Using Scenarios to Validate Requirements in a Plausibility-Centred Approach

D. Filippidou & P. Loucopoulos
Department of Computation, UMIST, Manchester, M60 1QD, U.K.
{despina, pl}@co.umist.ac.uk

Abstract

The situations within which Information Systems Development in general, and Requirements Engineering in particular are positioned, can be regarded as *design situations*. We subscribe to the view that understanding the problem amounts to solving it. The implication of this is that RE involves a continuous cycle of *problem generation - conjecture - evaluation*. This paper is concerned with the evaluation of designs for a given set of requirements. Our approach is based on a framework that demands a reflective stance whereby the issues, positions and arguments are clearly visible to all participants and a rationale behind the design decisions taken is kept for future examination. Within this context, we are concerned with both the reasoning for the planning of designing the target artefact, and the alternative specifications in the design 'product' themselves.

1 Introduction

Our approach to Requirements Engineering (RE) is centred on a framework for articulating, modelling and reasoning about knowledge pertinent to problem situations that can be collectively termed design problem situations. Our approach, known as Enterprise Requirements Analysis (*ERA*), encourages stakeholders to address the problem of requirements in terms of a paradigm whereby,

- given a set of objectives to be met by the artefact (the artefact's purpose)
- some design is proposed such that if this design were to materialise then the artefact would satisfy the originally stated objectives.

In other words, *ERA* is based on the premise that dealing with requirements cannot be divorced from the form of the required artifact. This close interplay between requirements and designs is essential in attempting to validate requirements.

Goals acquisition and operationalisation deals with the informal aspects of articulating, documenting and agreeing about high-level goals. Such goals are refined into plans that eventually lead to design models. Design generation decisions concentrate on generating such models, the purpose of which is to describe the form of the artefact. Ultimately, design validation decisions are utilised to ensure and assess the suitability of the generated designs. They are concerned with evaluating both the hypotheses chosen to plan the design process, *and* the competing designs, each one being potentially realisable and leading to the implementation of the artefact. These four decision classes are carried out within a design rationale framework that captures, documents and uses the reasons behind the decisions taken (see Figure 1).

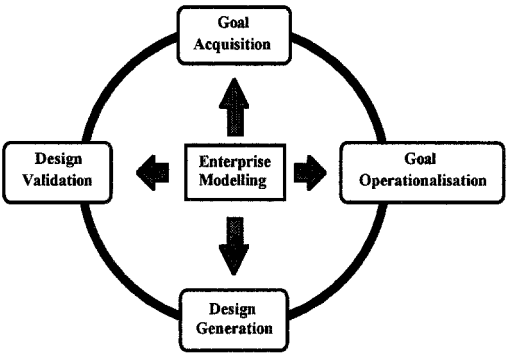

Figure 1: *ERA* Framework

The purpose of this paper is to describe the techniques used to capture and use validation decisions within the *ERA* framework. The paper is organised as follows: Section 2 presents background of different validation and scenario-based techniques because validation is crucial in *ERA*.; Section 3 overviews the way-of-working within the validation layers, namely *elaboration*, *experimentation* and *evaluation*, described in Sections 4, 5, and 6, respectively; Section 7 describes implementation issues, and Section 8 presents conclusions and further work.

2 Background

In this section we introduce the concerns of validation, and give a brief overview of current practice within two areas: (a) the diversity of the validation approaches, and (b) how scenarios have been used in design situations.

2.1 Validation Processes

Validation has been closely related to *verification* and *quality measurement.* Verification has more to do with ensuring that the design products *really* fulfil the stated requirements [12], while quality measurement is more concerned with measuring, in a meaningful way, the quality attributes of the design product [15].

Validation is more about "... certifying the requirements model for correctness against the user's intention" [21]. Therefore, validation ensures that the generated design models are consistent with the customer's intentions, and that they are error and conflict free. Specifically, validation processes can be used to:

- communicate the specifications of the system to the customer, in order to locate misunderstandings;
- ensure that the intended functions are performed;
- exclude unintended functions;
- ensure that no other design solution can improve the quality of the target artefact;
- uncover errors and conflicts;
- correct errors;
- exhibit desired situations and behaviours.

2.2 Validation Techniques

Different categorisations of validation techniques have been established [22], [7] and [18]. We distinguish the techniques developed based on the three dimensions illustrated in figure 2.

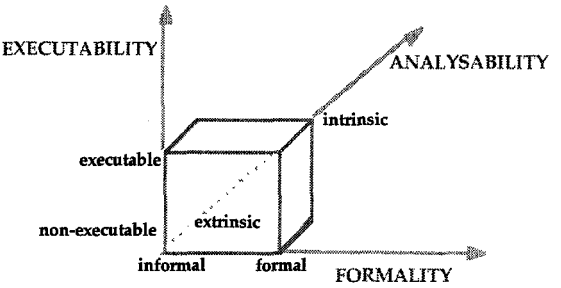

Figure 2: A framework for comparing validation techniques

1. EXECUTABILITY This group of techniques is concerned with the extent that requirements specifications are 'executed'. Techniques can be distinguished to be from non-executable to fully executable [9], [20];
2. ANALYSABILITY This groups techniques according to the extent that analytical methods are applied. Techniques can be distinguished to be from extrinsic (those with no concern for the semantic properties of the specifications), such as those based on visualisation techniques, to intrinsic (those involving the validation of the *contents* and *use* of the specifications themselves), such as those based on syntax-checking, behavioural checking, or scenario analysis [24];
3. FORMALITY This distinguishes validation techniques from formal ones (those based on formal methods like modelling languages) such as in [7], to informal ones, such as those based on brainstorming, open-ended meetings, reviewing, or visualisation techniques [17], [19].

The approach proposed in this paper aims to develop an executable, intrinsic and formal methodology to validation.

2.3 Scenarios in Design Situations

Several approaches have been developed using scenarios in designs [2], [11], [27]. Some use scenarios as *behavioural requirements* [1], [10] and [28]. A behavioural requirement "... describes the way a system behaves, concentrating on the interactions between system and users" [28]. Such behavioural requirements are similar to *use cases*: "A use case is just what it sounds like: It is a way to use the system. Users interact with the system by interacting with its use cases. Taken together, a system's use cases represent everything users can do with it." [13]. Other efforts use scenarios as a concrete *vocabulary* to envision the design process *before* any design is employed [4], [5], to satisfy objectives [23], or to contextualise pertinent knowledge [29].

Scenarios have been applied to many domains, such as the military, business marketing and HCI. These domains share common properties. These properties correspond to the ability of scenarios to refer to *concrete* descriptions of situations and

behaviours, to be open-ended, to focus on contextualised pertinent knowledge and ignore irrelevant ones, to be more informal, and to describe, in general terms, how a particular task gets done. Such scenarios address the incompleteness and scope problem in system requirements [14], and generally, issues of uncertainty, uniqueness and value in requirements specifications, which cannot be addressed by technical rationalisation alone [25].

3 Validation in *ERA* - An Overview

Validation processes in *ERA* are concerned with both the reasoning for the planning of designing the target artefact, *and* the alternative specifications in the design 'product' themselves. The first concern deals with 'design actions' based on 'hypotheses' that aim to bring the enterprise to a desired state. A design action designates any action taken in the course of a design project, whereas a hypothesis represents any option, suggestion or proposal about resolving a problem arising in the design process.

Once hypotheses are set and decisions are taken, participants need then to review and evaluate these decisions, to enable confirmation or refutation. Criticising previous decisions brings out errors, mistakes and difficulties that actually help knowledge grow and assist in finding the 'best' solution to the problem. Testing is about refuting current knowledge, until there is substantial evidence to confirm it. Ultimately, validation processes will produce confirmations or refutations, which will help to maintain the *objectivity* of the design knowledge, as well as to make designers feel confident about the design solution.

The successful development of a validation process is dependent on its ability to revise and correct previous knowledge in the presence of new knowledge. When new evidence appears, previous design actions should be revisited and undergo revision, and probably result in new actions that refute previous ones. In this way, if there are design actions that lead to a satisfying set of goals for solving the problem, then this set of design actions is to be part of an *established design path*; otherwise, it is part of a *rejected design path*, and the designer may backtrack and experiment with alternative options, until the desired outcome is encountered.

The second concern is the validation of the decisions on the alternative 'products' that give the form of the artefact, i.e. the design models. Once design models are developed, these need to be validated *early* in the design process. Design models are validated in terms of their appropriateness, feasibility, consistency, and quality. When alternative competing designs are constructed, it is necessary to provide mechanisms that guide the commitment to one of them, while demonstrating how the others have failed. The validation phase in *ERA* supports this design action guidance to the establishment or rejection of the design models.

Figure 3 summarises the *ERA* validation process. It shows explicitly how both the validation of the *reasoning* to the planning -i.e. the set of hypotheses chosen, *and* the validation of the generated design models can be accommodated in the same framework, resulting in a full validation methodology. The validation process is partitioned into three distinct phases: *elaboration*, *experimentation* and *evaluation*.

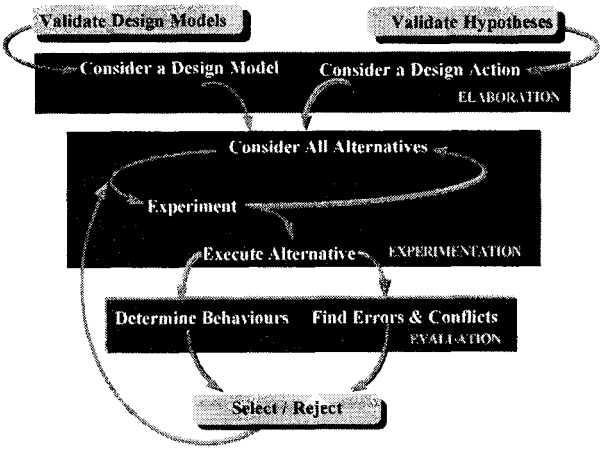

Figure 3: Validation Framework

The process begins with a set of *elaboration* procedures. For design model validation, the approach considers a particular design together with the quality-based expectations of the target system, and responses from the users or other quality measuring techniques. For evaluating the reasoning to planning design actions, the approach mainly considers a specific design action, and identifies dependencies with other design actions, generating inferences, and reviewing the results.

Following these procedures, scenarios are used to consider all candidate alternative options, shown in figure 3 as the experimentation layer. *Experimentation* reconsiders and tries previous alternatives that might bring the enterprise to a desired state. Experimentation is about setting the environment for trial-and-test activities on the remaining alternative options, but not generating new hypotheses. Each alternative is visualised and executed, demonstrating (possibly) unexpected design behaviours, or detecting feasibility inconsistencies, or missing conflicts and errors.

Such errors are used in the next layer, *evaluation*, as inputs for setting statistical comparison views between the alternative designs. Critical errors, such as contradictions, incompleteness and pay-offs are collected, classified as instances of errors, and used to determine whether the particular design model or the altered planning graph will be selected or rejected. The process can continue by considering even more alternatives by more experimentation.

4 Elaboration

Problem solving in *ERA* demands a large amount of *search* between candidate design possibilities. In considering options, practitioners build trees of decisions on hypotheses, with the side branches representing alternative possibilities, some of which are established, and others rejected. When designers traverse these trees of hypotheses, they are actually building 'design paths', each one composed of a sequence of design actions that collectively attempt to bring the enterprise to a new desired state. Practitioners normally choose one such design path, and test to see whether it satisfies

the attributes of the intended system, and subsequently confirm (or refute) the selected design path. In the case of refutation, previous options will need to be re-examined and tested, mainly by implementing backtracking and experimentation activities.

Practitioners need tools and mechanisms that will assist them in searching *efficiently* the options' space, before selecting a design path. It is important, in order to satisfy the need for efficiency, to ensure that the designer gets the most information from a particular part of the design knowledge, and that *all* possible choices are encountered and attempted. Rationale in design is about critically examining the choices given. Critical examination can be achieved only by going through all alternative possibilities, and evaluating the results.

In order to assist the problem solver to efficiently search the design options, pertinent knowledge should be brought together. By using applicable knowledge, practitioners can judge the desirability of an option more *objectively*, and be confident of their decisions. Of course, the process of searching for pertinent knowledge will almost never be complete, and valuable existing information will probably be missed. Domain dependent factors can be ignored, and misjudgements can be made. However, the *ERA* validation approach tries to compromise between concentrating solely on isolated design objects, and considering the whole of the available knowledge.

4.1 Elaborating on Planning - A Contextual Approach

To satisfy the need for collecting relevant knowledge together, validation in *ERA* shifts the focus from individual design actions to *sets* of them. Such sets place emphasis on how 'decisions', i.e. design actions, are logically 'linked' and synthesised. To ensure that design actions can be synergistically used, the validation approach seeks to identify embedded *dependencies* between these design actions. Such dependencies can guide the practitioners to consider the *use* of hypotheses. By analysing these dependencies, in terms of supporting an 'inference-generation machine', the validation process guarantees that any violations of these dependencies will be detected and resolved.

To demonstrate the types of concepts and methods employed, the Air Traffic Control (ATC) case study is used [3]. The enterprise of the ATC centre manages the air traffic flow of some airport. The enterprise is responsible for the safety of aircrafts approaching the airport, and the satisfaction of their passengers. ATC is already partly automated, however, causes such as delays in aircraft flights during the summer months, drive a change process in the enterprise.

4.1.1 Modelling Contexts of Design Actions

The validation process uses *contexts*, and the modelling formalism of an 'action context'. An action context is a logical group of a number of design actions, that externalises and makes explicit embedded dependencies and relations between the design actions themselves. An action context is a step towards to gaining an understanding of the nature and structure of the problem itself.

Putting order and structure into a vast amount of knowledge about design actions, is like providing a geometrical pattern where the system's variables can be observed, used and tested. We view these variables in terms of being *influential* and *dependent*.

An influential variable represents the design action that undergoes criticism, whereas dependent ones represent those design actions logically dependent on an influential variable, or on another dependent variable. However, such dependencies do not need to be stored, since they can be extracted using traversing queries.

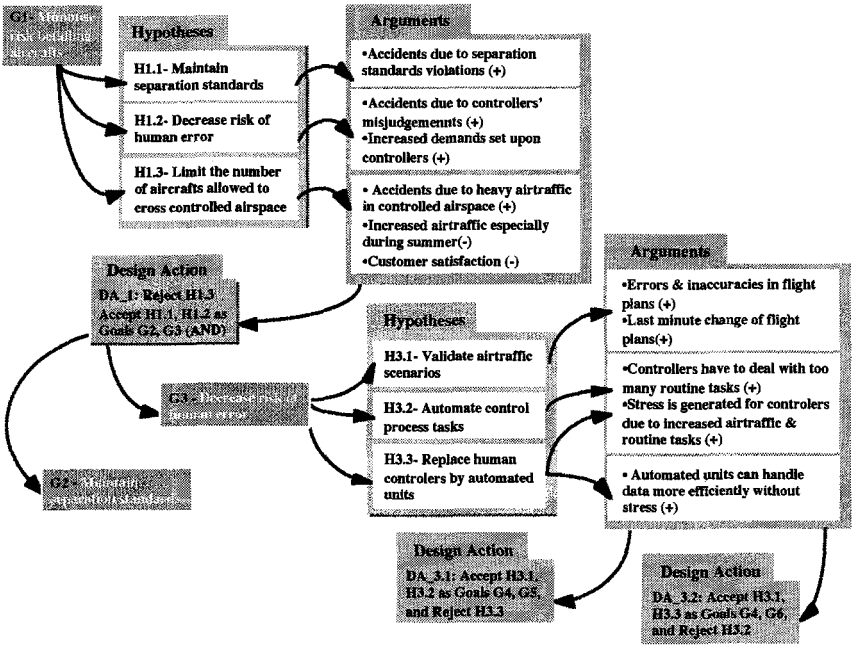

Figure 4: An Example Design Process

Dependencies in *ERA* are used to generate the contents of the action contexts. In order to view how dependencies are captured, we first look at how the *deliberation set* is used to reason about the decisions taken within the design processes. An example reasoning process is given in figure 4: design actions DA_1, DA_3.1 and DA_3.2 (with an OR link between DA_3.1 and DA_3.2), are generated. Such dependencies can be distinguished into three types:

1. *Consequent dependencies* to represent design paths, as sequences of applied design actions; for example, an implicit consequent dependency, based on the diagram of figure 4, would be that DA_3.1 or DA_3.2 follows DA_1;
2. *Priority dependencies* are used to establish logical dependencies between groups of design actions; such dependencies are further specialised by *establishing* and *rejecting priorities*; the establishing (rejecting) priority designates that a particular design action requests that another design action is within an established (rejected) design path; for example, based on figure 4, there is an establishing dependency from both DA_3.1 and DA_3.2 to DA_1;
3. *Justification dependencies* represent argumentative dependencies between design actions; such dependencies are established between design actions when the same arguments justify these design actions.

4.1.2 A Validation Inference Machine

Inferences can be made so that dependencies between design actions are validated. Such inferences drive forward the solution of the problem by being used as further arguments to confirm or refute the initial design actions. Figure 5 gives a diagrammatic view of how inferences are generated and used in *ERA*.

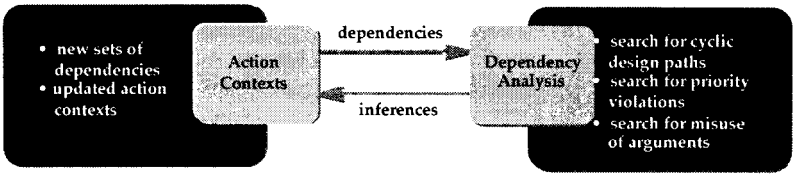

Figure 5: Inference system for Action Contexts

According to figure 5, inferences can be generated as the by-product of interactions between the action contexts and the dependency analysis processes. Once action contexts are built, sets of dependencies are input into the dependency analysis module, to be analysed for validity. Dependency analysis techniques involve searching errors, in terms of cyclic design paths, violations of the priorities between the design actions, and finally for misuses of arguments (i.e. text-based arguments that contradict one another).

Once dependencies are analysed, inferences are generated. If 'negative' inferences, are found, then designers need to detect and correct the source of the error. To do that, designers backtrack and experiment, until the action context of concern is error-free.

4.2 Elaborating on Design Models-A Quality Inspection Approach

Design models are generated as sets of specifications that attempt to describe how the intended properties and functions of the target artefact will work. However, it is more likely that alternative design models will be generated, and the designer will need to select one and demonstrate how the rest will fail. In order to ensure that the designer has made the 'correct' choice, the validation process assists the decision-making task by supporting an environment for quality-based inspection in the design model.

Quality inspection of the generated design models aims at:

- stating the designers' quality expectations of the design model in terms of determining *criteria* to be satisfied;
- measuring the degree that these criteria are satisfied using quality measuring tools, ex. cost-effective techniques;
- capturing how the atomic components of the design models influence the quality of the total model.

The validation process is mostly influenced from the Goal-Rule-Checklist-Metric (GRCM) approach to quality inspection [26]. GRCM hierarchies are built to inspect and evaluate the produced design. The main idea is to initiate quality inspection from a starting goal, and define rules as criteria upon which evaluation proceeds. Checklists are used as guides on how to evaluate the goals, by stating specific properties of the design product that need to be investigated. Finally, metrics are used as values that

measure the degree that checklists are 'satisfied', by implementing the checking and inspection of the product in use.

The validation process uses the following semantics, to inspect and trace the quality influences in the design models:

- *design models* designate the product specifications that state the intended functions and attributes of the target artefact;
- *criteria* represent quality, or non-quality, factors that need to be satisfied;
- *model properties* represent properties of the design models, to which *criteria* refer;
- *components* are atomic objects that constitute the contents of the *design models*;
- *component properties* designate properties of the model *components* that impact the overall *model properties*;
- *metrics* represent values that are the by-product of the empirical estimations of the designers, or the outputs of any evaluation and measurement techniques.

What the elaboration phase of the validation process discovers is a set of metrics, or more accurately a new set of arguments, that can be used to confirm the decision to select or refute the design model under criticism.

5 Experimentation

In most cases, the problem will not be solved by the current set of options or design objects employed, and the designers will try to find new ones that will solve the problem. Hypothesising on different design objects builds new 'design paths', that can lead to other versions of the planning and the specifications for the target artefact. However, finding those 'alternative' and 'possible' paths is an iterative process.

The feasibility of implementing this process of finding a new acceptable solution is largely dependent on the designer's creative skills. Accordingly, a crucial factor is the support of a *systematic* way of selecting and trying each of the 'pending' solutions, instead of deliberating from the beginning. Broadly, such a technique would be based on going through all the design possibilities encountered, but not tried, and attempt each alternative option. *Experimenting-in-action* can be successfully employed to build 'versions of artefacts', by intelligent combination of the alternative options set.

Through experimentation, participants set 'what-if 'questions. Whilst 'what-if' cases are experimented with, designers are actually separating the design process into *testing episodes*. A testing episode is represents the alternative choices selected within a context of reference. Putting design objects into episodes is like arranging the actors in a movie, where the director tries different 'acts', settings, props, and scripts, until a better result is achieved. Neither the 'result', nor the 'end' of the episodes are known from the beginning. We define sequences of successive episodes as *scenarios*. Scenarios in $\mathcal{ERA}$ are used to designate the sequences of the experimentation in which designers engage, in order to generate alternative 'versions' of design artefacts.

Scenarios are generated to systematically consider all alternative options. They are automatically generated by re-arranging the environment with different choice routes. To generate a scenario is to take a current action context or design model, and take different decisions upon the option set. Once scenarios are generated, new action contexts and design models are generated. This process is depicted in figure 6.

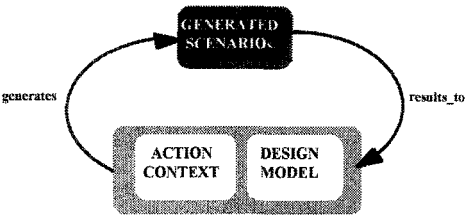

Figure 6: Scenario generation

Sequences of experimentations on choice decisions result in new design products, and scenarios correspond to the paths traversed. Such design products have the form of new, or improved, action contexts and design models, (the two basic concerns of the validation process). Explicit tracing of the sequences of experimentation steps, i.e. the scenarios, that led to introducing new design products, will give rise to attempts to *refute* the initial hypotheses established. Scenarios are used in *ERA* as attempts to refute the initial hypotheses, in order to confirm their rightness or falsifiability. By testing scenarios, two types of 'critical' arguments will be generated:

- CONFIRMATIONS that designate confirming evidence, generated when every attempt to falsify the initial hypotheses presented is unsuccessful;
- REFUTATIONS that designate refuting evidence, generated when an attempt to falsify the initial hypotheses is successful, *and* exposes errors in the initial solution.

Confirmations and refutations, as the user's response to examining specific design alternatives, are then used to establish or reject design choices. In this sense, valuable empirical experimentations and tests from a host of stakeholders and designers in not lost, rather it is reviewed and examined by another host of people.

6 Evaluation

Evaluation activities in the *ERA* validation process are concerned with: (i) detecting errors; (ii) classifying errors; (iii) using errors to build comparative arguments; (iv) resolving errors; and, (v) recording decisions about errors.

The evaluation phase represents elements of *failures*. Such failures capture those errors that designers consider in order to take decisions on design contexts, or on design models. Failures can be either accepted, and thus resolved, or rejected, and therefore ignored. Decisions to accept or reject failures encountered are recorded, to assist the tracing of 'mistakes' in the design process that lead to an unsuccessful result. Failures are used as entries in statistical charts, to generate comparative views for alternative designs. There are three basic failure criteria used in *ERA*:

1. CONTRADICTIONS, generated when truths about the influence and use of arguments, as well as the decisions on the DESIGN MODELS, are contradicted;
2. INCOMPLETENESS, to trace any parts in the design process that are not fully developed;
3. PAY-OFFS, to designate the extent that the generated design solutions are feasible.

A complete description of the techniques and the methodology that uncovers and resolves errors is out of the scope of this paper (the interested reader is referred to [8]).

7 On-going Implementation

A prototype to implement validation of designs is under development. The prototype is an application component of the DELOS (Development Environment Located Over SIS) architecture [16]. DELOS is based on the employment of Java as the single development language, and SIS as the single repository and central element of the development environment (more information on SIS can be found in [6]). A snapshot of the types of objects in the repository is given in figure 7.

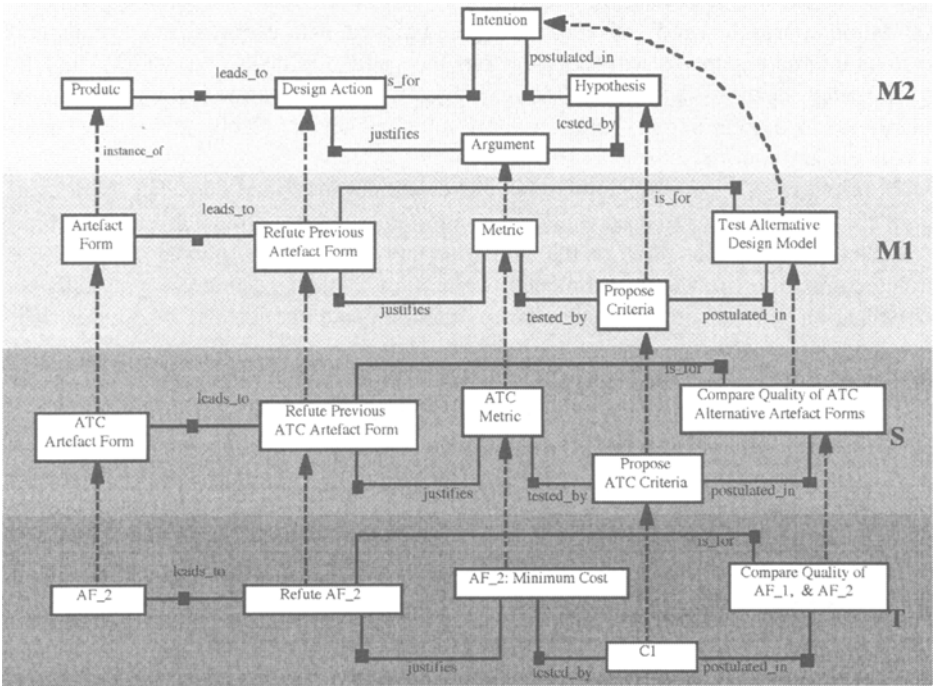

Figure 7: A snapshot from SIS

Figure 7 shows an example of the deliberation process to validating alternative design models. Different levels of abstraction correspond to different instantiations of the models established. Specifically, the objects in the M2 level define a meta-meta-model composed of the necessary semantics to describe the objects defined in the M1 meta-level of abstraction. TheS level describes the application-dependent objects, whereas in the T level, raw data entries are recorded. Visualisation techniques, such as dynamic interactive animation, icons, 3D graphs, etc. are used to give more comprehensive views of the knowledge in the repository.

8 Conclusions

The approach proposed in this paper provides the mechanisms for confirming or refuting generated designs, based on a more formal view to validation. A key feature in

the approach is the use of scenarios to capture the sequences of steps followed to experiment on the set of choices given, in order to ensure the "fitness" of designs.

Scenarios are traditionally viewed as fragments of dynamic visualisations, or text-based stories. They are considered as visualisations of the design knowledge, rather than as knowledge sources themselves. A few approaches try to base scenarios on more concrete grounds, mainly by suggesting inquiry-based techniques, search of requirements deficiencies, analysis of generated claims, or elaboration with tasks and behaviours. However, what these approaches all share is lack of a more formal basis of how scenarios are generated, recorded and used. To be successful in introducing new validation approaches and tools into the design practice, it is essential that we support systematic and automated ways to carry out the validation tasks. We believe that by considering scenarios as 'active' design objects, the generation of which can be automated to a great extent, solutions may emerge that are likely to be acceptable, successful and useful.

Current work focuses on automating the techniques described here. Feedback came from the use of the validation techniques and methodology within the ESPRIT project ELEKTRA. The tool that supports the validation activities is a component of the DELOS architecture. 3D browsing techniques, animation, SGML generators, and different visualisation views, are used in order to communicate the design behaviour. The development of the tool treats scenarios as concrete design objects that can be documented, monitored, deliberated, agreed, and used to drive the design process.

Future work on scenario-based validation techniques will involve investigation of more complex relationships between the generated scenarios and automated generation of domain-dependent scenarios. The former will attempt to inspect how alternative scenarios interact, whereas the latter will identify possible scenarios on the basis of domain-dependent knowledge.

References

[1] Anderson, J.S., & Durney, B. (1993). *Using Scenarios in Deficiency-driven Requirements Engineering.* Paper presented at the IEEE International Symposium on Requiremnts Engineering (RE'93), San Diego, California.

[2] Benner, K. M., Feather, M. S., Johnson, W. L., & Zorman, L. A. (1993). *Utilizing Scenarios in the Software Development Process.* Paper presented at the IFIP '93, Holland.

[3] CAA. (1983). *CIVIL AVIATION AUTHORITY, A report on the supplying by the Authority of navigation and air traffic control services to civil aircraft.* London: Her Majesty's Stationery Office.

[4] Carroll, J. M. (Ed.). (1995). *Scenario-Based Design: Envisioning Work and Technology in System Development*: John Wiley & Sons, Inc.

[5] Carroll, J. M., & Rosson, M. B. (1992). Getting Around the Task-Artifact Cycle: How to Make Claims and Design by Scenarios. *ACM Transactions on Information Systems, 10* (2, April), 181-212.

[6] Constantopoulos, P., & Doerr, M. (1994). *The Semantic Index System: A Brief Presentation* [http://www.ics.forth.gr/proj/isst/Systems/SIS/].

[7] Dubois, E., Du Bois, P., & Dubru, F. (1994, May 17-20). *Animating Formal Requirements Specifications of Cooperative Information Systems.* Paper presented at

the Second International Conference on Cooperative Information Systems -CoopIS, Toronto, Canada.

[8] Filippidou, D., & Loucopoulos, P. (1997). *A Scenario-based Approach to Considering Alternative Designs* (Technical Report ISE-97-1). Manchester, UK: UMIST, Department of Computation, Information Systems Engineering Group.

[9] Gulla, J. A., Willumsen, G., Lindland, O. I., & Solvberg, A. (1994). Executing, Viewing and Explaining Conceptual Models. *IEEE*, 166-176.

[10] Holbrook, C. H. (1990). A Scenario-Based Methodology For Conducting Requirements Elicitation. *ACM SIGSOFT Software Engineering Notes, 15*(1, January), 95-104.

[11] Hsia, P., Samuel, J., Gao, J., Kung, D., Toyoshima, Y., & Chen, C. (1994). Formal Approach to Scenario Analysis. *IEEE Software, 11*(2), 33-41.

[12] IEEE. (1996). *Glossary of Software Engineering Technology, IEEE Std.* New York.

[13] Jacobson, I. (1995). The Use-Case Construct in Object-Oriented Software Engineering. In J. M. Carroll (Ed.), *Scenario-Based Design: Envisioning Work and Technology in System Development*, (pp. 309-336): John Wiley & Sons, Inc.

[14] Kavakli, E., Loucopoulos, P., & Filippidou, D. (1996). *Using Scenarios to Systematically Support Goal-Directed Elaboration for Information Systems Requirements*. Paper presented at the ECBS'96, Friedrichshafen, Germany.

[15] Kitchenham, B., & Pfleeger, S. L. (1996). The Elusive Target. *IEEE Software*, 12 - 21.

[16] Klimanthianakis, P., & Loucopoulos, P. (1996, 6-9 November). *DELOS : Development Environment Located Over SIS*. Paper presented at the OOPSLA'96, Workshop on Integration of Object-Oriented and Web Technologies, San Jose, California.

[17] Kramer, J., & Ng, K. (1988). Animation of Requirements Specification. *Software Practice and Experience, 18*(8), 749-774.

[18] Lalioti, V. (1995). *CineVali: a cinematographic validation of conceptual specifications*. PhD Thesis, UMIST, Manchester, U.K.

[19] Lalioti, V., & Loucopoulos, P. (1993). *Visualisation for Validation*. Paper presented at the 5th International Conference on Advanced Information Systems Engineering (CAiSE•93), Paris, France.

[20] Lindland, O. I., & Krogstie, J. (1993). *Validating Conceptual Models by Transformational Prototyping*. Paper presented at the 5th International Conference on Advanced Information Systems Engineering (CAiSE•93), Paris, France.

[21] Loucopoulos, P., & Karakostas, V. (1995). *System Requirements Engineering*. (1st ed.). London: McGraw Hill.

[22] Lubars, M., Potts, C., & Richter, C. (1993). *A Review of the State of the Practice in Requirements Modelling*. Paper presented at the IEEE International Symposium on Requirements Engineering, San Diego, California.

[23] Potts, C. (1994). *Requirements Completeness, Enterprise Goals and Scenarios*, Dagshtull.

[24] Potts, C., Takahashi, K., & Anton, A. (1994). *Inquiry-Based Scenario Analysis of System Requirements* (Technical Report GIT-CC-94/14): College of Computing, Georgia Institute of Technology, Atlanta.

[25] Schön, D. A. (1987). *Educating the Reflective Practitioner*. San Francisco: Jossey-Bass.

[26] Tervonen, I. (1996). Support for Quality-Based Design and Inspection. *IEEE Software*, 44-54.

[27] Thebaut, S. M., Interrante, M. F., & Burch, T. F. (1990). *Marcel: A Requirements Elicitation Tool Utilizing Scenarios*. Paper presented at the 4th International Workshop of Computer-Aided Software.

[28] Wexelbat, A. (1987). *Report on Scenario Technology* (Technical Report STP-139-87). Manchester, UK: MCC.

[29] Wirfs-Brock, R. (1995). *Designing Objects and Their Interactions: A Brief Look at Responsibility-Driven Design.* In J. M. Carroll (Ed.), *Scenario-based Design: Envisioning Work and Technology in System Development*, (pp. 337-360): John Wiley & Sons, Inc.