Explaining Conceptual Models — An Architecture and Design Principles

Hercules Dalianis
Paul Johannesson
Department of Computer and Systems Sciences
Stockholm University and the Royal Institute of Technology
Electrum 230, S-164 40 Kista, Sweden
email: {hercules, pajo}@dsv.su.se

Abstract

An important activity in requirements engineering is validation, which is the process of checking whether a model correctly represents a piece of reality and the users' requirements. One technique for supporting validation is explanation generation which combines paraphrasing of a specification with question–answer facilities that interactively support a user in exploring a model. In this paper, we propose an architecture and design principles for constructing explanation generation systems for conceptual models. The architecture is partly based on Toulmin's argumentation model, which provides a framework for structuring arguments. We argue that this architecture assists in building explanation generation systems that are highly interactive, provide an adequate amount of information for different user categories, and support a wide range of validation techniques.

1 Introduction

Requirements engineering is the branch of systems engineering concerned with eliciting and specifying the functionality of large and complex software-intensive systems. Put simply, requirements engineering focuses on the early phases of the systems development life-cycle. The result of requirements engineering is a requirements specification, which plays several roles in the process of systems engineering. First, it is the basis for and part of a contract between requirements holders and systems developers. Secondly, it is an architectural, implementation independent drawing of the future system to be built. Thirdly, it provides an explicit basis for reasoning with and among the clients about various qualities of the system.

Requirements engineering consists of several related activities. First, the acquisition of requirements through observation, interviews, analysis of texts, reverse engineering, group work, co-development, etc. Secondly, the modelling and specification of requirements. Thirdly, the analysis of requirements, which includes validation and verification. Finally, communication and documentation of the results of requirements engineering, where traceability and information management are needed.

By validation is meant the process of checking whether a model correctly represents a piece of reality and the users' requirements. Verification, on the other hand,

concerns formal properties of a model, such as syntactical correctness, consistency, and completeness. While verification often can be automated, validation is inherently an informal process requiring subjective, human judgement. In this informal process, it is essential that different types of stakeholders participate, including people with limited knowledge of modelling and systems design. However, people who are unfamiliar with modelling languages may have severe difficulties in understanding and validating a model. Furthermore, the sheer size and complexity of a model may make it difficult even for experienced designers to validate a model.

In order to ease the validation process, several different techniques have been proposed. One approach is to introduce *graphical symbols*, [Kung93], or *user defined concepts*, [Mylopoulos90]. Another approach is to *paraphrase* parts of a conceptual model in natural language, [Rolland92], [Dalianis92], [Dalianis96]. In order to validate large models, a technique called *complexity reduction* has been proposed, which presents views of a conceptual model and hides irrelevant details, [Seltveit93]. *Model simulation* is still another technique, which can be used for observing and experimenting with the dynamic properties of a model, [Harel92], [Zave84]. A validation technique similar to model simulation is *model planning* that allows users to explore a model by constructing plans, i.e. operation sequences that result in states where certain conditions hold, [Costal96]. *Explanation generation* techniques have been used to integrate the techniques mentioned above and provide a uniform interface to the users. Explanation generation extends paraphrasing by including question–answer facilities that interactively support a user in exploring a model. Model simulation can be complemented by explanation generation that guides a user through the execution and explains the system's behaviour in more detail. Explanation generation has previously been used mainly for expert systems, e.g. the Explainable Expert System (EES), [Moore91], which provides natural language explanations of rule-based systems based on Rhetorical Structure Theory, [Mann87]. Recently, there has also been an interest in using explanation generation for validating conceptual models. The most comprehensive work on explanation generation for validating conceptual models seems to be [Gulla96], which discusses an explanation component for CASE environments. This component provides a natural language interface to complex model information and integrates model simulation by explaining a model's behaviour by relating trace information to properties of the model.

The purpose of this paper is to outline an architecture and design principles for an explanation component for object-oriented conceptual models. The work reported upon in this paper extends previous work by providing explanation generation not only for static parts of a model, history traces, and simulation, but also for planning. Another difference is that our work focuses on the interaction between users and systems in the explanation process and provides guidelines for the design of interactive dialogue structures. The structure of the paper is as follows. Section 2 introduces the modelling language and notation used throughout the paper as well as a number of examples. Section 3 outlines the architecture of an explanation component. Sections 4, 5, and 6 elab-

orate on this architecture by discussing explanations of history traces, simulations, and plans, respectively. Section 7 summarizes the paper and gives suggestions for future work.

2 The Delphi Language

The modelling concepts and notation used in this paper are based on the language Delphi, [Höök93]. Delphi is a conceptual modelling language developed by Ericsson Utvecklings AB, where the language has been used for defining specifications of telephone services. The language is supported by a development environment including tools for verification and validation. The formal semantics of Delphi is given by means of first order logic and state machines, which makes it possible to carry out advanced analyses on Delphi specifications. We give here only an informal overview of Delphi as this is sufficient for the purposes of the paper.

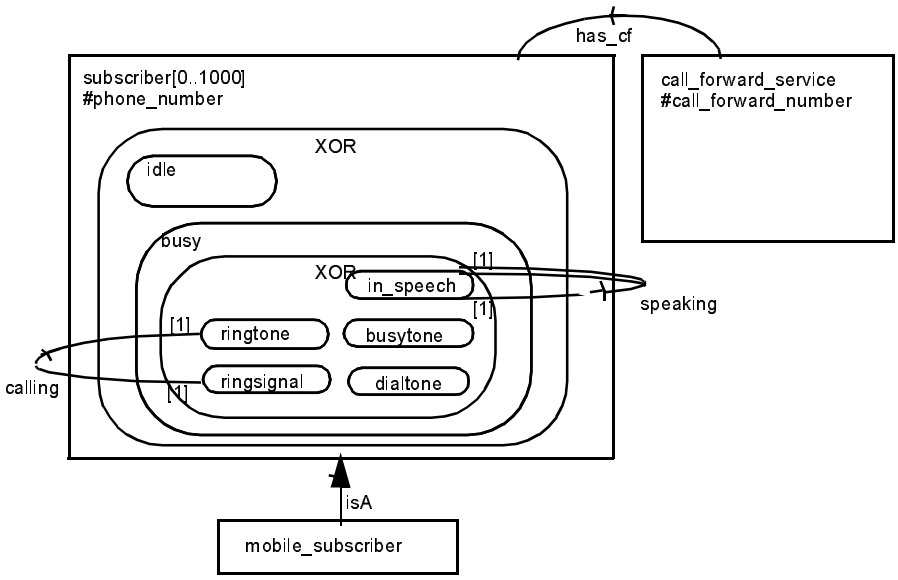

Fig. 1. A Delphi static model

A Delphi specification consists of two parts: a *static model* describing static, structural aspects and a *dynamic model* describing behavioral aspects. The static model describes the *entity types* that exist in a system, their properties, and their relationships. The static model can also include static constraints, called *invariants*, that put restrictions on allowable states. The dynamic model consists of a set of *dynamic rules* that define the state transitions of the system. A dynamic rule consists of a stimulus, conditions, and

conclusions. The informal semantics of a dynamic rule is that if a stimulus occurs in a state and the conditions are true in this state, then the conclusions will be true in the next state.

An example of a Delphi specification is shown in Figure 1 (the static model) and Figure 2 (the dynamic model). This specification defines POTS (Plain Old Telephone Service) extended with the service Call Forward. The static model includes three entity types: `subscriber` with the property `phone_number`, `mobile_subscriber` that is a subtype of `subscriber`, and `call_forward_service` with the property `call_forward_number`. The model specifies the set of states of a subscriber, e.g. a subscriber may be `idle`, have `ringtone`, or have `dialtone`.

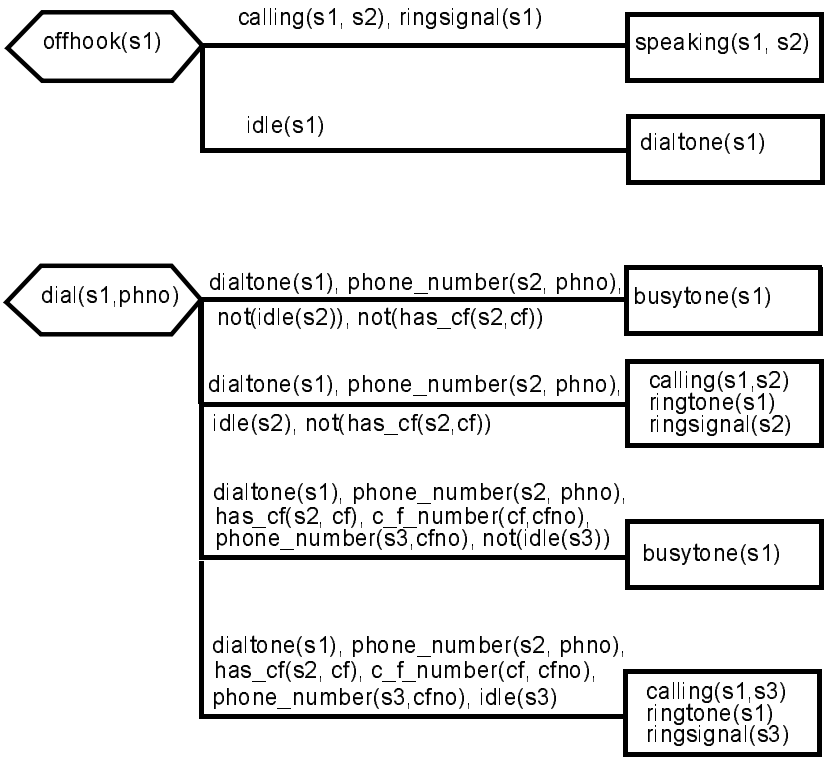

Fig. 2. A Delphi dynamic model

The static model also graphically defines several invariants. For example, if a box is included in another box, the state corresponding to the first box is a substate of the state corresponding to the second box. This means that the static model defines that any subscriber who has `ringtone`, `ringsignal`, `dialtone`, `busytone`, or is `in speech` is also `busy`. The XOR notation indicates that the substates are exclusive, e.g. a subscriber cannot simultaneously have `ringtone` and `busytone`, nor can she be both `busy` and `idle`. Furthermore, the static model defines a number of relations between subscribers, e.g. that one subscriber can be `calling` another. The model also expresses the invariant that if one subscriber calls another, then the first subscriber must have `ringtone` and the second `ringsignal`. Furthermore, the model specifies cardinality constraints for the relations.

The dynamic model consists of two dynamic rules. For each rule, the stimulus is shown within a hexagon, the conditions are placed along a line, and the conclusions are shown within a rectangle. The second rule contains four branches; the first branch states that dialling to a busy subscriber without call forward results in busytone; the second branch states that if one subscriber dials a phone number and the subscriber with this phone number is idle and does not have call forward, then the dialling subscriber will get ringtone and the other one ringsignal; the third and fourth branches state that dialling to a subscriber with call forward results in busytone if the subscriber forwarded to is busy, and otherwise in ringtone at the dialling subscriber and ringsignal at the other one.

A Delphi specification can have instantiations, or *fact bases*, describing a number of entities and their properties and relations at a particular point in time. Some examples of fact bases, which will be used in later sections, are given below.

Fact base A

```
subscriber(john)
phone_number(john, 625118)
idle(john)
subscriber(mary)
phone_number(mary, 500198)
idle(mary)
```

Fact base B

```
subscriber(john)
phone_number(john, 625118)
subscriber(mary)
phone_number(mary, 500198)
calling(mary, john)
ringtone(mary)
ringsignal(john)
```

Fact base C

```
subscriber(john)
phone_number(john, 625118)
subscriber(mary)
phone_number(mary, 500198)
speaking(mary, john)
in_speech(mary)
in_speech(john)
```

Fact base D

```
subscriber(john)
phone_number(john, 625118)
has_cf(cf1)
call_forward_service(cf1)
idle(john)
subscriber(mary)
phone_number(mary, 500198)
idle(mary)
```

3 Architecture for an Explanation Generation System

A basic problem in explaining conceptual models is that of identifying forms of explanations and dialogue structures that effectively support users in understanding the models. Experience has shown that it is not particularly effective to simply paraphrase large fragments of a conceptual model, as people easily get lost in a long text and may even find it unfocused or irrelevant, [Gulla96]. Much more effective are short texts, or visualizations, that answer focussed questions about the structure or behaviour about a model in the context of a particular use case. Especially effective are answers that combine information from different parts of the conceptual model, e.g. from the fact base as well as from the dynamic rules, [Dalianis92]. In order to design an adequate explanation component, two fundamental issues have to be addressed. First, what types of questions and explanations should be supported? Secondly, how should the dialogue between user and system be structured?

When exploring a conceptual model, a user should be able to ask several different types of questions about constructs in the model. There seem to be, at least, four different types of questions in this context. The first type of question is the structural question that asks about the structure of a construct, its properties and relationships to other constructs as well as the constraints pertaining to the construct. The second type of question is the causal question that asks about the cause of a construct, i.e. the events that have made it to come into existence. The third type is the reason question that asks about the reason for a construct, i.e. why the construct has been introduced and what purpose it serves. Finally, there is the procedural question that asks how something is carried out.

A user should not only be allowed to ask isolated questions about a model and get answers, but also to engage in a more structured dialogue with the system involving follow-up questions and more detailed explanations. In order to design such dialogues, some kind of argumentation model is needed as foundation. An attractive candidate for this purpose is Toulmin's argumentation model, [Toulmin59], which describes the structure of arguments. The starting point of an argument is a *claim*, which is a sentence asserting some proposition. This claim is related to *grounds* supporting the claim; the grounds should provide the required basis for accepting the claim. If a presumed listener to the argument is not convinced by the grounds given for the claim, further grounds can be given in support. If the listener is still not convinced, 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 be sufficient to convince the listener, further supportive arguments can be given in the form of *backing*. Backing is used to show that the warrant is reliable and relevant in the context. Normally, backing takes the form of rules at a higher level than the warrant. However, grounds, warrant, and backing may not invariably lead to the required claim – *qualifications* such as "usually" and "possibly" may then be used to clarify the

relationships between grounds and claim. If the claim does not follow with certainty from the grounds, an argument may describe the circumstances where the claim does not hold – these are called *rebuttals*. An argument can thus schematically be described as shown in Figure 3 below.

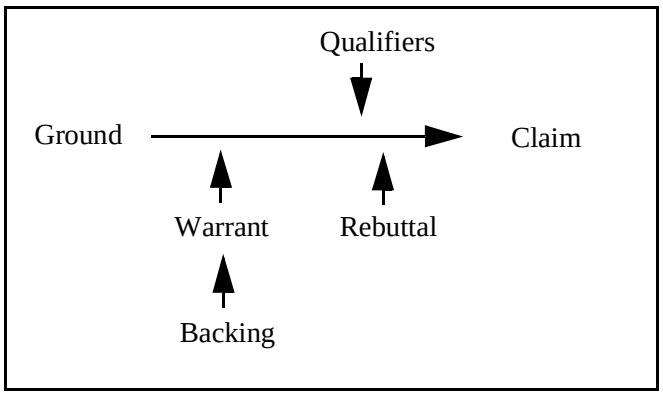

Fig. 3. Toulmin's argumentation model

A major benefit of Toulmin's argumentation model for the purpose of explaining conceptual models is that it can guide the gathering of information from different components of the conceptual model. For example, in an interactive approach, if the user questions the grounds for a specific claim, the warrant can include additional information from the conceptual model and form the basis for a more detailed explanation. In this way, Toulmin's model provides a simple and informative explanation structure that gives an adequate amount of information at the right time and level. Explanations designed according to Toulmin's model are probably most useful for claims on the instance level, i.e. claims about particular objects and their relationships. If a claim is on the instance level, it is natural to structure the argument about the claim in such a way that the grounds are also on the instance level, the warrant on the schema level, and the backing on the meta schema level. This structure gives a correspondence between the central parts of Toulmin's model and the components of a conceptual model. This correspondence can be used to organize the explanation of model constructs as shown in the following sections.

.

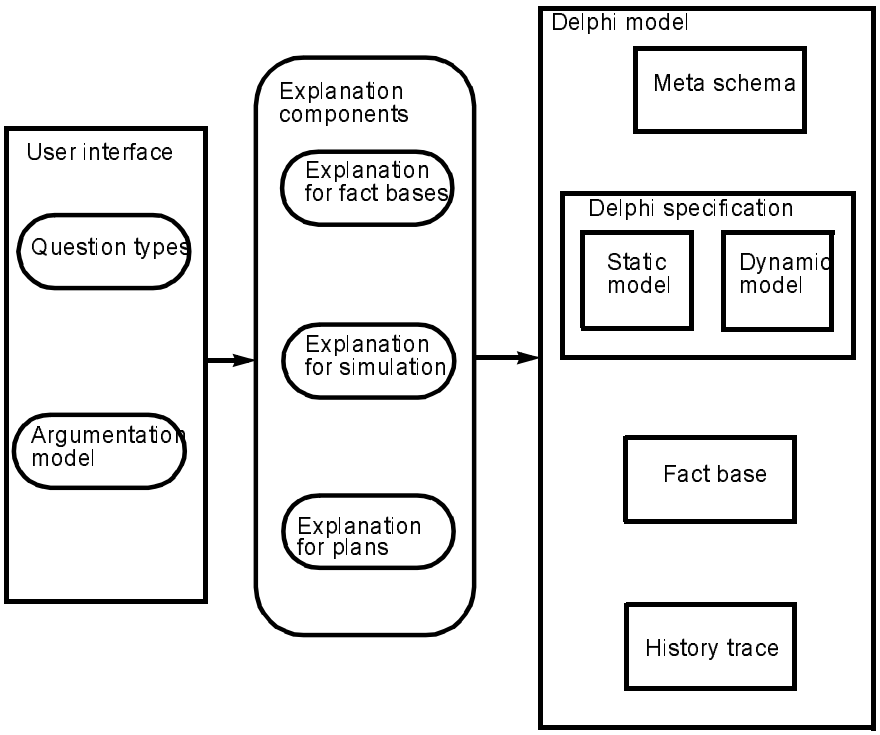

Fig. 4. Architecture of an explanation system

The architecture of an explanation system is outlined in Figure 4. In the rightmost box, the components to be explained are displayed; in addition to explaining a Delphi specification, it is possible to explain the Delphi meta schema, fact bases, and history traces. The explanation component provides three different validation techniques: explanation of instance level information in the fact base, explanation of the results of simulation, and explanation of plans, i.e. sequences of operations for achieving certain goals. All these validation techniques utilize and combine information from several model components, in particular from the fact base and the history trace. The dialogue between user and system is managed by the user interface, which offers the user a set of different questions for initiating explanations. The user interface also uses Toulmin's argumentation model for structuring the system's explanations.

The architecture above suggests the following two design principles for constructing an explanation generation system:

- For each construct in a model, determine which types of questions can be asked about it: structural, causal, procedural, and reason. For each type of question, determine which sources of information that must be used to provide an answer.
- For each model construct and question type, determine how the answer can be divided into smaller units based on the components of Toulmin's argumentation model. Furthermore, determine which follow-up questions are applicable.

In the following sections, we will show a number of examples illustrating how these design principles can be applied.

4 Explanation for Fact bases

Structural question – combining static model and fact base

Explaining fact bases involves answering questions about why specific facts hold true in the current state of the system. A basic question to ask about a fact base is:
– Why does this fact hold?
This question can be interpreted in two ways. First, it can be seen as a structural question asking whether the fact holds as a consequence of other facts and invariants. Secondly, it can be interpreted as a causal question asking for the event, or sequence of events, that caused the fact to hold. For the structural question, there will be an answer only if there is an invariant that provides a sufficient condition for the fact to hold. In such a case, the simplest answer to the fact will be just to display that part of the information base that causes the fact to hold – this corresponds to giving the grounds in Toulmin's argumentation model as explained in Section 3. A more elaborate answer would include the relevant invariant(s) as well as a part of the information base – giving the invariant(s) would correspond to providing a warrant for the fact to be explained. These ways of explaining a fact are illustrated in the example below:

Example 1. Based on fact base B.

USER: *Why is John busy?*

SYSTEM: Because John has ringsignal. (Grounds)

USER: *I do not understand.*

SYSTEM: Whenever a subscriber has ringsignal, he is busy. And John has ringsignal and is therefore busy. (Warrant)

Causal question – combining dynamic model, history trace, and fact base

The causal question about a fact requires for answer an event or a sequence of events, which means that the history trace of the system must be consulted. Also in this case, an answer can be more or less elaborate. The simplest answer is to provide the event(s) that caused the fact to hold – corresponding to the grounds. The more elaborate alternative is to include also the conditions and conclusions of the relevant dynamic rules in the answer – corresponding to a warrant. These alternatives are exemplified in the following:

.

Example 2. Based on fact base C.

USER: *Why is John speaking with Mary?*

SYSTEM: Because John made offhook. (Grounds)

USER: *I do not understand.*

SYSTEM: Whenever a subscriber is calling another subscriber and the second subscriber makes offhook, then the subscribers start speaking with each other. And in this case, Mary was calling John, and John made offhook. (Warrant)

Causal question – larger parts of a history trace

A special problem in answering causal questions about a fact is to determine how many events to include in the answer. In some cases, it is sufficient to include just one single event, namely the event that introduced the fact due to the conclusion part of the corresponding event rule. In other cases, however, several events are needed to account for a fact, for example when the fact is derived from other facts introduced by distinct events. It may also be appropriate to provide a complete event sequence for explaining a fact when the event that introduced the fact is part of a larger use case; in such a situation, all events in the relevant use case should be provided in the explanation. An example of this is given below:

Example 3. Based on fact base C.

USER: *Why is John speaking with Mary?*

SYSTEM: Because Mary made offhook and dialled John, and John made offhook. (Grounds)

USER: *I do not understand.*

SYSTEM: If a subscriber is idle and makes offhook and then dials another subscriber, who is idle and does not have call-forward, and this subscriber makes offhook, then the subscribers start speaking. And Mary was idle, made offhook and dialled John, who was idle and did not have call-forward. (Warrant)

5 Explanation for Simulation

Procedural question – combining dynamic model and fact base

Simulation means to execute dynamic rules in the conceptual model and explore the consequences of these executions. The simplest simulation consists of executing one single dynamic rule, and the basic question is a procedural question asking how the dynamic rule was executed. The answer will tell which branch in the rule was chosen and which conclusions the execution gave rise to. The answer can be more or less detailed, including only facts about the conclusion (grounds) or specifying the dynamic rule (warrant). This is illustrated in the following example:

Example 4. Based on fact base B.

USER: *What happens if John makes offhook?*

SYSTEM: John and Mary will be speaking with each other. (Grounds)

USER: *I do not understand.*

SYSTEM: Whenever a subscriber is calling another subscriber and the second subscriber makes offhook, then the subscribers start speaking with each other. And in this case, Mary was calling John, and John made offhook. (Warrant)

Procedural question – with rebuttal follow-up question

A slightly more complex simulation consists of the execution of a dynamic rule and a particular fact. The natural question to ask is a causal question asking whether the fact will hold after the execution or not. If the answer is yes, this case reduces to the same one as a causal question about a fact in the fact base. But if the answer is no, it is natural to continue with a follow-up question that amounts to a rebuttal, i.e. asks for circumstances under which the specified execution would indeed make the fact true. Thus, the answer to this question would describe the circumstances under which the execution of the dynamic rule would follow another branch that resulted in the fact to hold – if such a branch exists. An example of this case is the following:

Example 5. Based on fact base D.

USER: *If Mary dials John, will he get ringsignal?*

SYSTEM: No, he will not. (Grounds)

USER: *Under which circumstances would he get ringsignal?*

SYSTEM: If John did not have call forward, he would get ringsignal. (Grounds)

USER: *I do not understand.*

SYSTEM: If a subscriber calls another subscriber who is idle and does not have call-forward, then that subscriber will get ringsignal. Mary dialled John, who was idle but he had call-forward, and therefore he did not get ringsignal. (Warrant)

6 Explanations for Planning

A classical planning problem has the following form: given a description of the problem domain, an initial state and a goal, find a plan that leads from the initial state to a final state where the required goal holds. The plan will consist of zero, one or more operations. Previous work on planning for conceptual model verification and validation has focussed on how planning can be used to check that a model satisfies certain desirable properties, such as satisfiability, liveliness of predicates, and non-redundancy of constraints, [Costal96]. We will here outline how planning can be used also for interactive exploration and explanation of a model.

.

Causal question – combining dynamic model and fact base

A basic question about the future states of an information base is the following:
- How can this fact be made to hold?
An answer to this causal question would consist of an event sequence such that starting from the current information base and carrying out the events results in an information base where the requested fact holds.

Example 6. Based on fact base A.

USER: *How can Mary become speaking with John?*

SYSTEM: Mary makes offhook, dials John, and John makes offhook. (Grounds)

USER: *I do not understand.*

SYSTEM: If a subscriber is idle and makes offhook and then dials another subscriber, who is idle and does not have call-forward, and this subscriber makes offhook, then the subscribers start speaking. And Mary is idle, so if she makes offhook and dials John, who is idle and does not have call-forward, then they will become speaking. (Warrant)

Reason question – combining dynamic model and fact base

The planning question above is similar to the causal question in Section 4, as both require an event sequence as answer. However, the planning question is different in the way that it becomes meaningful to ask reason questions about the answer. For example, one could ask: "Why is this event included in the sequence?" An answer to this type of question would take the form: "This event is included because it ensures that these facts are true, and this is needed for the following event, occurring later in the sequence, to happen." In order to produce this answer, one has to consult the conclusion part of the dynamic rule that is the subject of the question, and the condition parts of all dynamic rules of the events coming later in the sequence.

Example 7. Based on fact base A.

USER: *How can Mary become speaking with John?*

SYSTEM: Mary makes offhook, dials John, and John makes offhook. (Grounds)

USER: *Why is the event "dials John" included?*

SYSTEM: Because this event results in the fact that Mary is calling John, and this is required if the event of John making offhook is to result in Mary and John becoming speaking. (Grounds)

This kind of validation through explanation for planning can be continued with explanation for simulation, e.g. by asking whether substituting one event in a plan with another event will result in the same goal holding true. In this way, the explanation component should provide a uniform interface to different kinds of validation techniques and allow for their combination.

7 Concluding Remarks

In this paper, we have discussed explanation generation systems for conceptual models and proposed an architecture as well as design principles for constructing such systems. These principles will assist a designer in building explanation generation systems that are highly interactive, provide an adequate amount of information for different user categories, and support a wide range of validation techniques. In the examples in the previous sections, we have shown only explanations expressed in natural language. However, an adequate explanation component should combine natural language with graphical representations. These representations should provide a user with a graphical view of a system and allow her to obtain explanations of its various components by means of point and click interaction. When a user clicks at a component to have it explained, the system should display a menu of the types of explanations applicable for this particular component – these types are given by the four types of questions introduced in Section 3. The system will then display an explanation combining a graphical view with text.

Another line of research is to investigate the usefulness of a more complex argumentation model than Toulmin's. One candidate for this purpose is Rhetorical Structure Theory (RST), [Mann87], which is a model for describing the structure of coherent text. The basic idea of RST is that a segment of text has a central item, called a nucleus. This nucleus is supported by other segments of text, called satellites, that are related to the nucleus in different ways. For example, a satellite may be a causal link or express evidence for the nucleus, or state a condition that must hold if the nucleus is to be accepted. A main advantage of using RST for explaining conceptual models is that it provides a basis for constructing more coherent natural language output, [Gulla96]. However, using RST has the drawback that it requires a customization for each domain or application, since the typical RST relations do not have any immediate correspondences to conventional conceptual modelling constructs. This is in contrast to Toulmin's model where there are natural correspondences between the model's components and conceptual modelling constructs. These correspondences make it easy to apply Toulmin's model for any specification expressed in an object oriented conceptual model. RST may, therefore, be less attractive for these types of models, but can be useful for semantically richer models that provide, for example, constructs for time and intentionality.

References

[Costal96] D. Costal, E. Teniente, T. Urpi, and C. Farre, "Handling Conceptual Model Validation by Planning", *7th International Conference on Advanced Information Systems Engineering, Springer*, 1996.

[Dalianis92] H. Dalianis, "A Method for Validating 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.

.

[Dalianis92] H. Dalianis, *Concise Natural Language Generation from Formal Specifications*, Ph.D. thesis, Department of Computer and Systems Sciences, Royal Institute of Technology, Stockholm, 1996.

[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.

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

[Höök93] H. Höök, *A General Description of the Delphi Language*, Ellemtel internal report, 1993.

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

[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.

[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.

[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.

[Rolland92] C. Rolland and C. Proix, "Natural Language Approach to Conceptual Modeling", 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.

[Seltveit93] A. Seltveit, "An Abstraction-based Rule Approach to Large-scale Information Systems Development", in *5th International Conference on Advanced Information systems Engineering*, Ed. pp. 328 - 351, Springer Verlag, 1993.

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

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