ACQUIRING DESIGN KNOWLEDGE THROUGH DESIGN DECISION JUSTIFICATION

Department of Civil Engineering, Stanford University, Stanford, CA 94305, U.S.A.

Currently design documentation rarely records the designer's decision process or the reasons behind those decisions. This paper describes an effort to improve design documentation by having the computer act as an intelligent apprentice to the designer to capture the rationale during the design process. The apprentice learns about the features that make a specific case different from the standard. Whenever the designer proposes a design action that differs from the apprentice's expectations, the interface will ask for the designer for justifications to explain the differences. Later queries for design rationale are answered using a combination of the apprentice's domain knowledge and the designer-supplied justifications. The apprentice model is being implemented in a prototype system called ADD (Augmenting Design Documentation). The initial focus of the work is on HVAC (Heating, Ventilation, and Air Conditioning) design. Our starting point for implementing the apprentice model is observing how people develop HVAC system designs and then explain those designs.

1. Introduction

The lifecycle of civil engineering facilities can be measured in decades—a long period of time during which a facility may undergo substantial changes. Moreover, most facilities are highly complex and require substantial time from initial planning to final construction. The agencies involved in the realization of these facilities are often required to exchange information and communication during the facility development life-cycle a critical issue. Huge amounts of data and knowledge are produced and consumed during the various project phases. However, the final design documentation for the facility contains primarily drawings and numbers. Very little, if any, of the decisions and rationale for those decisions is captured in current documents. Generally, the documentation does not record the intermediate decisions or the reasons behind those decisions. This information is what we call the design rationale or project-specific knowledge (Howard, et al., 1989) or design intent (Ganesan, et al., 1991). The lack of design rationale in project documentation makes it difficult for users to understand the design. The project documents are developed by the designers, but they are consulted by many different users. During the development of a project, the documentation is used as a communication medium among the different trades involved in the project.

Received 21 June 1991 and in revised form 24 October 1991.

Plan checkers use the documentation as a medium for verifying whether the project is consistent with the codes and standards. Contractors use the documentation as a plan for realizing the artifact. Designers use the documentation to support redesign and as a repository of experience. To capture design rationale, we have developed a model for an intelligent design operation that acts as an apprentice to the designer. As an apprentice, it learns about the features that make a specific case different from the standard. Whenever the designer proposes a design action that differs from the apprentice's expectations, the interface will ask the designer for justifications to explain the differences. Later queries for design rationale are answered using a combination of the apprentice's domain knowledge and the designer-supplied justifications. The model to be implemented in a prototype system called ADD (Augmenting Design Documentation), is described in Sections 4 and 5.

The implementation of the ADD prototype focuses on the preliminary design of HVAC systems (Heating, Ventilation and Air Conditioning) for commercial office buildings. There is a cost of choosing elements (such as ductwork, pipes, heaters and chillers) and configuring them to provide the proper environmental control. HVAC system design was selected as the sample domain for this project because of the nature of the design task. It is a goal-driven procedure—find a satisfying set 'V' of parameters' values considering a set 'C' of design constraints and a set 'E' of evaluation criteria (Simon, 1981). Furthermore, the domain is well-understood, with much first-principles knowledge underlying the behavior of the desired artifact. However, the design activity is still ill-structured with incomplete or changing specifications and with no predefined path from the initial to the goal state. This non-deterministic aspect of the design makes the domain of HVAC system design interesting for capturing rationale. After explaining our choice for the application domain, the other main explanation needed is concerned with the design phase selected to carry our study. As in many other design domains, such as in software design (Guindon, 1990a, b), the major impact on the design comes from the decisions taken during the high-level design. In the case of HVAC system, these decisions include the selection of a main HVAC system type, the cooling and heating media equipment, the ductwork and piping selection, and finally the rough location of those components into the floor plan.

Figure 1 presents an example of the preliminary ductwork layout design activity, illustrating the configuration aspect of the design. Figure 2 shows a HVAC project proposal reporting the results of the major decisions made during the design process. Both figures are part of a design proposal package of a selected project. The final decision results are reported there (in this example, a closed loop ductwork layout and a direct expansion cooling system). However, neither the project proposal nor the designer's own notes contain the reasoning to explain why those choices were made. The focus of our work is to capture this information about the design process to allow a better understanding of the artifact design. The knowledge for developing this research came from analysis of design session videotapes (Garcia and Howard, 1991). The impact of having such information in later stages of the design is being studied in (Rafiq, 1990).

Our research proposes a computational model for assisting designers in documenting design cases. Section 2 explores the design documentation problem in greater detail. The documentation problem has been treated over different perspectives about recording argumentation as noninterpreted text, to capturing the basic principles of the domain to produce a complete explanation. Section 3 will outline the relevant contributions of previous work in this area. Section 4 describes ADD's design rationale model, followed by an overview of the HVAC design problem in Section 5. Then we present ADD's model for acquiring and retrieving design rationale in Section 6. Section 7 describes the prototype architecture for implementing the model.

2. The design documentation problem

Design documents are written records intended to provide information about a project. Each kind of documentation is developed to answer certain kinds of questions. However, there are usually many questions relevant to subsequent engineering activities that are
difficult or impossible to answer from available documents. Many of these questions are related to the design process that led to a specific device structure, including questions such as:

- Which constraint was the designer trying to satisfy or violate in making this decision?
- In what context was the decision made?
- Did the designer consider issue Y?
- Which alternatives did the designer consider?
- Did the designer consider alternative Y?
- Why didn’t the designer consider alternative Y?

Even though the questions that are raised about the design can be very diverse, we can recognize some patterns in them. Our observations on the documentation users in the form of HVAC system design led us to identify three different patterns of questions: the WHATs, the WHYs and the WHY-NOTs. The first type of question represents the desire to know what was done; for example, what is the value of some device form or what was the scenario for deciding about a specific form? The second type of question is concerned with the context in which the decisions were made, what issues were considered, and in what stage of the design process the decision was made. The last set of questions illustrates the designer’s conjectures on what should have been done; for example, what would be the impact on the design if instead of ‘x’, we chose ‘y’?

A design rationale document should provide the right amount of detail for each of these cases, and it should capture all information that other users will need later. Our observations on the preliminary HVAC system design sessions indicated that the tool should contain the following information in a machine-readable form; i.e., one piece of information can be used to infer, retrieve, access or check other pieces of information:

- the parameters and values of the final devices
- drawings for visualization of the final devices
- the decisions involved in the design
- the rationale supporting those decisions

Existing design documents do a reasonably good job of conveying the first two items. However, the decision points are usually not recorded explicitly, and rationale is rarely very captured. Our focus is on the rationale. The design rationale should include:

- hypothetically proposed
- assumption made
- constraints that had to be relaxed in order to make valid a decision
- source of those constraints (such as code, specifications and preferences)
- evaluation functions applied
- changes in previous parameter values
- designer’s preferences, represented either as constraints or as evaluation functions (frequently derived from experience)
- decision history (the pattern created by the previous decisions create a unique scenario for the specific decision)

The above parameters illustrate knowledge that has not been recorded in design documents, but is important for documentation users. When the rationale is not available, user representatives and plan checkers have to ask the designer. Sometimes, they use official forms for requesting information. In addition, phone calls, notes and meetings lengthen the list of ways for requesting design explanations. Our observations have helped us to define a representation and a knowledge acquisition model for design rationale in the area of HVAC system design.

3. Background

Improving design documents has been a recent research topic in engineering, computer science and information systems. Researchers have proposed different design rationale models for supporting documentation in different computational models for acquiring rationale. Although the basic goals are similar, the models differ in the ways that they aid designers in recording and retrieving design rationale, both during and after the design process.

There are three major models for representing design rationale: argument-based design rationale, action-based design rationale and model-based design rationale. In the first approach, rationale is represented as a set of arguments (pros and cons) attached to issues, and the issues (decisions) are interconnected. Existing applications of this methodology differ from each other in the type of relationships allowed between issues and between arguments and issues. The Issue-Based Information Systems (IBIS) method (Kunz and Rittel, 1970) is a significant example of such methodologies. IBIS has been the basis for many computer systems dedicated to capture design rationale since the seventies. The method consists of representing a decision as a network containing:

- an issue node representing the decision being made
- position node connected to issues representing the alternatives for an issue
- argument nodes (pros or cons) connecting to position and issue nodes.

A number of prototypes implement partially IBIS method. gIBIS (Conklin and Begeman, 1988) is one of the most successful ones. Successor, generalization, specialization, replace, suggest, similar, and question are the relationships allowed in gIBIS. Extending IBIS, the Procedural Hierarchy of Issues (PHI) (McCall, 1987) method introduces the dependency relation. Mikropolis implemented PHI method. The notion of goals was introduced in Design Representation Language (DRL), implemented in Sybil (Lee, 1990). Using a similar approach (although not explicitly based on the IBIS model), Designworld (Genesereth, et al., 1989) proposes to attach rationale to design decisions as noninterpreted text.

The methods described so far are very useful for structuring design information. In addition, they could be considered useful as assistants to designers’ thinking. The information elicitation is straightforward, since people are used to dealing with the argumentation terminology. On the other hand, the potential of information retrieval and the artifact explanation are questionable. The lack of consistency checking between the decision network and the fact that arguments are represented as noninterpreted text jeopardizes the usefulness of the method. In addition, the documentation load is increased for designers, which make us believe that these documentation aids will not be useful.

To decrease the documentation load and to make the argumentation method more useful, JANUS (Fischer et al., 1991) (Fischer and McCall, 1989) integrates the PHI argumentation model to a domain-specific design critic tool and to a domain-specific environment. The result is an environment in which designers design and document at the same time. The use of domain knowledge and the integration with the design environment enriches the PHI implemented method in JANUS, although the rationale is still a non-interpreted text.

Another interpretation for rationale would be information fully able to explain the design decisions. This meaning is more powerful than the argumentative models proposed above. In this research branch, exemplified in work by Gruber and Russell (1990) and Baudin et al. (1990), a deep model containing form, function and behavior of information about domain concepts supports the acquisition of rationale in a specific domain.

Although their approach is very attractive due to its explanatory power, it requires a strong and complete knowledge base. Designers do not always use first-principles knowledge during the design process, instead they primarily use heuristic knowledge accumulated from their experience. We believe that documentation users will get the greatest benefit if knowledge is acquired at the same level of abstraction, using designers to explain their design. Our observations have shown that this explanatory level of abstraction is neither as deep as to need a first-principled model underlining it, nor as superficial as the heuristic level in which the designers perform the design. Therefore, a design rationale model should be dependent upon the way designers in a specific domain communicate their projects.

At last, the action-based rationale claims that actions can be explained by themselves, exemplified by Lakin’s electronic notebook project (Lakin et al., 1989). In this model, documentation should contain all information and actions that took place during the design. It requires an integrated design environment that would allow designers to access all information they need and to use the tools they need. The environment should record the entire designer’s log, i.e. all actions the user takes during a design session—all the design information to be re-produced later. This approach alleviates the problem of interfering in the design process, but creates a new problem in managing the large volume of information.

The three major approaches discussed above can be summarized considering two axes of discussion: the complexity of the knowledge elicitation and the explanatory power of the acquired information (Figure 3). In the action-based approach, information is acquired very easily, since it is a sub-product of using the tool. However, the amount of information...
and the lack of organization makes the explanatory power very low. In the argumentation-based approach, information is also captured easily, but the lack of consistency checking and understanding of rationale jeopardizes the explanatory power of what was recorded. The most powerful explanation skills should be offered by the model-based approach, since it has a model of the artifact. However, the explanatory power of the model-based approach is still questionable, particularly due to the lack of a user model for providing explanation. As far as the knowledge acquisition and the quality of the product are concerned, the model-based approach offers greater complexity. It requires knowledge elicitation at the first-principles level.

ADD proposes to use a model of the domain, but not necessarily using first-principle knowledge. It contains knowledge at the level of abstraction that designers use in developing their designs. As in JANUS, ADD integrates design and documentation, containing a design critique tool for assisting the design. However, it represents rationale as an interpreted and reusable piece of knowledge. Instead of being apart from the design, rationale is the basis for the design.

In comparison to the argumentation-based methods, ADD not only connects the decisions (issues) connected to each other, but also connects the decisions to the artifact data structure. Constraints, evidence, and design context and observations that compose the argumentation of a decision. Sub goals and dependency relations are explicitly represented. Our research goal is to enrich decision arguments by making their components the same as the design itself. This aspect makes us introduce more domain knowledge. The elicitation problem of the model-based approach is solved by using a domain model in the level of abstraction as the level used by designers and by documentation users.

4. ADD's design rationale model

One of the goals of our research is to develop a computational model for design rationale. Before presenting our model of design rationale, we introduce here our understanding of design and rationale.

From the point of view of documentation, a design is the result of a set of chained decisions responsible for incrementally instantiating the necessary attributes to define a device. To understand the decisions that compose the design, the proposed model for design documentation should be able to reproduce those decisions—this is the key aspect of the proposed research. Each decision represents a change in the design state; i.e., a design attribute or object was instantiated, altered or destroyed. However, the rationale for the design is not just a set of specific decisions, but includes the relationships between those decisions and the domain context within which those decisions are made. Therefore, the design rationale model has three components:

1. The decision model consists of elements that describe a specific decision within the design, including decision alternatives, evaluation criteria, constraints, topics, fixes, impacts, previous cases, goal, design context, and design agents.

2. The decision network model describes the relationships (sequencing, composition, and dependency) between the decisions that comprise the design.

3. The domain model defines the design strategies, heuristic design rules, code constraints, device models, and environment models.

The first two models are explored further in the following subsections. The third model is outlined in the next section on the HVAC system design problem.

4.1 DESIGN MODEL

As defined in previous sections, the HVAC system design task can be characterized as a search for parameter values guided by a set of evaluation criteria and subject to a set of constraints. The characteristics of the task suggest some type of constraint-satisfaction (Sussman et al., 1979) approach for doing the design. The design model we are using is similar to the one used in VT's model for elevator's design (Marcus, 1989). Figure 4 presents the representation we are using for design. This representation emphasizes the microlevel of the decision-making. Decision alternatives, evaluation criteria, constraints, topics, fixes, impacts, previous cases, goal, design context, and design agents are the nodes represented in this network. The set of relations in this representation includes generates, constraint, evaluates, and selects. The process starts with a design goal to be achieved within a specific decision context and considering a set of design agents. The context represents the starting state for the decision process (the parameters with their values and the decisions made). The design agents (architect, code, standards, mechanical engineer, etc.) represent the sources for the constraints and criteria that should be considered in the design. A design goal generates design topics (parameters) to be specifically instantiated. Topics might require the instantiation of other topics or other goals. A topic might trigger the selection of a case to support an alternative. Constraints are generated by the design context, the goals, the agents, and retrieved cases. Constraint violations generate fixes that, in turn, generate impacts on the design, which result in new constraints. Topics generate alternatives that are constrained by the constraints. Agents generate the evaluation criteria that evaluate each alternative. A decision is the act of selecting one or more alternatives.

4.2 DECISION NETWORK MODEL

In general, a design is composed of multiple decisions and those decisions are interrelated. As a starting point, the ADD model defines three types of relationships between decisions: dependency, sequencing, and composition (see Figure 5).

Decisions are linked sequentially to indicate their order in the design process. In the figure, the design sequence is represented by the directed path (D4, D2, D2, D6, D2, D2, D2, D2, D2). Certain decisions involve the decomposition of the design space (e.g., HVAC distribution systems are decomposed into heating, cooling, and ventilation subsystems). In the figure, decision D1 represents a decomposition into two subsidiary decisions D1 and D2, and the decisions D1 and D1 are combined in decision D2. Finally, decisions may be interdependent because they are linked through bidirectional constraints (e.g., the ductwork size depends on the required number of air changes per hour and vice-versa). In the figure, decisions D1, D2, and D3 are interdependent.

5. HVAC system design problem

HVAC system designers are working to control environment conditions such as temperature, humidity and air cleanliness in buildings, cars, spacecraft, etc. We are focusing on HVAC systems in buildings. The constraints, criteria, and goals of that design problem are given in Table 1.

Although the definition of the HVAC system depends on the definition of its subsystems, they are not equally important. Most of the time, the cooling subsystem leads the entire design due to its higher cost. The choice of the subsystems is not independent. In addition to the constraints, criteria, and goals, a design domain model also includes design strategies that guide the decision-making process. A strategy defines a preference for a sequence of goals. The availability of explicit strategies allows the documentation user to follow the intent of the designer.

The HVAC system design is a typical design problem in which the specification is incomplete and sometimes ambiguous. In addition, the decisions with higher impact on the project are made during the conceptual and preliminary design. Typically, these decisions are not recorded. Unless complete documentation is an explicit design requirement, designers generally do not have time to produce the careful documentation that would allow the above questions to be fully answered. One reason for this time pressure is the low ratio between the number of projects accepted and the number of projects bid, particularly affecting the documentation of preliminary design phase.

6. ADD model for acquiring and retrieving design rationale

This section presents the ADD model for documenting design decisions. ADD simulates the behavior of
TABLE 1. Elements of the HVAC system design problem

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Criteria</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building dimensions</td>
<td>Least cost</td>
<td>Generation systems for</td>
</tr>
<tr>
<td>Building shape</td>
<td>Best system performance</td>
<td>cooling</td>
</tr>
<tr>
<td>Building orientation (sun impact)</td>
<td>Least maintenance</td>
<td>heating</td>
</tr>
<tr>
<td>Glass information in each exposure</td>
<td>Most centralized</td>
<td>ventilation</td>
</tr>
<tr>
<td>Number of floors</td>
<td>Most independent</td>
<td></td>
</tr>
<tr>
<td>Building major function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Building available mechanical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equipment area equipment location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Building shaft space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Building shaft location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floor to floor height</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ceiling space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position of main structural elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size of main structural elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Building location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desired indoor temperature</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desired inside humidity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desired inside cleanliness</td>
<td></td>
<td></td>
</tr>
<tr>
<td>First cost limitations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating cost limitations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tenant cost limitations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Owner’s cost limitations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Durability limitations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintenance limitations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Owner’s preference and need of back-up system</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

an apprentice who is observing the designer, asking questions whenever it does not understand what the designer did. By understanding, we mean being able to reproduce the decisions involved in the design. Based on this apprentice metaphor, ADD creates an expectation for each of the designer’s decisions. Whenever its expectation is frustrated, ADD will request the user to justify the decision, allowing ADD to adjust its expectation.

The design documentation model, illustrated in Figure 6, consists of a user interface for developing a design case; a controller that contains domain knowledge for checking designers’ decisions and captures all the design decisions, decisions’ argumentation and artifact values discussed during design process; and an interface for retrieving the information related to each recorded design case. ADD works by documenting the complete design decision path associated with the artifact, as well as the rationale behind each decision presented by the user. This solution path represents the designer’s strategy in which each node is a sequentially linked decision.

Figure 6. ADD’s apprentice model

ADD’s knowledge acquisition model is based on five assumptions:

- Rationale should be captured in the context of use when a design is being made. The knowledge involved in the design process may be completely captured only if the acquisition is made during that process. Generally, a person is not able to recall every bit of information about the value of a given attribute of the design, for example, the size of a ventilation duct. Each decision is made up of two elements: a focus (what to decide) and an action (a value for a parameter).

7. ADD architecture

ADD’s architecture for supporting design documentation is presented in Figure 8. ADD consists of a documentation capture component and a documentation retrieval component that share information about the design scenarios of various cases. The two components are described in the following sections.

7.1 DOCUMENTATION CAPTURE

The documentations capture component consists of (see Figure 8):

- the interfaces responsible for developing the design and acquiring new pieces of knowledge (the design and justification interfaces, respectively)
- the design scenario, which contains knowledge about the domain and knowledge about the specific case
- the reasoning components responsible for generating design decisions, comparing those decisions with the designer’s decisions, and querying the user for additional knowledge as needed (the anticipator, reconciler, and knowledge elicitor, respectively).

7.1.1 User interfaces for documentation capture

Interviews with designers in the domain area of HVAC system have been the basic data for developing ADD’s interface. We videotaped designers from three different companies developing projects.
We also analysed the documentation users. Our goal was to define rationale as answers to design questions made by the documentation users. In addition to identifying the heuristics used by designers, we were concerned with the interface needs. We concluded that a menu-driven interface is sufficient for the engineers' needs, given current technology. We also observed that they used drawings (floor plans) as the reference point for designing. Based on these preliminary observations, we have developed an initial Design User Interface and the Justification User Interface.

The Design User Interface is menu-driven. This feature lets the user understand how the system is doing its reasoning and the system to identifying what type of decision is being taken. The menus correspond to the system's strategy and component description for designing a device, as illustrated in screen display from the current prototype shown Figures 9 and 10. Since we are considering design as an opportunistic activity, we do not want to impose a specific order of actions. The design criteria window allows the user to select the evaluation criteria that should be considered in the design. The design parameter window allows the user to define constraints over parameters. The two windows on the right side of the display (Figure 10) show the typical floor plan and floor elevation.

The Justification User Interface supplements the Design User Interface by allowing the system to output its decision path and the user to input rationale for contradictory decisions. The Justification User Interface will start working as soon as the Reconciler detects a conflict between the designer's and the Anticipator's decisions. The Justification User Interface will provide the user with the following features:

- The user will be able to look at the entire description of the concepts involved in the system's explanations.
- The user will be able to modify those concepts by updating the knowledge the system contains about the case.
- The user will be able to introduce additional rules and design constraints. There are two possibilities: either the system contains the necessary concepts for forming a new constraint or the system needs first to acquire the necessary primitive concepts. In the first case, the system conducts the acquisition by proposing protocols to be filled in by the user. In the latter case, the system first needs to connect the new concepts with some previously known concepts.

7.1.2 Reasoning for documentation capture
The documentation process is controlled by the Anticipator, Reconciler and Knowledge Elicitor. The role of the Anticipator is to create an expectation for a decision topic given by the designer considering the current state of the design and the active requirements. To make a prediction, the Anticipator uses domain knowledge base and information about the specific design case being developed. It is a constraint-based reasoner; i.e. given some constraints and a set of evaluation criteria, it is able to generate and analyse alternatives and propose a set of solutions.

The Reconciler compares the solution proposed by the user and the system. If they are similar, then the system assumes that its own reasons are correct. If not, the system presents its own solution for that scenario followed by its rationale and asks the user for reasons for the difference. If the system, with these new pieces of knowledge, can reproduce the user's action, then it will attach that knowledge to the case. The determination of similarity between the decisions is intrinsically connected with the problem of 'when' to ask the user for more information. This issue is very important since we do not want to overload the user with the documentation. On the other hand, the system should not jeopardize the validity of the explanation generated for the designer's actions by not asking questions.

When ADD identifies a single solution from a set of alternatives, the Reconciler needs to be able to deal with the three possible cases (see Figure 11):

(A) The user proposes the same decision as the one proposed by ADD. In addition, all of ADD's other alternatives are ranked much lower. In this case, ADD considers its expectation to be correct.

(B) The user proposes a solution that was lower ranked by ADD. Therefore, ADD needs to
Design knowledge through design decision justification

understand the reasons that led the user to select that alternative to ADD's selection.

The user proposes a decision that was not recognized by ADD. ADD needs to acquire knowledge to be able to generate and evaluate the new alternative.

When ADD identifies a set of closely ranked alternatives for a decision considering a certain evaluation criterion, cases B and C are still applicable, and case A is no longer valid. In addition, we identify one new case (see Figure 12);

(D) The user selects a good alternative (one of ADD's highly ranked alternatives). The Reconciler needs to re-examine the differences between the user's choice and the other alternatives to see if a clear basis for a preference can be found within ADD's knowledge base (e.g. a particular constraint that is much better satisfied in the user's choice than any of the other alternatives).

If no clear basis can be found, then the Reconciler must query the user for more information.

The Knowledge Elicitor defines the knowledge elicitation strategy. It is the module behind the Justification Interface. We plan to have a guided knowledge elicitation. The user will be able to browse the system's chain of reasoning and detect where it fails. Then the user can define new pieces of knowledge using the concepts already known.

7.1.3 The design scenario

The design scenario contains the information needed by the system to generate explainable expectations. It contains a parametric model of the domain, design heuristic rules, a model for the artifact, a model for the environment (the context in which the artifact fits in), a database of components and a model for design rationale. A specific project is an instantiation of this design case, and it is later recorded in the library of design cases. The key model elements are shown in Figure 13.

The parametric model contains information about all parameters in the design process. A parameter can be a derived parameter or a primitive parameter. They represent information about the form, function, and behavior of artifacts and environments. The parameters are the elements that are to be designed (instantiated). Figure 14 is partial snapshot of ADD's parametric model.

The environment model contains information about the context in which the artifact is being designed. The artifact model contains all parts and concepts necessary to define the artifact. In addition to the artifact model, ADD contains an artifact component database with information about the available components and the ways that they can be assembled. The design decisions are connected to the artifact model, allowing the retrieval of information from the artifact to support the design.

The design decision model is responsible for predicting which decisions can be made at this moment. It constrains the strategy for the decision-making. In the current prototype, ADD will use this knowledge to augment the rationale for explanation purposes, but will not use it to help predict the next design step. Instead, the user will choose the decision topic for ADD's reasoning.

The design knowledge is represented by heuristic rules. Those rules reflect the way designers develop

7.2 DOCUMENTATION RETRIEVAL

Once the knowledge is captured, the next step is to allow the user to retrieve it. We have made a preliminary analysis of the needs of the documentation users to guide the design of the explanation interface. The interface for retrieving cases consists of a smart browser. This read-only interface will allow the user to navigate graphically through the decisions (and alternatives) connected to the device structure and verify the knowledge required for justifying each decision. The sequence of decisions will be presented to the user allowing him or her to verify all the decisions at a specific level of abstraction (both detailed input to specific decisions and overall strategies that link decisions).

8. Conclusion

This paper describes ADD, an integrated computational model for assisting designers in documenting projects at design time. In addition to capturing the device parameters, this documentation model captures the decisions and the rationale for those decisions. Our model was developed based on observations of designers developing HVAC systems and on observations of documentation users accessing design documents.

By integrating both design information and rationale, we can provide the end user with a new kind of product: an intelligent design document. This document has many potential uses. It can support project activities, including design verification, redesign, error reduction and optimization in downstream tasks, and construction. Across the facility life cycle, the information can be used to maintain and upgrade the HVAC system. The designers can also use the new documentation as a robust repository of experience for application to future projects.

We have implemented first versions of the Anticipator, the Design Scenario and the Design User Interface. We are working to increase the HVAC knowledge base, code the Reconciler, and test the system. We have developed some metrics for measuring the potential of the Anticipator and to define the points where the Anticipator should stop in the design process and ask the user for more information.

A. C. B. Garcia and H. C. Howard

Acknowledgements

We would like to thank Mark Sterlik for his comments on the ideas presented in this paper. We also want to thank Tom Gruber for his insightful comments in early versions of this paper and the referees for their comments which helped us to make our ideas clear.

This work has been financed by the Stanford Center for Integrated Facility Engineering, the National Science Foundation (Grant MSM-9558316) and a fellowship to the first author from CNPq-Brazil.

References


McCall, R. 1987. PHIRE: procedurally hierarchical issue-based
Design knowledge through design decision justification

Ana Cristina Bicharra Garcia is a PhD candidate in the Civil Engineering Department at Stanford University. Her research interests are primarily in computer applications that facilitate engineering design and documentation. She is a member of the Brazilian Society for Civil Engineering and AAAI. She received her BS in structural engineering in 1983 from the Federal University of Rio de Janeiro, Brazil, and her MS degree in Computer-Aided Civil Engineering from Stanford University in 1989. She is currently working on her PhD thesis in design documentation.

H. Craig Howard is an Assistant Professor of Civil Engineering at Stanford University. He focuses his research and teaching in the area of Civil Engineering computer applications, with particular interests in integrated systems, knowledge-based systems, engineering databases, and educational software. He is actively involved in Stanford's Center for Integrated Facility Engineering. He is also a member of the IEEE Computer Society, ACM AAAI, Sigma Xi, Tau Beta Pi, and the American Society of Civil Engineers. He received his BS in Civil Engineering from Virginia Polytechnic Institute and State University in 1981, and his MS and PhD degrees in Civil Engineering from Carnegie-Mellon University in 1983 and 1986 respectively. He joined Stanford University in 1986.