In: *Proceedings of the 5th European Software Engineering Conference ESEC'95* (Sitges, Spain, September 25-28), 1995

The Treatment of Non-Functional Requirements in MIKE

Dieter Landes and Rudi Studer

Institut für Angewandte Informatik und Formale Beschreibungsverfahren
Universität Karlsruhe, D-76128 Karlsruhe, Germany
e-mail: { landes | studer }@aifb.uni-karlsruhe.de

Non-functional requirements significantly affect and determine the quality of software systems. In this paper it is shown how non-functional requirements are modelled in MIKE, an approach to the development of knowledge-based systems. A semi-formal hypermedia-based model is used to describe the results of the elicitation and interpretation of non-functional requirements and their relationships. Non-functional requirements are the driving force behind the decisions taken in the design phase of MIKE. The impact of non-functional requirements on design decisions and interdependencies between design decisions are explicitly recorded in an additional model in MIKE, thus resulting in a rich documentation of the rationale of design decisions and also providing an important contribution to the traceability of these requirements.

1 Introduction

In recent years, the influence of non-functional requirements (NFRs) on the quality of software systems has increasingly been acknowledged (cf., e.g., [24]). However, the elicitation, specification, and implementation of NFRs, such as, e.g., efficient use of resources, maintainability, understandability etc., are still less understood than the corresponding issues in the context of functional requirements. In particular in the area of knowledge-based systems (KBS), the focus of interest in the research community still lies almost exclusively on aspects related to the functionality of the system. MIKE (Model-based Incremental Knowledge Engineering) [2] is a framework for the development of knowledge-based systems which, among other things, tries to overcome this shortcoming by including the treatment of non-functional requirements into its development cycle. Although MIKE focuses on KBS development, its treatment of non-functional requirements bears the promise to be also applicable in the development of non-knowledge-based software systems since the relevant categories of NFRs and the types of design decisions taken on their behalf are quite similar for both types of system.

MIKE distinguishes several phases in the development cycle. The results of these phases are explicitly described in various models. During knowledge acquisition, the main interest lies on the elicitation and specification of the knowledge required to solve a given task. This expertise does not only comprise knowledge about the application domain, but also knowledge about how the problem in question can be solved. Therefore, the specification of a KBS must, in contrast to specifications in conventional software engineering, not only address *what* functionality the system must provide, but also has to pay attention to *how* the required functionality can be exhibited, i.e. which steps must be performed in order to solve the given problem. Aspects concerning the *realization* of the required functionality, however, are still at a different level and considered in the design phase. In MIKE, the functional specification of the KBS consists of three interlinked views of the application (similar to the object, functional, and dynamic perspectives taken in OMT [27]) and is expressed formally using the specification language

KARL (cf., e.g., [8]). Due to KARL's executability, the specification can be used as an operational prototype, which is refined during knowledge acquisition until it meets the stakeholders' expectations wrt. functional requirements ([1], [3]).

Thus, the design phase is strongly determined by non-functional requirements since design decisions are triggered by deficiencies that the prototype (i.e. the operational specification) exhibits wrt. non-functional requirements. Putting it another way, NFRs constitute the justifications of design decisions and constrain the way in which the required functionality may be realized [16].

Due to their difference in nature, functional and non-functional requirements usually cannot be specified at the same time: "in order to be able to formulate non-functional requirements, the functionality of a system has to be known" ([22], p. 21). However, the techniques for the elicitation and interpretation of functional requirements can also be used in the context of non-functional requirements. NFRs are elicited from future users, project sponsors, domain experts, or other stakeholders and described semiformally in a hypermedia-based fashion in a specific part of the so-called *Structure Model* [20], namely the *NFR context* [15]. The NFR context describes which NFRs the system must fulfil, but also accounts for relationships between NFRs such as, e.g., conflicts. Other sections of the Structure Model focus on the expertise of domain experts concerning relevant concepts of the application domain (concept context) and important activities performed by the expert (activity context). These parts of the Structure Model serve as an intermediate stage on the way to a formal specification of the desired functionality. The NFR context of the Structure Model and its connection to other (parts of) models is described in section 2.

The impact of NFRs on decisions taken in the design phase is explicitly described in an additional model, the so-called *Process Model* [15], which is presented in section 3. The effects of design decisions are recorded in the so-called *Product Model* [15] which constitutes a refinement of the original functional specification. We use DesignKARL [14] as a common formalism for describing both of these models.

The Process Model improves the transparency of the design process by providing an account of the design rationale and contributes to the traceability of NFRs by linking them to the portions of the system design in which they become manifest. The combination of NFR context and Process Model matches the claim that "rationale of different degrees of formality [needs] to be linked to design artifacts of different sorts, as it is triggered by construction activity" ([29], p. 629).

In section 4, aspects concerning the evaluation of NFRs are highlighted while section 5 relates the approach taken in MIKE to similar work.

2 The NFR Context

Functional requirements are collected in the knowledge acquisition phase mainly by conducting interviews with domain experts. Similar techniques can be employed to elicit non-functional requirements; basically future users and project sponsors are asked which features they expect of the future system in addition to the desired functionality. This elicitation process results in a collection of interview transcripts which are stored as protocols in the so-called *Elicitation Model*[1]. Yet, it may not be obvious from these transcripts what the NFRs really are. Consequently, the requirements engineer has to interpret these transcripts to identify relevant pieces of information and formulate NFRs as instances of generic requirement categories such as those used as quality factors in [12] and [13].

The result of this interpretation process is described in the NFR context of the Structure Model. Information in the Structure Model is generally expressed in terms of nodes and directed links of different types. Specific subsets of node and link types constitute contexts.

The basic idea behind the NFR context is to provide the vocabulary to express NFRs in a structured way without forcing the requirements engineer already at this stage to be too detailed and too much concerned with the question of how these NFRs can be operationalized.

2.1 Internal Node and Link Types of the NFR Context

In particular, the NFR context encompasses the node types *requirement*, *requirement category*, *evaluation criterion*, *conflict*, and *argument*. Each node is associated with a textual description constituting its information content. Nodes of type *requirement category* model generic categories of NFRs (e.g., efficiency) or subcategories (e.g., efficiency wrt. storage needs or efficiency wrt. response time). Nodes of type *requirement* denote application-specific instances of such categories.

In a later stage of the development process, the extent to which the system meets the posed requirements has to be evaluated. To that end, it is necessary to identify criteria from the documents in the Elicitation Model that can be used to determine whether a particular requirement has been met. Such criteria are described by nodes of type *evaluation criterion* in the NFR context.

Only in very rare cases, the requirements posed by the different parties involved will be consistent. Rather, some of the requirements may be in conflict. The interpretation of the protocols of the Elicitation Model also aims at detecting and solving conflicts as early as possible. In the NFR context, conflicts are modelled by nodes of type *conflict*. In order to solve identified conflicts, arguments in favour of some of the requirements involved have to be collected. Their description is accomplished by means of nodes of type *argument*.

In addition to those node types, the NFR context contains links of the types *association*, *conflict solution*, *correlation*, *is-a*, and *instance-of*. Links of type *association* can be used in various situations. For instance, the association between requirements and criteria for their evaluation is established by that type of link. Furthermore, in case of a conflict, *association* links point to the requirements causing the conflict by connecting the *conflict* node with the nodes denoting the requirements that cause the conflict. In addition, the *argument* nodes which are relevant for the solution of the conflict are also targets of *association* links originating at the *conflict* node in question.

The result of a conflict's resolution is again expressed by means of nodes of type *re-*

1. In [20], the Elicitation Model is called *protocol context*.

quirement. These *requirement* nodes are connected with the respective *conflict* node using links of type *conflict solution*. The resolution of a conflict goes along with a deactivation mechanism: the requirements that constitute the solution of the conflict supersede those which originally caused the conflict. That is, *requirement* nodes which are the target of a *conflict solution* link are activated while the *requirement* nodes which are target of *association* links are deactivated in case that all the links involved originate at the same *conflict* node.

A mutual influence between two requirements or two requirement categories is expressed using *pos-correlation* and *neg-correlation* links. *Pos-correlation* and *neg-correlation* links express the fact that activities for the benefit of one of the requirements involved have a positive or, respectively, negative impact on particular other requirements.

Is-a links connect two nodes of type *requirement category* in order to express that one of them denotes a subcategory of the other. Similarly, an *instance-of* link indicates that a particular requirement is an instance of a requirement category.

2.2 Inter-Context Connection and Inter-Model Connection

In addition to the link types discussed so far, two additional link types are used to connect the NFR context to the Elicitation Model or to the activity and concept contexts of the Structure Model.

Links of type *elicitation* connect nodes of the NFR context to those documents in the Elicitation Model that triggered the creation of the respective nodes. Thus, elicitation links contribute to the traceability of requirements since they explicate on what basis particular requirements are formulated.

Concepts that are relevant for the considered application and activities that must be performed to solve the given task are described in the concept context or, respectively, the activity context of the Structure Model [20]. NFRs can be connected to the concepts and activities they affect by linking *requirement* nodes with *concept* nodes or *activity* nodes through *reference* links.

Example 1. In a task such as the configuration of elevator systems from elementary parts [23], the project sponsors usually demand that the KBS to be developed should be able to solve the problem faster than a human expert. This statement is recorded in the Elicitation Model (cf. Fig. 1). Its interpretation reveals that it expresses a requirement concerning the runtime behaviour of the problem-solving process. Consequently, a *requirement* node in the NFR context is created and connected to the corresponding text fragment in the Elicitation Model by means of an *elicitation* link. Additionally, the requirement is recorded as an instance of the requirement category *runtime efficiency* by means of an *instance-of* link connecting the respective nodes.

Since the requirement affects the activities that must be performed for solving the problem rather than the data involved, a *reference* link connects the *requirement* node with the *activity* node *ProposeAndRevise* in the activity context. The node *ProposeAndRevise* denotes the employed problem-solving strategy.

Furthermore, the statement in the Elicitation Model also mentions a criterion to evaluate the system's runtime behaviour, namely the effort required by a human expert (which is estimated to be about 30 hours on average). Consequently, a node of type *evaluation criterion* is created in the NFR context which is connected to the corresponding text fragment in the Elicitation Model via an *elicitation* link. Furthermore, the previously discussed requirement is linked to the criterion by means of an *association* link between the respective nodes.

Fig. 1 summarizes the resulting nodes and links of the NFR context.

❏

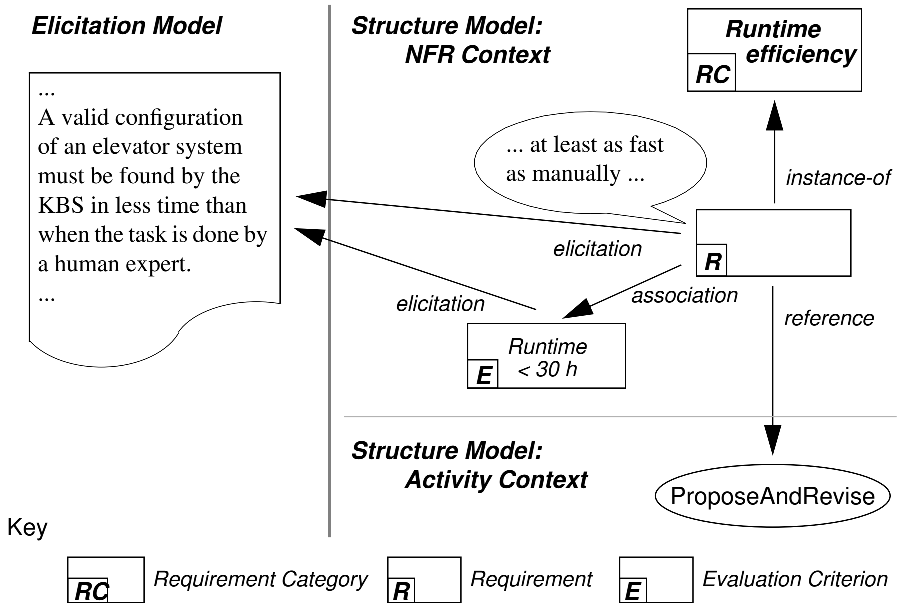

Fig. 1 Schematic depiction of nodes and links in a NFR context

2.3 The NFR Context vs. the IBIS Model

In some sense, the modelling primitives used in the NFR context are related to those of the IBIS method for policy decision making [26]. In IBIS, the argumentative process of solving a complex problem is expressed primarily by means of issues, positions responding to particular issues, and arguments supporting or objecting to specific positions. IBIS issues play roughly the same role as NFRs in the NFR context. Yet, IBIS neglects the origins of issues as well as mutual influences between issues which are explicitly modelled by means of *conflict* nodes, *correlation* links, or *reference* links in the NFR context. Since the NFR context focuses on the identification of NFRs, but does not pay attention to how NFRs can be operationalized, analogons to IBIS positions and arguments need to be provided in the Process Model only. Thus, in MIKE, the identification of relevant issues (i.e. NFRs) and their interdependencies is still cleanly separated from their actual treatment by, e.g., formulating positions and arguments.

3 The Process Model

In the design phase, the focus of interest shifts from the identification of non-functional requirements to their operationalization by means of suitable design decisions. Usually, the efficiency of the prototypical solution that results from the knowledge acquisition phase has to be improved by appropriate algorithms and data structures and the solution might have to be integrated in a fixed hardware or software environment. The development steps taken during the design phase and their underlying rationale are recorded in the Process Model. The required primitives for modelling development steps, their rationale, and their effects are provided by the design language DesignKARL [14].

3.1 Goals and Their Relationships

The non-functional requirements modelled in the NFR context form the basis for the design process. NFRs are viewed as goals to be achieved by means of suitable design decisions. Thus, we take a goal-oriented viewpoint on KBS design as, e.g., [7] do on requirements engineering and [19] do on information systems design.

Goal Decomposition. The refinement of the functional specification, e.g. by using particular data structures, is motivated by NFRs and is effected by elementary design decisions (see below). Yet, top-level requirements formulated in the NFR context tend to be fairly coarse: a top level goal might, e.g., express that "the system should be efficient enough to respond to a query within less than 10 seconds". It is not immediately evident how such a requirement can be met or which portions of the system it affects in the first place. Therefore, goals are gradually decomposed into subgoals until they can be satisfied by performing a collection of elementary design decisions. Two aspects may be used for achieving a decomposition. On the one hand, a requirement (i.e. a goal) may be reduced to a collection of more basic requirements. For instance, the global aim for efficiency may be narrowed down to efficiency wrt. processing time or efficiency wrt. storage needs. On the other hand, the scope of the requirement, i.e. the portion of the design product it refers to, may be made more specific: the aim for efficiency of the complete system may, e.g., be reduced to efficiency of a crucial subtask (or several crucial subtasks). As decomposition continues, goals become more constructive, i.e. they provide a high level outline of how to reach a goal rather than just claim that a requirement must be met. Usually, goals can be decomposed in several ways, i.e. there are alternative ways to reach a goal. Therefore, decomposition of goals generally results in an AND/OR graph.

Characterization of Goals. The top level goals in the Process Model are exactly those NFRs that are described by active *requirement* nodes in the NFR context of the Structure Model. DesignKARL describes *GOALs* in an object-like fashion, i.e. characterizes them by means of particular attributes. Attributes include, e.g., ancestors and successors in the decomposition hierarchy, dependencies on and correlations with other goals, the importance of the goal, its status (*ACTIVE*, *INACTIVE*, *ACHIEVED*, etc.), references to the evaluation criterion used and the portion of the design product affected by the goal, and a textual description of the goal.

Conflicts. During the decomposition of goals, conflicts between goals may arise which must be resolved. Similar as in the NFR context, claims that substantiate goals involved in a conflict are modelled as *ARGUMENTs*. *CONFLICTs* and *ARGUMENTs* are again DesignKARL primitives that are associated with suitable attributes.

Example 2. In the elevator configuration task already mentioned in the previous example, the requirement for acceptable runtime behaviour (i.e. the goal *RtEffTotal*) may be decomposed by exploiting the hierarchical decomposition of the overall task into subtasks: the goal can be met if the two fundamental steps of the problem-solving strategy employed exhibit acceptable runtime behaviour (cf. Fig. 2a). This means that the extension of a partial configuration with new constituents (*Propose*) should be realized efficiently, but this should also hold for the correction of partial configurations (*Revise*) that violate some given constraints on legal configurations. Thus, decomposition results in two goals, *RtEffPropose* and *RtEffRevise*. Both goals can be further decomposed: *RtEffPropose* can be reduced to the goal *RtEffPropParallel* which expresses the fact that in each extension step as many constituents as possible should be added to the partial configuration before checking the constraints (cf. Fig. 2b). Conversely, *RtEffRevise* can be met if the subgoal *RtEffRevSequential* can be achieved, i.e. constraints should be checked after each extension of the partial configuration by a single constituent since corrections are easier to handle in that case (cf. Fig. 2b).

Obviously, these two goals are antagonistic: the former aims at few large extensions of partial configurations, while the latter gives priority to many small extensions. This conflict is recorded in the Process Model by means of the *CONFLICT* depicted in Fig. 3a. Since the constraints are fairly weak in the considered context, only few corrections will be required during the configuration process even if an extension step adds several new constituents. Therefore, the conflict is resolved in favour of the goal *RtEffPropParallel*, i.e. the goal *RtEffRevSequential* is dropped completely. The solution of the conflict also implies a deactivation of the goals causing the conflict (i.e. *RtEffPropParallel* and *RtEffRevSequential*) and the subsequent activation of the goals constituting the solution of the conflict (i.e. *RtEffPropParallel*). The reasons for this particular outcome are captured as an *ARGUMENT* in the Process Model (cf. Fig. 3b).

❏

Relationships Between Goals. Since goals may be decomposed in various ways, the designer has to select a subset of the available alternatives that seems to be most appropriate in the given situation. The motivation for preferring one alternative over another can be expressed by *PREFERENCEs* in a similar way as *ARGUMENTs* explain the reasons for resolving a conflict. *PREFERENCEs* indicate, e.g., according to which criteria an alternative is preferred over another or explicate on which previous decisions the current decision depends.

In some cases, the selection of an alternative may be due to the fact that some potential alternatives are excluded because they are incompatible with previous design activities or, conversely, implied by earlier activities. These circumstances can be expressed as *IMPLICATIONs* or *EXCLUSIONs* between subgoals in the Process Model.

Often, design decisions are taken tentatively and have to be withdrawn at a later stage of the design process after additional information has been gained. This circumstance can be expressed as a *REVISION* which points to the elements of the Process Model that

a. *GOAL RtEffTotal*
INSTANCE OF RuntimeEfficiency;
DECOMPOSED TO RtEffPropose, RtEffRevise;
DECOMPOSITION TYPE OR;
REFERENCE COMPOSED INFERENCE ACTION ProposeAndRevise;
EVALUATION RtManuell;
STATUS ACTIVE;
DESCRIPTION "The system should find a legal configuration in less time than a human expert.";
IMPORTANCE HIGH;
END;

b. *GOAL RtEffPropParallel*
DECOMPOSITION OF RtEffPropose;
NEGATIVE CORRELATION HIGH RtEffRevSequential;
REFERENCE COMPOSED INFERENCE ACTION Propose;
STATUS POTENTIAL;
DESCRIPTION "Extend a configuration with as many constituents as possible in each extension step before checking constraints.";
IMPORTANCE HIGH;
END;

GOAL RtEffRevSequential
DECOMPOSITION OF RtEffRevise;
NEGATIVE CORRELATION HIGH RtEffPropParallel;
REFERENCE COMPOSED INFERENCE ACTION Revise;
STATUS POTENTIAL;
DESCRIPTION "Reduce the effort for correcting inconsistent configurations by removing inconsistencies as soon as possible.";
IMPORTANCE HIGH;
END;

Fig. 2 Parts of a Process Model: a. top level goals, b. subgoals

are now superseded. Revised elements are still retained in the Process Model since they document which design alternatives have already been explored unsuccessfully and thus prevent the designers from wasting resources for exploring them again. However, the status of these elements is changed to *INACTIVE* as a consequence of the revision.

3.2 Elementary Design Decisions

The decomposition of a goal continues until subgoals are reached that can be achieved by means of elementary design decisions. Elementary design decisions effect a modification of a portion of the Product Model [15], i.e. the "refinement" of the functional specification. Thus, elementary design decisions also establish the link between functional requirements (embedded in the Product Model) and non-functional requirements (captured in the Process Model). Four basic types of elementary design decisions are distinguished in MIKE and provided as language primitives of DesignKARL ([16], [14]): *REFINEMENT*, which refines parts of the model by introducing algorithms and

a. *CONFLICT PropParallelVsSequential*
CAUSED BY RtEffPropParallel, RtEffRevSequential;
RESOLVED BY RtEffPropParallel;
ARGUMENTS ArgPropParallel;
STATUS ACTIVE;
DESCRIPTION "Extension by as many constituents as possible before checking constraints implies increased effort for corrections due to the larger number of constituents involved in the propagation of changes.
Sequential determination of constituents immediately followed by checking the constraints ensures low effort for the propagation of changes, at the expense of unnecessarily many checks of the constraints.";
END;

b. *ARGUMENT ArgPropParallel*
SUPPORTS RtEffPropParallel;
OPPOSES RtEffRevSequential;
STATUS ACTIVE;
DESCRIPTION "The effort for the propagation of changes grows with the strictness of the constraints as this implies a higher frequency of corrections and propagations. The strictness of the constraints is relatively low in the current setting; thus the effort for propagation does not have to be spent too many times even if all possible constituents are added in an extension step before the constraints are checked." ;
END;

Fig. 3 Parts of a Process Model: a. conflict between goals, b. argument for solving the conflict

data structures, *STRUCTURE*, which basically indicates the application of structuring primitives [17] to decompose the overall model to smaller, largely self-contained portions, *INTRODUCTION*, which refers to portions of the model which appear without being a refinement of previously existing parts of the model, and *ELIMINATION*, which indicates that portions of the model are no longer needed and, thus, removed.

During the design process, the Product Model passes through a series of states which are determined by the versions of its constituents. The transition between two adjacent states is caused by (a collection of) elementary design decisions. The description of elementary design decisions in the Process Model indicates which portions of a state of the Product Model are replaced by new versions in the following state. Furthermore, *REFINEMENTs* additionally specify by means of logical formulae how the two versions of affected constituents relate to each other in detail.

Elementary design decisions may also be undone as a consequence of revisions.

4 Evaluation of Non-Functional Requirements

The choices between alternative ways to reach a goal are crucial steps in the design process which should rest on as firm a basis as possible. Ideally, the selection of a design alternative is based on a reliable quantitative estimate of what can be achieved with each of the available alternatives. To that end, quantitative measures for the involved non-functional requirements are necessary. In the context of evaluating, e.g., time efficiency, estimates of the algorithmic complexity of the algorithms employed or execution time estimates (cf., e.g., [28], [30]) may give an indication which alternative to choose. Maintainability can be linked, e.g., to complexity: [4] show in a case study how complexity may be used to predict, e.g., the effort required for maintaining a software system. The complexity of a modular software design is determined by the interconnectivity of modules and the internal complexity of individual modules, i.e. the average number of “decisions” in the modules. Individual measures might then be combined using a scheme similar to the one presented in [11].

Alternatively, a tentative decision for one of the alternatives may be substantiated by running the current description of the design product as a hybrid prototype. The prototype then combines parts of the executable functional specification with parts which have already been mapped to the target language, in particular those portions of the design product which are affected by the design decision to be evaluated.

Based on the top-level requirement categories proposed in [12] and [13], the categories summarized in Table 1 are particularly interesting for KBS development using MIKE. Understandability and, to some extent, robustness seem to be fairly specific for knowledge-based applications, while categories like, e.g., security or accuracy, which are important for non-knowledge-based systems (cf., e.g., [5]), are usually less important for KBS development.

Table 1 Important categories of non-functional requirements

Requirement category	Characterization
Efficiency	How well does the system use its resources?
Reliability	To what extent does the system fulfil its mission without failures within a specified time period?
Maintainability / Expandability / Flexibility	How easily can software failures be located and fixed or components or knowledge be added or modified?
Understandability	How easy is it to understand how the system arrives at a solution?
Robustness	How well does the system perform its mission under adverse conditions, e.g. inoperability of parts of the system, incorrect or incomplete case descriptions, missing knowledge etc.?
Portability	How easily can the system be ported to a different target environment?
Reusability	How easily can the system or its parts be used for other applications?
Requirements due to the target environment or target architecture	Which constraints must be taken into account due to the target environment or architecture?

The question whether analytical evaluation of design decisions or evaluation by prototyping should be preferred in a particular context depends on the nature of the involved requirement categories: some requirement categories are amenable to quantitative measurement while others can be evaluated more easily by means of prototyping. Among the categories in Table 1, maintainability / expandability / flexibility and portability are more suitable for "analytical" evaluation, while understandability, reliability and environmental requirements are more amenable to "operational" evaluation using a prototype. Both approaches are possible to evaluate efficiency aspects. Clearly, the decision whether the selection of a design alternative should be based on the evaluation of a prototype or on a quantitative evaluation also depends on factors, such as, e.g., the availability of useful measures, the required effort for the computation of measures vs. the required effort for constructing a prototype, etc. If neither type of evaluation can be used, the selection of a design alternative has to be based on qualitative considerations similar as proposed in, e.g., [19].

5 Discussion

In contrast to most other approaches to KBS development (with the notable exception of recent work by [32] and [31]), MIKE explicitly integrates the treatment of non-functional requirements and the capture of rationale for design decisions. The interpretation of transcripts of interviews with future users, project sponsors etc. that contain indications of such requirements is described in the NFR context of the Structure Model. The operationalization of non-functional requirements, i.e. their decomposition into design decisions that cause an extension or modification of the system design, is recorded in the Process Model.

This approach conveys several benefits. First of all, the design process itself is made more transparent since the record of design decisions and their rationale helps the designers to avoid repeating erroneous design decisions as well as inadvertently undoing earlier design decisions simply because the reasons have got lost why they initially had been made. Furthermore, the explicit connection between requirements and affected portions of the design product on the one hand and between requirements and the protocols from which they were derived on the other hand ensures traceability of requirements. Similarly, this also holds for functional requirements since elementary design decisions link two versions of the design product such that parts of the final design can be traced back to corresponding sections of the functional specification and even further back to parts of the Structure Model concerned with the functional aspects of the KBS. The explicit description of design decisions and their connection to requirements is also indispensable if not only parts of the specification, but also corresponding designs are to be re-used since it is then much easier to determine which design decisions are still valid in a different context and which others must be treated differently.

The ideas concerning the treatment of non-functional requirements in MIKE are considerably influenced by similar work in the context of information systems design, specifically the DAIDA project ([5], [19], [6]). The major difference between this work and the approach taken in MIKE lies in the fact that different types of requirements are considered most relevant in the two domains (e.g., accuracy and security are of minor importance in MIKE) and that MIKE also accounts for the elicitation and interpretation of

NFRs. Furthermore, MIKE also describes elementary design decisions while the decomposition of goals in, e.g., [5], stops at a higher level of granularity. In contrast to MIKE, [5] does not pay attention to the need to revise design decisions, e.g., as a consequence of changing requirements.

The model adopted in MIKE for describing the rationale of design decisions is based on earlier work by [25] and [18] which promote an issue-based style, basically consisting of setting up questions and providing potential answers. In MIKE, a more result-oriented stance is taken and design decisions are linked to requirements more directly. No attempt is made to capture the discourse leading to the preference of one possible solution over others in order to avoid putting too much additional overhead for documentation on the designer. Again, there is no equivalent to elementary design decisions in [25] and [18], nor do these authors address the elicitation and interpretation of requirements in their models.

As a case study, the treatment of non-functional requirements as outlined in the paper has been applied to the configuration of elevator systems [23] mentioned in the examples. However, neither the use of design metrics nor the use of a hybrid prototype for the evaluation of design decisions has been explored in depth in this case study yet.

Currently, work is in progress to implement tools supporting the construction of the NFR context and the Process Model [33]. These tools will be integrated in the MeMoKit tool ([21], [20]) which already supports the treatment of the functional aspects in the development of a KBS with MIKE.

6 References

[1] J. Angele: *Operationalisierung des Modells der Expertise (Operationalization of the Model of Expertise)*. Dissertation, Infix, St. Augustin / Germany, 1993 (in German).

[2] J. Angele, D. Fensel, D. Landes, S. Neubert, and R. Studer: Model-Based and Incremental Knowledge Engineering: The MIKE Approach. In *Knowledge Oriented Software Design*, J. Cuena, ed. IFIP Transactions A-27, Elsevier, Amsterdam, 1993, 139-168.

[3] J. Angele, D. Fensel, and R. Studer: The model of expertise in KARL. In *Proceedings of the 2nd World Congress on Expert Systems* (Lisbon/Estoril, Portugal, January 10-14), 1994.

[4] D.N. Card and R.L. Glass: *Measuring Software Design Quality*. Prentice-Hall, Englewood Cliffs, 1990.

[5] L. Chung: Representation and utilization of non-functional requirements for information system design. In *Advanced Information Systems Engineering*, R. Andersen et al., eds. LNCS 498, Springer, Berlin, 1991, 5-30.

[6] L. Chung, P. Katalagarianos, M. Marakakis, M. Mertikas, J. Mylopoulos, and Y. Vassiliou: Mapping information systems requirements to designs. In *Database Applications Engineering with DAIDA*, M. Jarke, ed. Research Reports ESPRIT Project 892 DAIDA Vol. 1, Springer, Berlin, 1993, 243-280.

[7] A. Dardenne, A. van Lamsweerde, and S. Fickas: Goal-directed requirements acquisition. In *Science of Computer Programming 20*, 1993, 3-50.

[8] D. Fensel: *The Knowledge Acquisition and Representation Language KARL*. Kluwer, Boston, 1995.

[9] B.R. Gaines and M. Musen, eds.: *Proceedings of the 8th Knowledge Acquisition for Knowledge-Based Systems Workshop KAW'94* (Banff, Canada, January 30 - February 4). SRDG Publications, University of Calgary, 1994.

[10] B.R.Gaines and M. Musen, eds.: *Proceedings of the 9th Knowledge Acquisition for Knowledge-Based Sytems Workshop KAW'95* (Banff, Canada, February 26 - March 3). SRDG Publications, University of Calgary, 1995.

[11] G. Guida and G. Mauri: Evaluating performance and quality of knowledge-based systems: foundation and methodology. In *IEEE Transactions on Knowledge and Data Engineering 5(2)*, 1993, 204-224.

[12] IEEE Computer Society: IEEE Standard for a Software Quality Metrics Methodology. IEEE Std 1061-1992, Institute of Electrical and Electronics Engineers, New York, 1993.

[13] S.E. Keller, L.G. Kahn, and R.B. Panara: Specifying software quality requirements with metrics. In *System and Software Requirements Engineering*, R.H. Thayer and M. Dorfman, eds. IEEE Computer Society Press, Los Alamitos, 1990, 145-163.

[14] D. Landes: DesignKARL - A language for the design of knowledge-based systems. In *Proceedings of the 6th International Conference on Software Engineering and Knowledge Engineering SEKE'94* (Jurmala, Latvia, June 20-23), 1994, 78-85.

[15] D. Landes: *Die Entwurfsphase in MIKE - Methode und Beschreibungssprache (The Design Phase in MIKE - Method and Description Language).* Dissertation, Infix, St. Augustin / Germany, 1995 (in German).

[16] D. Landes and R. Studer: The design process in MIKE. In [9], 33/1-33/20.

[17] D. Landes and R. Studer: Mechanisms for structuring knowledge-based systems. In *Database and Expert Systems Applications*, D. Karagiannis, ed. LNCS 856, Springer, Berlin, 1994, 488-497.

[18] J. Lee: Extending the Potts and Bruns model for recording design rationale. In *Proceedings of the 13th International Conference on Software Engineering* (Austin, Texas, May 13-17), 1991, 114-125.

[19] J. Mylopoulos, L. Chung, and B. Nixon: Representing and using non-functional requirements: a process-oriented approach. In *IEEE Transactions on Software Engineering 18(6)*, 1992, 483-497.

[20] S. Neubert: Model construction in MIKE (Model-based and Incremental Knowledge Engineering). In *Knowledge Acquisition for Knowledge-Based Systems*, N. Aussenac et al., eds. LNAI 723, Springer, Berlin, 1993, 200-219.

[21] S. Neubert and F. Maurer: A tool for model-based knowledge engineering. In *Proceedings of the 13th International Conference on Artificial Intelligence Tools, Techniques, Methods and Applications Avignon'93* (Avignon, France, May 24-28), 1993, 427-436.

[22] H.A. Partsch: *Specification and Transformation of Programs*. Springer, Berlin, 1990.

[23] K. Poeck, D. Fensel, D. Landes, and J. Angele: Combining KARL and configurable role limiting methods for configuring elevator systems. In [9], 41/1-41/32.

[24] K. Pohl, G. Starke, and P. Peters, eds.: *Proceedings of the 1st International Workshop on Requirements Engineering: Foundation of Software Quality - REFSQ'94.* Augustinus Verlag, Aachen / Germany, 1994.

[25] C. Potts and G. Bruns: Recording the reasons for design decisions. In *Proceedings of the 10th International Conference on Software Engineering* (Singapore, April 11-15), 1988, 418-427.

[26] H. Rittel and M. Webber: Dilemmas in a general theory of planning. In *Policy Sciences 4*, 1973, 155-169.

[27] J. Rumbaugh et al.: *Object-Oriented Modelling and Design.* Prentice-Hall, Englewood Cliffs, 1991.

[28] A. Shaw: Reasoning about time in higher level language software. In *IEEE Transactions on Software Engineering 15(7)*, 1989, 875-889.

[29] S. Buckingham Shum and N. Hammond: Argumentation-based design rationale. In *International Journal of Human-Computer Studies 40(4)*, 1994, 653-676.

[30] C.U. Smith and L.G. Williams: Software performance engineering: a case study including performance comparison with design alternatives. In *IEEE Transactions on Software Engineering 19(7)*, 1993, 720-741.

[31] A. Stutt and E. Motta: Recording the design decisions of knowledge engineers to facilitate re-use of design models. In [10], 33/1-33/19.

[32] J. Vanwelkenhuysen: Embedding non-functional requirements analyses in conceptual knowledge systems designs. In [10], 45/1-45/15.

[33] F. Zimmer: Werkzeugunterstützung zur Modellierung nicht-funktionaler Anforderungen in MIKE (Tool Support for Modelling Non-Functional Requirements in MIKE). Master's Thesis, University of Karlsruhe (in German, in preparation).