts who know the modelling language. It is often an informal process where the different stakeholders participate. This will include people with only a limited amount of knowledge of modelling and systems design. If a person does not understand the formal language of the model it is hard to make assumptions as to weather or not the model contains what is essential for the person in question, i.e. it will be hard for him or her to validate the model.

While verification can often be automated, the validation task is not fully formalizable and requires subjective human judgement. It is desirable to provide the validator with the maximum set of tools that assist him in the validation process [Bubenko86]. A number of techniques have been established to ease the process of validating a model, i.e. to bridge the gap between the language employed in the model and the language used by the domain expert. The term domain expert here is used to refer to a wide range of people, e.g. the stakeholders who will be involved in the future utilisation of the system-to-be-designed.

One approach to validate a model is to *simulate* it, i.e. to build an interface to facilitate the observation and experimentation with the dynamic properties of the model [Harel87][Zave84]. A validation technique similar to simulation is *model planning* where users can explore a model by constructing plans, i.e. operation sequences that result in states where certain conditions are met [Costal96]. In order to validate large conceptual models a technique called *complexity reduction* is used. This technique presents different views of a conceptual model and hides details irrelevant for the particular view. Since most people are not trained to understand formal descriptions, but every one understands at least one natural language, *paraphrasing (parts of) the conceptual model into natural language* is yet another technique to ease the understanding of the model [Rolland92] [Dalianis96]. An integrated approach of the techniques mentioned above is *explanation generation*. Generating explanations can be looked upon as an extension of paraphrasing. An explanation-generation interface will allow the user to explore the model interactively, posing questions about model and receiving appropriate answers. Explanation generation facilities can thus be used for other validation techniques such as simulation and planning. Explanation generation in general and as a validation technique will be discussed in the next two sections.

3.1 Automatic natural language generation

Informally, natural language generation (NLG) as well as the problems connected with NLG can be condensed into the two questions:

- What to say?
- How to say it?

Intuitively these two questions correspond to the two main phases identified in NLG: *deep generation* and *surface generation*. Both concepts originally stem from the way humans go about in the process of generating text. In the *automatic* generation of natural language we create computer programs that mimic these human text generation patterns. When a human being wants to communicate with the outside world, in written text or orally, the overall *purpose* of the discourse (the coherent text) is what must be decided on first. *What* the text is about. Having made her decision the person then formulates the discourse, i.e. chooses appropriate words and puts the words together to form syntactically well formed sentences. *How* to implement the purpose as text. Accordingly deep generation deals with the selection of information from a, potentially large, knowledge pool and the planning of the organisation of this information into a coherent text. Surface generation, on the other hand, realises the output from the deep generator via grammatical rules and different lexica, domain dependent as well as domain independent.

It is possible to categorise text generation systems with respect to sophistication and expressive power. The simplest approach is *canned text systems*, where the system simply generates a string of more or less informative words. Trivial to create, this kind of text generation is stereotype and often next to useless to the intended user. Created by the programmer at the design time of a system, these texts do generally not respond to the needs of the user of the system at run time. Examples of canned texts that carry virtually no information at all is "Internal errno 3421" or "The program has performed a forbidden action and will be terminated immediately". More sophisticated systems use *templates* or predefined frames to be instantiated at run-time to better match the needs of the user. TEXT [McKeown88] and TAILOR [Paris88] are examples of template systems where the instantiated templates, or schemas, are further nested into coherent paragraphs. *Phrase-based systems* employ what can be seen as generalised templates used at either sentence-level or discourse level. An example system in this category is Rhetorical Structure Theory [Mann88]. RST defines a number of so called rhetorical relations, used to categorise and relate different parts of a text, i.e. sentences or paragraphs, to create a coherent discourse. *Feature based systems* represent to some extent, the limit point of phrase-based systems. In feature based systems, each possible minimal alternative of linguistic expression is represented by a single feature, making it possible to create very fine grained output tailored to meet the needs of different users. The weakness in this approach lie in the anticipated large number of features and the problems of determining and maintaining interrelationships between features. Language generators using feature based systems have therefore mainly been used for single-sentence generation, where one of the most commonly used systems is the Functional Unification Grammar Framework (FUF) [Eldhadad92].

An observation regarding the functional categorisation of language generators discussed above is that the manner in which they choose to represent the knowledge needed to generate natural language, is by no means independent of the intended *user* of a text generation system. To be able to adjust the text to different kinds of users, e.g. novices as well as experts, is an essential feature in any text generation system. The model of the user for whom the text is generated must be incorporated in the overall knowledge representation employed by the

generation system. Returning to the essence of text generation as stated above, the two questions “What” and “How” should therefore be complemented with a third one:

- Who is the responder of the text?

To define what a good text, or a good explanation of a concept, with respect to a responder is an essential but complex task. The goal is to be able to extract an appropriate amount of information and organise it in a manner that best suites the user. The explanation must be informative and yet comprehensible enough, i.e. the text must be on the right level. What is indeed the right level depends on what the user already know, the domain and the complexity of the concept to be explained. Moreover, the system must be able to respond to the user and change the level of explanations to match these criteria.

Several approaches exist in detecting what is the level of the user. One way is to analyse the behaviour of the user. In [Appelt88] a categorisation of the questions posed by the user is made in order to detect what the user does know as well as does not know. Another method is to store and analyse *sequences* of user dialogs. If a certain pattern occurs several times, in the simplest case if a user ask a question twice, this may be an indication that the a given explanation was not on appropriate level. A third approach is to use predefined *user models* describing level characteristic criteria. TAILOR [Paris93] is an example where these strategies are combined dynamically at runtime. Several systems, e.g. expert systems are originally designed not only to calculate answers on various problems but also to justify these answers to the user of the systems. KAMP, Knowledge And Modalities Planner, use modal logic to reason about the users knowledge and beliefs in order to plan explanations in natural language [Appelt85].

3.2 Mapping concepts of conceptual models onto natural language constructs.

Planning and structuring explanations of conceptual model constructs correspond to the content determination problems of deep generation discussed in the previous section. Explaining conceptual models means defining strategies to determine what parts of the model are appropriate to explain as well as determining on what level the explanations should be formed with respect to novice- and expert users, and, from a generation system point of view, the means of interaction between user and system.

3.2.1 Relationship between natural language and conceptual models

In addition to these general NLG-problems, a number of issues that arise from the difficulties in explaining conceptual models in particular, must be addressed. The links between conceptual models and natural language in general have been explored by a number of authors including the founder of the ER-modelling technique Peter Chen. [Chen83] advocates 11 rules that visualise the correspondence between English sentence structure and ER-diagrams. As a general practise, given a narrative description of the system requirements to be represented by a conceptual model, the *nouns* appearing in the narrative give rise to entity type names and *verbs* tend to indicate names of relationship types. *Attributes* names generally arise from additional nouns that play the role of descriptors of other nouns in the text. Accordingly *adverbs* maps onto attributes of relationships.

The original mapping between ER-model and natural language as proposed by Chen was however rather coarse, and did by no means capture all of the semantics present in a natural language requirements specification. The reason for this is not restricted to NLG application mainly but rather to the fact that the original ER-model could not represent more complex features, needed to catch more of the semantics of the UoD. These features were instead present in the various semantic data models present in literature at the time of the introduction of the ER-model, for a survey please refer to [Peckham88]. To resolve this state of affairs many extensions of the ER-model have been proposed to include the data abstractions of semantic data modeling. [Elmasri92], [Motschnig-Pitrik&Mylopopoulos92], [Brachman93] have all suggested extensions of the ER-model with data abstractions that have immediate correspondences in natural language sentences (part-of, member-of, role-of, isa). Data abstractions such as ‘isa’-relationships, for instance, have their counter parts in English sentence constructions such as “… a dachshound is a kind of a dog”. It is possible to find mappings between all data abstractions used in conceptual modelling and natural language constructs, for an example see for instance [Goldstein&Storey99], [Elmasri92] or [Lewerenz99]. One problem is that these mapping are heavily dependent on domain and/or context. [Burg95], [Hakkarainen99] proposes extensions of the ER-model by means of domain independent linguistic theories for the purpose of increasing understandability and, in the case of [Burg95], ease of explanation generation.

3.2.2 Similarities and discrepancies in conceptual models

Point of departure in the interpretation and validation of conceptual models is the observation that one and the same real world concept may give rise to many different modelling constructs and vice versa. Typically, the

discrepancies can originate from different naming practises, modelling practices, differences in focus or abstraction level or level of detail [Batini86], [Johannesson 93], [Hakkarainen99].

- The discrepancies can have their origin in *different naming practise*, which in conceptual schemas may appear as terminological conflicts, such as synonyms, homonyms and scale differences.
- Discrepancies may also be caused by *differences in focus*, which may result as behavioural conflicts, such as different events, dynamic rules or integrity conflicts
- The reason for discrepancies can lie in the use of *different abstraction levels or modelling practises*, which will give rise to structural conflicts, such as different types or level of detail and abstraction.

Following the above sources of different representations in conceptual schemas of one and the same real-world concept, we discuss three types in more detail:

- *Terminological discrepancies* arise when people from different organisations refer to the same thing using their own terminology. Terminological discrepancies are classified as:
  - *Synonyms* occur when the same object or relationship in the UoD is represented by different names
  - *Honomyms* occur when different objects or relationships are represented by the same name, e.g. the term 'article' could refer to a product or a piece of related text in a newspaper.
  - *Scale differences* occurs when the value of the same property in the UoD is expressed using different scale factors or different scale of measurement.
- *Behavioural discrepancies* arise when the same phenomena in the UoD involve different events or integrity constraints. Cardinality constraints is the most frequent example. A system that records the different owners of real estate, one piece of real estate may be allowed to point to one juridical owner only. This may not be the case in real life, however, where two or more persons may co-own a piece of land.
- The same aspect of the real world may also be modelled using *structurally* different constructs. A structural discrepancy occurs when the same concept have been modelled using different schema constructs. A common example is the relationship construct, which can either be modelled directly as an association, or indirectly by introducing an extra entity that ties the associated classes together. The reversed situation is also possible, i.e. the same type of structural component may have different meaning in different modelling situations. The most significant example is the overloading of the relationship construct, which can denote many types of data abstractions, as will be discussed in section 4.3.

Due to the difficulty in finding an exhaustive classification of semantically equivalent but structurally different modelling constructs, structural synonymy poses one of the most difficult problems in finding general strategies to interpret conceptual models and create natural language explanations of the same.

4 Contexts for explanation generation

To interpret conceptual models, and in this respect to create explanations of the same, it is necessary to resolve the ambiguities that arise due to the occurrence of synonyms, homonyms, and structural differences. To determine whether two terms are synonyms several strategies from the field of linguistics may be applied. Concept interpretation include detecting other relationships between concepts such as subset-relationships and part-whole relationships. One way is to fuse concepts [Lin&Hovy97] together into more general unifying concepts. Concept fusion can be done as a part-whole construction. Terms like 'Wheel', 'chain', 'pedal', 'saddle', 'light' etc. may be fused into the more general concept of 'bicycle'. Another way of fusing concepts together are by means of a concept taxonomy. "John buys apples, oranges and bananas" may be transformed to "John buys fruit". This kind of generalization is also known as lexical aggregation. Syntactic aggregation (Dalianis, 1996) performs so called coordinations to make the text shorter and in certain cases less redundant (Dalianis, 1999). "Mary walks and John walks" become "Mary and John walk". The generalizations described above are sometimes difficult to carry out because taxonomies like WordNet [Miller90] lack domain specific knowledge. In a restaurant domain, a set of words like 'customer', 'food', 'menu', 'waiter', 'chéf' etc. are indeed related and should be automatically fused into the general concept of 'restaurant'. Terms may be related in certain contexts and domains and not in others. Information retrieval techniques like concept fusion are therefore

not adequate for detecting similarities and degree of relatedness between terms in a domain specific conceptual model.

4.1. Determining proper explanation generation contexts

When a term is used in a conceptual model, synonymy may also be resolved by comparing the context of a term. When the term refers to an entity (class) the context may be defined [Johannesson93] as the set of attributes, the entities to which it is related via associations, and its subtypes and super types. The context of an attribute may be defined as its domain and range, i.e. the entity type where the attribute is defined together with the data type of the attribute. Several other approached exist, i.e. to take into account the constraints at hand or dynamic properties. This line of research is pursued in [Dalianis97] claims that in explaining a formal model three parts have to be considered; first, the static description of the model (how the parts relate to each other), second, the dynamic part; (how the model behaves); third, the instantiation of the model (the actual references between model and reality).

Generally explaining to small parts of the model may not be informative enough. Simply paraphrasing, for instance an association between two classes, may not contribute much to the users understanding of the significance of the association, than actually inspecting a graphical schema fragment of the same phenomena. Explanations of too large schema fragments may also prove counter productive, since the user may find the constructed texts too general or unfocused [Gulla96]. [Dalianis97] argue that short texts or visualisations that answer focused questions about structure and behaviour of the model is more effective. The question about how much of the structure or behaviour should be included in an explanation remains still, however, an open issue.

Our approach is to use a number of data abstractions as a scope for explaining any term used in a conceptual schema. By the term data abstraction we mean semantic relationships, such as generalisation/specialisation (is-a) but also larger parts of conceptual schemas such as analysis patterns, i.e. small conceptual views that solve a certain modelling problem. In [Berholtz00] we introduce this notion in a framework for combining three different sources of contexts for explaining the constituents of conceptual models. The first source builds on the work on [Dalianis92] and [Dalianis97], where generations is formed with respect to the different levels of models per se; meta level explanations (explanations about the formal language in which the model is expressed); schema level explanations (explanations of named concept types in the model); and instance level explanations (explanations about real world individuals). The second source of contexts for explanations of CM concepts are analysis patterns, i.e. small conceptual views that solve a generic modelling problem. The third source for structuring CM explanations is the data abstractions [Goldstein&Storey99] used in conceptual models. The rules for when and where a certain pattern or other model construct should be applied will contribute to structure the explanations of, or motivations for, why a schema contain certain combinations of elements as well as contribute to the users understanding of the constraints pertaining to different patterns and the dependencies that exist between patterns and other model constructs.

4.2 Analysis patterns as a means for determining the explanation generation context

In terms of structuring explanations of conceptual models, the concept of *analysis patterns* may serve as a natural context. We believe that it provides an additional and important abstraction level for effectively explaining the semantics of different modelling constructs as well as naturally limiting the scope of the explanations to include proper parts of the model to be explained.

Analysis patterns is a special case of the more general pattern concept which has its roots in many disciplines, including literate programming, and most notably in Alexander's work on urban planning and building architecture [Alexander77] [Alexander79]. A pattern according to Alexander is some representation of information solving a generic problem. In the area of systems development, *design patterns* [Gamma95] address the design stage while *analysis patterns* concerns the analysis and specification stage. An analysis pattern describes a set of real-word objects, their interrelationships, and the rules that govern their behaviour and state. Examples of analysis patterns are the patterns in [Fowler97], the data model patterns of [Hay96] and the domain abstraction discussed in [Maiden92]. These patterns may be viewed as conceptual patterns, to be used, and reused, as an already constructed solution to a generic modelling problem. In our work we view these patterns from the opposite direction, as an instrument for generating explanations of conceptual models. The use of patterns in this respect is not very well investigated and we haven't been able to find any direct references to this use of analysis patterns. [Maiden98] uses design patterns in a vaguely related way, as a means of validating system requirements.

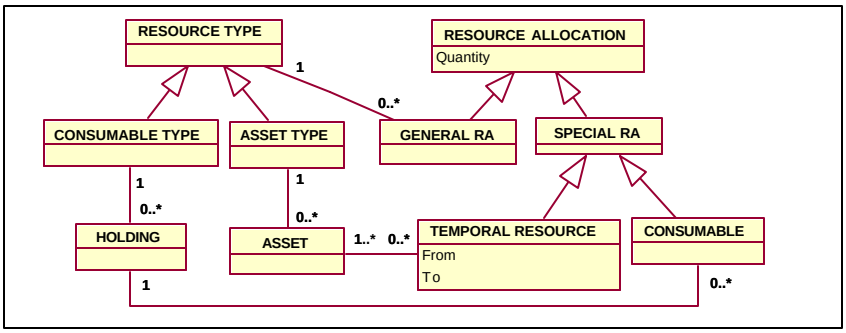


Figure 8. Resource allocation analysis pattern [Fowler97] (modified)

In Figure 3 an example analysis pattern, *resource allocation* [Fowler97], is given. The resource allocation pattern models different types of allocation of resources. Some resources are consumed in an activity, e. g. in surgery blood plasma is consumed. Other resources, assets, can be reused, e. g. a nurse. The consumable type classifies the consumable resources while individual assets are categorised by the asset type. A temporal resource is a specific resource allocation of an asset, whereas a consumable is a resource allocation of a consumable type from a certain holding (finite store). General resource allocation is used to represent what resource types are required for a certain activity. This feature is shown in the class diagram of Figure 9 where the resource allocation pattern is used to model the allocation of resources for patient activities in a health-care system. Figure 10 shows part of a corresponding object diagram.

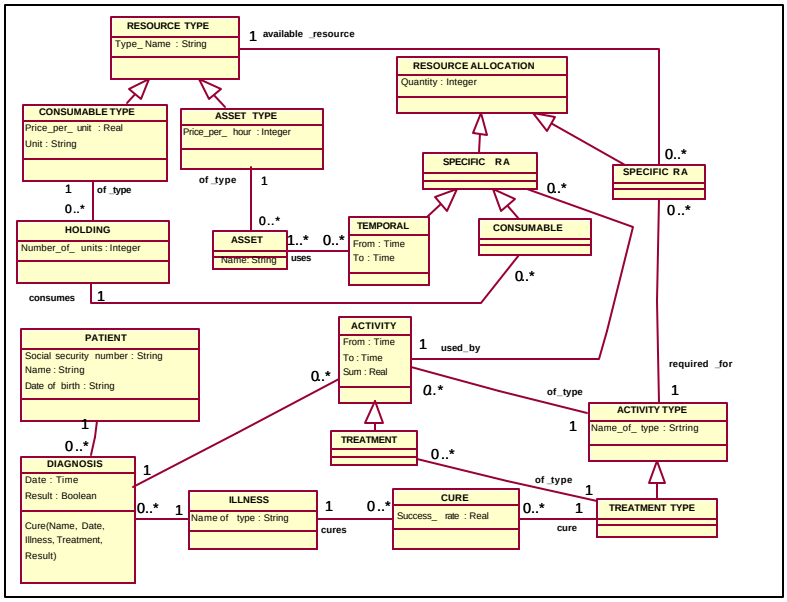


Figure 9. A class diagram utilising the resource allocation analysis pattern

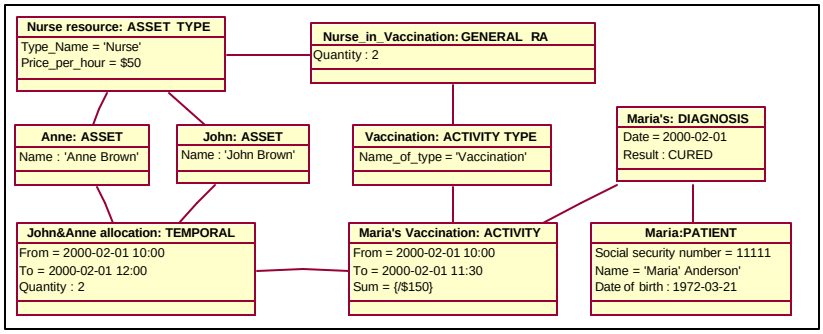


Figure 10. An object diagram corresponding to the class diagram of Figure 4

4.3 Additional data abstractions for explanation generation

The motivation for introducing new abstraction levels into conceptual model explanations is to provide a natural scope for structuring the explanations. Exactly what abstractions to use is not obvious. In addition to analysis patterns we will also utilise a number of data abstraction relationships commonly used in conceptual data modelling. Explaining data abstractions is motivated by a number of reasons. They are present in conceptual schemas in their own right as very basic domain independent building blocks, capturing the *meaning* as well as the structure of data. Equally important, they form the basis (constitute the building blocks) of more complex conceptual patterns such as analysis patterns. Thirdly we observe that even if some graphical constructs of conceptual modelling languages are indeed self explanatory, some of the semantic modelling concepts are not. Examples can be found in the area of cardinality constraints and the constraints pertaining to different types of data abstractions. Finally there exist several interpretations of even the most common data abstractions employed in conceptual data models such as the Extended Entity Relationship Model (EER) [Elmasri92]. From an explanation generation point of view the situation is further complicated by the fact that there does not exist a common agreed upon graphical notation for denoting neither the different data abstractions nor the variants of one and the same abstraction. This is manifested in the diagramming techniques of various systems development tools [RationalRose], [Access], [ArgoUML], which do currently not support all data abstractions. As a result, the graphical symbols present in the diagramming tools are overloaded in order to represent many different kinds of data abstractions. To make a novice modeller understand the significance of , for instance an association, an explanation generation system must make him or her aware of the possible interpretations in terms of semantic data abstractions.

INCLUSION

An inclusion abstraction, often denoted by is-a, represent a supertype/subtype relationship [Brachman93]. In a relationship, A is-a B, A is referred to as the specific type and B, the generic entity type. Several kinds of inclusion abstractions may be distinguished [Goldstein&Storey99]:

- *Classification* is an inclusion abstraction between an entity occurence and its corresponding entity type [25 Storey]; for example: 'Donald is-a Duck'. The reverse relationship from an entity type to its occurrence is called *instantiation* [Motschnig-Pitrik&Mylopopoulos92]. This data abstraction is generally supported by most ER languages. The correspondence in UML is the object diagrams.
- Generalisation/Specialisation: Specialisation is the process of classifying a number of occurrences of a generic class into more specialised sub-classes. Specialisation is based on a number of distinguishing features possessed by some occurrences only. Generalisation is the inverse process of generalising several classes into a higher level abstract class, the super class, that includes all the occurrences in all the classes.
  - If the super type is defined as the union of non-overlapping specific entity types , it is called a partition [Goldstein&Storey99]. For example, *Student* is a generalisation of *Undergraduate Student* and *Graduate Student*.
  - When overlapping against the specific entity type can occur, a subset hierarchy is formed . As an example consider the super class *Employee* with sup classes *Part Time Employee*, *Full Time Employee*, *Secretary* and *Technician*.

Extended ER-models, such as EER and the class diagrams of UML, provide support for the generalization/specialisation abstraction relationship. The distinction between partitions and subset hierarchies is however generally not present.

POWER TYPES

Another type of abstraction relationship applies between the operational level and the knowledge level, [Fowler97], [Geerts00]. The operational level models actual, concrete individuals in a domain, e.g. a concrete car. The knowledge level models information structures that characterise categories of individuals at the operational level; an example of such an information structure is a car model. [Martin/Odell95] employ the concept of *power types* to refer to the correspondence between the objects of the knowledge- and operational levels. A power type is an object type whose instances are sub types of another object type. A concrete car, e.g. a blue, three years old SAAB, is related to the abstract SAAB-model (the power type) by means of a power type relationship. The power type relationship is not present in the EER-model and EER-modelling languages, for example UML. Diagramming tools such as Rational Rose, Access and ArgoUML do not provide graphical

symbols power types. The use of stereotypes in UML can be used to indicate which entity type (class) that play the role of power type in a relationship.

RELATIONAL CLASSES

An important concept is objectification [Hofstede97] (introduction of a relational class/association class) of relationships that occur time and again in different analysis patterns and even form the basis of some of these patterns, examples are the accountability pattern with it's numerous variations [Fowler97] and the asset structure element models of [Hay96]. We will give an example of a description in natural language of the relational class. In a user dialog, this text will serve as a motivation why a relational class has been introduced into the conceptual schema as well as giving the semantics of the relational class. The definition below should be additionally augmented by example instances from the schema.

> A relational class (or association class) must be introduced if there exists either a multivalued relation (in UML this corresponds to a relation where both roles have either multiplicity 0..* or 1..*) *where the relation has properties of its own* or a relation between more than two classes. The properties of the relation become attributes of the relational class.

4.4 Argumentation model

The constituents of a conceptual model will be explained in the context of the particular data abstraction to which it belongs. In addition, an argumentation model is needed to structure the dialog between a user exploring a conceptual model and the system generating answers in natural language. Ideally the fundaments of the argumentation model should correspond closely to the components of the conceptual model in the sense that the argumentation model is independent of domain of application. Another important quality of an argumentation model is that is should be able to structure the explanations so that the appropriate amount of information is given on the right level.

A natural candidate is the Rhetorical Structure Theory (RST) [Mann87], which is a model for describing the structure of a coherent text. RST-*schemas* specify how a text could be broken down into smaller parts. Each schema defines a number of *rhetorical relations* where each relation associates a *nucleus*, a central concept in a text, to a *satellite*, another concept in a text which support the satellite. The drawback of using RST for structuring explanations in natural language of a conceptual model is that the rhetorical relations have no immediate correspondence to the constructs of the model to be explained, which will require heavy customisation for each domain.

An argumentation model that is domain independent and where the constituents are easily mapped onto the specification of a conceptual model is the Toulmin argumentation model [Toulmin59]. [Dalianis97] provide an overview of the Toulmin model. The starting point of an argument is a *claim* which is a sentence asserting some proposition. The claim is related to *grounds* supporting the claim. If the presumed listener to the argument is not convinced that the claim holds on basis of the grounds the argument may continue with a *warrant*, i.e. a relation between a ground and a claim showing that the grounds are indeed relevant for the claim. A warrant has usually the form of a general rule that is applicable to the case at hand. If grounds and warrant should not provide enough evidence for the listener, a supportive argument can be given in form of a *backing*. Normally backing takes the form of rules at a higher level than the warrant. If the propositioned claim still not follows with certainty from the grounds, warrant and backing certain *qualifications* such as "usually" and "possibly" may then be used to clarity the relationship between grounds and claim. A *rebuttal* in Toumin's argumentation model stands for an argument that describes the circumstances under which a claim does *not* hold.

The constituents of the Toulmin argumentation model can be mapped onto a specification of a conceptual model in a way which makes it possible to structure the explanations by gathering information from different levels and components of the conceptual model. If a user questions the grounds for a specific claim, the warrant may include additional information from the conceptual model and form the basis for a more detailed explanation. User-queries involving phenomena in the conceptual model may be given at the instance- schema or meta-schema levels. On the schema-level we distinguish between a queries pertaining to an isolated schema-fragment and queries made in the larger context of an analysis pattern. If the query is given at the schema-level (for instance "What is an *animal*?") it is appropriate to structure the system answers so that the grounds are given at the schema-level, the warrant at a more detailed schema-level involving a larger part of the schema corresponding the analysis pattern to which the schema fragment belongs and the backing, finally, is given at the meta-schema level. An argument and it's mapping onto the different levels of the conceptual model can be described as shown in Fig. 11.

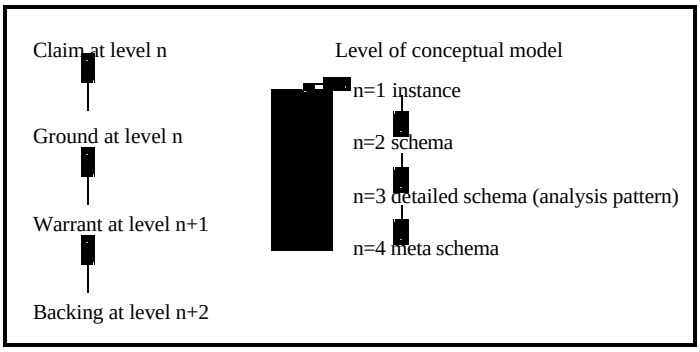


Figure 11. Toulmins argumentation model mapped onto different levels of a conceptual model

The basic idea of Figure 6 is that when a claim is given on a certain conceptual level the system answer by giving an explanation on the same level or the level immediately following the claim-level (corresponding to the grounds). If the user request more information, the next level of the conceptual model will be utilised in the warrent. The mapping between conceptual levels and constituents of Toulmin's argumentation model is complete only in the case when the query is given at the schema level and we believe that explanations designed according to Toulmin's model are probably most useful for claims on the instance or schema-level. For explanations of analysis patterns it is possible to identify at least the following contexts:

- An explanation of an individual construct warranted by a larger explanation of the analysis pattern in the context of the queried schema fragment (in which situation the claim is on instance or schema-level and the grounds in the instance or schema-level)

- An explanation of the (entire) analysis pattern present in the conceptual schema to be explained where the grounds are given at the meta-level describing the semantics pertaining to the general domain independent analysis pattern (in which case the claim is on the pattern-level)

- An explanation of an (entire) general analysis pattern warranted by exemplifications in the domain depicted in the conceptual schema to be explained (in which case the claim is on the meta-level asking for the semantics of an analysis pattern and the grounds are given at the meta-level using the semantics pertaining to the general possibly domain independent analysis pattern)

Queries at meta-level (i. e. queries regarding the modelling language) may very well be warranted or backed by exemplifications at the instance level as is indicated in Figure 11.

5 Application of the explanation generation architecture

In this section we will discuss and exemplify how explanation generation can be used for different validation techniques such as planning, simulation and explaining the class and object diagrams per se. Due to space limitations we will only exemplify simulation and structural explanations of object diagrams in detail.

5.1 Explanations of class and object diagrams per se

To explain object diagrams involve answering questions why specific elements in the conceptual model. Elements may be objects themselves, attributes or relations between objects or even larger parts of schema for instance an analysis pattern. Class diagrams may give rise to a large variety of queries and explanations on both schema and meta schema-level. The most obvious question is probably the structural one asking about the structure of a certain phenomena or why a certain fact holds.
Queries involve phenomena on the meta-schema-, schema- or instance level whereas the answers, (the explanations), are structured from instance-level and upward in the instance-schema-metaschema hierarchy described in the previous section. Below is given an example of a user-query where the system-explanations are structured according to Toulmin's argumentation model. Every user-system interaction will utilise an analysis

pattern or other kind of data-abstraction as generalisation hierarchy, ralational (or association) classes or power-type (or all four together) to provide a semantic context in which to explain a queried schema-fragment.

- **Utilising the semantics pertaining to an UML association class**

USER: What is a MEMBERSHIP? (Schema-level structural query)
SYSTEM: A MEMBERSHIP is used to denote that a certain PERSON has entered a certain CLUB on one particular date. (grounds at the schema-level)
USER: I don't understand.
SYSTEM: A relational class (or association class) must be used if there exists either a multivalued relation *with properties of its own* or a relation between more than two classes. A PERSON may enter several CLUBs and a CLUB may have many members. To model that a certain PERSON became a member of a particular CLUB on a certain time an association class MEMBERSHIP is required to relate exactly one instance of PERSON to exactly one instance of CLUB with the property 'admission_date' set to the time of the PERSONs admission in the CLUB. (warrant at detailed schema-level)
USER: May a PERSON enter the same CLUB more than one time?
SYSTEM: Not in this particular case. The association class defines only one legal combination of each pair of classes related through the association class. This implies that there can only exist one MEMBERSHIP for every pair of PERSON-CLUB. (backing at the meta-schema level where one constraint belonging to the UML association class serves to motivate why the particular schema-fragment is used).

5.2 Explanations of simulation

Simulation means to execute dynamic rules – events – in a conceptual model and explore the consequences of these executions. The system exemplifies and justifies the result of the execution by referring to the relevant parts of the conceptual model. This can be done either by displaying the results of the execution of the rule; an insertion, deletion or update of an object (s) or object property (corresponding to giving the grounds in Tolmains argumentation model) or by verifying what preconditions must be met for the event (or method) to execute as well as what post conditions hold after the execution (corresponding to a warrant). If the question involve an event with several alternative execution paths the system must provide a motivation to why a certain branch is chosen over another.

A slightly more complex situation occurs when the user wants to execute a method in a context where the preconditions are not met. The natural answer from the system would be to display the preconditions and explain what corresponding facts in the schema violate the execution of the method (corresponding to the grounds of Toulmin's argumentation model). If the user request more input the system continue by giving the circumstances under which the method will execute (corresponding to a rebuttal in Toulmain's argumentation model). This amounts to not only defining what instances or values of properties of instances must be present for the precondition to hold but possibly also what sequence of methods must execute to cause the correct facts to be inserted into the object- or class diagram.

Returning to the UML-model of figure 9 we can simulate what happens if a new booking of a patient is introduced, e.g. insert of an instance of class BOOKING:

Utilising the semantics pertaining to power types

USER: What happens if a vaccination is booked for Maria on Sunday between 2.00 to 4.00 PM? (I. e. the method book(Date, From, To, Name, Illness, Treatment type) is executed)
SYSTEM: This cannot be done. A vaccination requires two nurses. On Sunday all available nurses are already booked during 2.00 to 4.00 PM. (Grounds on the instance level)
USER: Under which circumstances may a vaccination be booked for Maria?
SYSTEM: If the time of the vaccination is changed to either after 4.00 PM or before 1.00 PM the vaccination can be booked. Then a new instance of class ACTIVITY will be inserted. This activity will be related to the already existing instance of DIAGNOSIS that correspond to the input given by parameter 'Illness'.(Grounds at the instance level).
USER: Why?
SYSTEM: Different TREATMENT TYPE's require different numbers of different RESOURCE TYPE's. This is shown in the model by means of the RESOURCE ALLOCATION analysis pattern . The class RESOURCE ALLOCATION include two subclasses, one of which is the RESOURCE need class which is used as a dictionary to show what resources are *required* (as opposed to actually utilised or booked) for a certain kind of TREATMENT TYPE. TREATMENT TYPE can be seen as a power type of TREATMENT. By this we mean that the class TRTEATMENT TYPE can be seen as a template for the individual TREATMENTs belonging to

an TREATMENT TYPE class. In this case Maria's treatment is of treatment type vaccination. Since a vaccination type requires two nurses and there are nurses free only during the hours 1.00 through 4.00 PM on the desired day the input-parameters 'From' and 'To' must be in this interval. (warrant at pattern level).

The fact that a queried schema-fragment belongs to different analysis pattern constitutes a problem in structuring the answers. Including all the patterns to which a certain schema-construct belongs may easily create to long and unfocused explanations. Clearly rules for what pattern is best suited in an explanation needs to be established. In the former procedural question type we have vaguely indicated that the question type might indicate what objects in the schema are related to the focus of the question. This may be an indication of what analysis pattern to choose over another. Another approach to let the system show different suggestions for the user to choose from if there exist more than one applicable analysis pattern in which to explain a user query.

5.3 Explanations of planning

The explanation generation can also be used for planning. Planning is a more complex issue than simulation. Instead of mere execution of events and inspections of the results, planning involve defining goals and determining what events to execute in order to accomplish the goals. Continuing the resource allocation example, the goal of having a reasonable measure on the number of cured patients may be introduced by a user exploring the model. The objective of an explanation generation system is in this respect to show what implications this goal has on the behaviour of the model, i.e. what events must occur, what pre- and post conditions must hold.

6 Conclusions and further research

Validating conceptual models is a complex issue which spans a number of issues. Validation can not be totally automated but requires human judgement. Explanation generation can be used to combine several approaches for easing the validation process. Theories from the field of linguistics, in particular natural language generation, may be applied but their significance is limited due to the difficulty in finding an exhaustive classification of semantically equivalent but structurally different modelling constructs. Structural synonymy poses one of the most difficult problems in finding general strategies to interpret conceptual models and create natural language explanations of the same. Techniques from information retrieval such as concept fusion are also not adequate, mainly due to the lack of domain specific taxonomies. We have advocated the use of analysis patterns and data abstractions as a natural context for explaining such implicit dependencies between different constituents of the conceptual model, as well as for focusing the generation of explanations on relevant parts of the model. This approach is integrated with Toulmin's argumentation model in order to organise the dialog structure and the detail level of the explanations to meet different user requests. An advantage is that patterns and data abstractions are domain independent and hence applicable in the validation of any conceptual schema.

A number of issues have not been discussed in this paper. The rules for when and where a certain pattern or data abstraction should be applied have only been vaguely addressed. As can bee seen in the user dialogs of the previous sections, the fact that a queried model fragment may belong to several analysis patterns constitutes a problem in structuring the answers. Rules for what pattern is best suited in an explanation need to be established. One approach is to let the system show different suggestions for the user to choose from if there exists more than one applicable analysis pattern by which to explain a user query.

The issue of surface generation have also not been addressed. For the purpose of this paper we mainly envisage user interaction directly with a graphic view of the conceptual model by means of point and click interaction, in which case no parsing of user input is necessary. Follow up questions from the user could also be managed via a graphical user interface where the system displays the possible alternatives. This includes giving more detailed information as well as letting the user query part of an explanation, for instance by highlighting key-words used in the explanation.

Relevance for Lyee

In [Rolland01] a Requirements Meta Model over the relationship between users requirements (User Requirement level) and the Lyee Requirements level, is presented. The work reported upon in [Rolland01] can be viewed both as a road map to model the users requirements as well as transform them into a Lyee program. The Lyee methodology clearly incorporates support for validation of the users requirements. This is done in at least two ways. First, the methodology itself aid the user in the structuring of the users requirements in a stringent

manner. Secondly, the methodology provide support for the detection of conflicting goals with respect to user input. However, the designer applying the Lyee methodology are self responsible for the correct definition of the relationship between the user requirements in terms of domain words (with its counterparts 'Items' in the User Requirement Level of [Rolland01]), i.e. what domain word are defined in what Logical Unit. In this respect we believe that some of the validation techniques described in this paper could be valuable to investigate as complimentary validation techniques to those already present in Lyee. Returning to the 'Split'- example of [Rolland01], it would be valuable to apply paraphrasing techniques in order to validate that the chosen domain words to be used as labels for user input, are indeed the correct ones. That is, what domain dependent relationships apply between the domain words in the example? A designer instantiating the meta-model, e.g. formulating a user requirement, need feedback in terms of the correctness of the domain words he or she chooses to use as labels for user input. We believe that the Lyee methodology is extremely suited for applying many of the validation techniques proposed in this paper. An example of this is the short lead times in implementing using requirements, which is feasible in the fast simulation and planning of systems to be designed.

References

[Access] http://www.microsoft.com/office/access/default.asp

[Alexander77] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel, A Pattern Language (New York: Oxford University Press, 1977)

[Appelt88] D. Appelt and A. Kronfeld, A Descriptive Model Of Reference Using Defaults, Technical Report 440. AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, May 1988. [Details]

[Appelt85] D. Appelt, Planning English Sentences. Cambridge University Press, Cambridge, England

[ArgoUML]http:// argouml.tigris.org/

[Boman et al. 97] M. Boman, J. A. Bubenko, P. Johannesson, B. Wangler,, *Conceptual Modelling*, Prentice Hall Series in Computer Science, 1997

[Brachman93] Brachman, R.J., "What IS-A is and isn't: An analysis of taxonomic links in semantic networks", IEEE Computer, October 1993.

[Bubenko86] J.A. Bubenko, Information Systems Methodologies – A research view. In *Information Systems Design Methodologies: Improving the Practice*, 289-318, North-Holland 1986

[Burg95] J. Burg and R. Van de Riet, The Impact of Linguistics on Conceptual Models: Consistency and Understandability. In *Proceedings of the First International Workshop on Applications of Natural Language to Data Bases*, Versailles, France pp. 183-197

[Chen76] P. Chen, The entity-relationship model: Towards a unified veiw of data. *ACM Transactions on Database Systems*, Vol1, no. 1

[Chen83] P. Chen, English Sentence Structure and Entity Relationship Diagrams, *Information Science*, Vol 29, No. 2 and 3

[Costal96] D. Costal, E. Teniente, T. Urpi and C. Farré, Handling Conceptual Model Validation by Planning, Seventh International Conference on Advanced Information Systems Engineering, Springer 1996

[Dalianis92] H.Dalianis, "A Method for Validation a Conceptual Model By Natural Language Discourse Generation", *CAISE-92* International Conference on Advanced Information Systems Engineering, Loucopoulos P. (Ed.), Springer LNCS 593, pp. 425-444, 1992

[Dalianis96] H. Dalianis, *Concise Natural Language Generation from Formal Specifications*, Ph.d. thesis, Department of Computer and Systems Sciences, Royal Institute of Technology, Stockholm 1996

[Dalianis97] H. Dalianis and P. Johannesson, "Explaining Conceptual Models – An Architecture and Design Principles", 16th International Conference on Conceptual Modeling-ER'97, 1997

[Dalianis98] H. Dalianis and P. Johannesson, "Explaining Conceptual Models – Using Toulmin's argumentation model and RST", in the Proceedings of The Third International workshop on the Language Action Perspective on Communication Modelling (LAP98) Stockholm, Sweden, pp. 131-140, 1998

[Eldhadad92] M. Elhadad. Using Argumentation to Control Lexical Choice: A Functional Unification-Based Approach. PhD thesis, Computer Science Department, Columbia University, 1992.

[Elmasri92] Elmasri, R.and Navathe, S.B., *Fundamentals of Database Systems*. Addison–Wesley 1992

[Fillmore68] Filmore, C.H., "The case for case", in *Universals in Linguistic Theory*, eds. Bach and Harms, New York 1968, Holt, Rinehart and Winston

[Fowler97] M. Fowler, Analysis Patterns: Reusable Object Models, Addison-Wesley, 1997

[Goldstein&Storey99] Goldstein, Robert, C., Storey, Veda, C.: "Data abstractions: Why and How?", *Data and Knowledge Engineering 29 (1999), pp. 293-311*

[Gamma95] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns*, Addison-Wesley, 1995

[Geerts00] Geerts G. and McCarthy W. E., "The Ontological Foundaton of REA Enterprise Systems", *working paper, Michigan State University* (August 2000 and being revised for journal submission).

[Gulla96] J. A. Gulla, "A General Explanation Component for Conceptual Modelling in CASE Environments", *ACM Transactions on Information Systems*, vol. 14, no. 2, pp. 297-329, 1996

[Hakkarainen99] S. Hakkarainen, Dynamic Aspects and Semantic Enrichment in Schema Comparison, Ph. D. Thesis, Department of Computer and System Sciences, Stockholm University, Report Series No. 99-009, ISSN 1101-8526

[Hay96] D.C. Hay, Data Model Patterns: Conventions of Thought, Dorset House Publishing, 1996

[Harel87] D. Harel, "Statecharts: a Visual Formalism for Complex Systems", Science of Computer Programming, vol. 8, no. 3, pp. 231 – 274, 1987

[Hofstede97] Arthur H. M. ter Hofstede, Henderik A. Proper and Theo P. van der Weide, "Exploiting fact verbalisation in conceptual information modelling", *Information Systems*, vol. 22, no. 6/7, pp. 349 – 385, 1997

[Hovy&Lin97] Hovy, E. and Lin, C-Y., Automated Text Summarization in SUMMARIST, in Proceedings of the Workshop of Intelligent Scalable Text Summarization, July 1997

[Johannesson 93] P. Johannesson, Schema Integration, Schema Translation, and Interoperability in Federated Information Systems, Ph. D. Thesis, Department of Computer and Systems Sciences, Stockholm University 1993 ISBN 91-7153-101-7

[Kung93] D. Kung, "The Behaviour Network Model for Conceptual Information Modelling", *Information Systems*, vol. 18, no. 1, pp. 1 – 21, 1993

[Lewerenz99] J. Lewerenz, On the use of natural language concepts for the conceptual modeling of interaction in information systems. In Proceedings of the 4th International Conference on Applications of Natural Language to Information Systems,

[Lubars92] M. Lubars, G. Meredith, C. Potts, C. Richter, Object-oriented analysis for evolving systems, Proceedings of the 14th international conference on Software engineering 1992 , Melbourne, Australia, ACM Press New York, NY, USA Pages: 173 - 185

[Maiden92] N. A. Maiden and A. G. Sutcliffe, "Exploiting Reusable Specifications through Analogy", *Communications of the ACM*, vol. 35, no. 4, pp. 55 – 64, 1992

[Maiden98] N.A.M Maiden, M. Cisse, H. Perez, D. Manuel, "CREWS Validation Frames: Patterns for Validating Systems Requirements", Proceedings of The Fourth International workshop on Requirements Engineering: Foundations of Software Quality – REFSQ'98, Pisa/Italy, June 1998

[Mann87] W. Mann and S. Thompson, "Rhetorical Structure Theory: Description and Construction of Text Structures", in *Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics*, Ed. M.Nijhoff, pp. 85 – 95, Dordrecht, 1987

[Martin94] J. Martin and J. Odell, *Object-Oriented Methods. A Foundation*, Prentice Hall 1994

[McKeown88] K. McKeown and W. Swartout: language generation and explanation, in Advanced Natural Language Generation, Ed. M. Zock and G. Sabah, Pinter Publishers Ltd, 1988

[Meyer85]

[Miller90] Miller, George A., Richard Beckwith, Christiane Fellbaum, Derek Gross and Katherine J. Miller: "Introduction to WordNet: an on-line lexical database." International Journal of Lexicography 3 (4), 1990, pp. 235 - 244.

[Moore91] J. Moore and W. Swartout, "A Reactive Approach to Explanation: Taking the User's Feedback into Account", in *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pp. 1 – 48, Dordrecht 1991

[Motschnig-Pitrik&Mylopopoulos92] Motschning-Pitrik, R. and Mylopopoulos, J., "Class and Instances", International Journal of Intelligent and Cooperative Systems, Vol. 1, No. 1 pp.61-92, 1992

[Mylopoulos90] J. Mylopoulos, A. Borgida, M. Jarke and M. Koubarakis, "Telos: Representing Knowledge about Information Systems", *ACM Transactions on Information Systems*, vol. 8, no. 4, pp. 325 – 362, 1990

[Paris88] C. Paris, Tailoring Object's descriptions to a User's Level of Expertise, Journal of Computational Linguistics, Vol. 14, no 3, 1988

[Paris93] C. L. Paris. The Use of Explicit Models in Text Generation. Francis Pinter, London, 1993.

[RationalRose] http://www.rational.com

[Rolland92] C. Rolland and C. Proix "Natural Language Approach to Conceptual Modelling", in *Conceptual modeling, Databases and CASE: An Integrated View of Information Systems Development*, Ed. P. Loucopoulos and R. Zicari, pp. John Wiley, New York, 1992

[Rolland01], C. Rolland, C. Souveyet, R. Kla, Requirements Modeling in Lyee, E-Lyee Consortium 2001

[Toulmin59] S. Toulmin, *The Uses of Arguments*, Cambridge University Press, 1959

[UML97] http://www.omg.org/uml

[WarmerKleppe99]

[Zave84] P. Zave, "The Operational versus the Conventional approach to Software Development", *Communications of ACM*, vol. 27, no. 2, pp. 104 – 117, 1984