Case-Based Design in the SEED System(1)

Ulrich Flemming(2), Robert Coyne(3), James Snyder(4)

Table of Contents

Abstract. *We present the requirements for case-based design in a software environment to support the early phases in building design and outline an approach to satisfy them. We briefly discuss implementation issues and the larger information management context for the work.*

1 Introduction

The SEED project intends to develop a software environment that supports the early phases in building design (Flemming et al., 1993). The goal is to provide support, in principle, for the preliminary design of buildings in all aspects that can gain from computer support. This includes using the computer not only for analysis and evaluation, but also more actively for the generation of designs, or more accurately, for the *rapid generation of design representations*. A major motivation for the development of SEED is to bring the results of two multi-generational research efforts focusing on "generative" design systems closer to practice: (1) LOOS/ABLOOS, a generative system for the synthesis of layouts of rectangles (Flemming et al., 1988; Flemming, 1989; Coyne and Flemming, 1990; Coyne, 1991); and (2) GENESIS, a rule-based system that supports the generation of assemblies of 3-dimensional solids (Heisserman, 1991; Heisserman and Woodbury, 1993).

The rapid generation of design representations can take advantage of special opportunities when it deals with a *recurring building type*, that is, a building type dealt with frequently by the users of the system. Design firms - from housing manufacturers to government agencies - accumulate considerable experience with recurring building types. But current CAD systems capture this experience and support its reuse only marginally. SEED intends to provide systematic support for the storing and retrieval of past solutions and their adaptation to similar problem situations. This motivation aligns aspects of SEED closely with current work in Artificial Intelligence that focuses on *case-based design* (see, for example, Kolodner, 1991; Domeshek and Kolodner, 1992; Hua et al., 1992).

The present paper gives a brief overview of SEED in the following section, followed by an outline of the requirements for case-based design in SEED in section 3. Section 4 introduces our approach toward indexing and retrieval and section 5 case adaptation. Section 6 discusses implementation issues that arise from this approach, and section 7 places case-based design in the larger context of information management in design.

2 Overview of SEED

We have set SEED up as a collection of *modules*. This allows us to make local use of various pieces of existing and possibly heterogeneous software and to distribute the development efforts among several teams and over time. On the other hand, each module should appear to the user as part of a unified whole. We attempt to achieve these multiple objectives by basing each module on a common logic, which allows us also to develop a common style for the interfaces.

In order to arrive at this common logic, we divide the overall design process into distinct *tasks* or *phases*, where a phase is defined by the type of *problem* it addresses and the type of *solution* it generates. A *module* supports work in an entire phase. A common architecture and interface can then be developed for the modules based on this uniform problem-solving view (Flemming et al., 1993).

Each SEED module offers designers a broad range of form generation capabilities that range from stepwise construction under the designer's control to the automatic generation of design alternatives and case-based design. In order to enable the module to participate actively and effectively in each of these modes, an explicit statement of the problem to be solved in an invocation of a module is needed. In addition to a generation and evaluation component, each SEED module therefore contains also a *problem specification component* that allows designers to specify and modify dynamically the design problem to be solved. The following three modules are planned for the first SEED prototype based on this generic architecture and currently under development.

Module 1: *Architectural Programming*	**Problem**: A project description as it is typically given at the start of a building project: context (including the site); the overall function and size of the building (e.g. an elementary school for 300 pupils); budget etc. **Solution**: An architectural program or design brief.

Module 2: *Schematic Layout Design*	**Problem**: A specification of functional components, a con text, budget, and other parts of an architectural program **Solution**: A layout of the functional units that determines the overall building organization, zones (including the distribu tion of units on different floors), and form subject to the specifications of the program

Module 3: *Schematic Configuration Design*	**Problem**: A schematic layout and programmatic require ments **Solution**: A three-dimensional configuration of spatial and physical building components that realizes the given organi zation subject to the programmatic requirements

3 Requirements for Case-Based Design in SEED

The reuse of solutions is ubiquitous in building design and extends through all phases, from programming to detail design. A designer should therefore be able to call upon and reuse past solutions to problems similar to a current one in any one of the planned SEED modules, and the retrieval and adaptation of such solutions are capabilities to be offered *across modules.*

Furthermore, the problem addressed in a module may be decomposed into a hierarchy of (sub)problems. For example, the layout of a building may be decomposed into the layouts of individual floors, of zones on a floor, of rooms within a zone, and of equipment or furniture in a room. Standard solutions may exist for problems at any level in such a hierarchy. As a result, retrieval capabilities for past solutions should be available in SEED also *across hierarchical problem decomposition levels within a module*. An added requirement is that a solution to any subproblem in a problem hierarchy should also be retrievable by itself.

This reuse capability within SEED aims at aiding designers in two crucial ways. First, it provides access to a large memory of past solutions that is likely to contain instances designers will not recall on their own, either because there are too many of them or because they were generated by different designers. This motivation clearly differs from the motivation behind the system described in (Hua et al., 1992). A second aim is to provide designers fast with an initial solution that is immediately available for editing and modifications under either the system's or the

designer's control. This motivation differentiates SEED, for example, clearly from Archie-II (Domeshek and Kolodner, 1992).

The amount of work required to prepare a case for storage, retrieval, and adaptation becomes a crucial issue for a case memory of the size envisioned for SEED. The system is intended for use by architectural and other design firms, who cannot be expected to command the resources needed to support substantial preparation efforts. Case-based design in SEED is therefore starts from the premise that *its case memory accumulates as a side-effect of a firm's normal design activities*. Any indexing and other types of processing that need to be done occur in the background and are hidden from designers. This requirement distinguishes SEED from many approaches that must rely on considerable efforts to build a memory of cases (Rosenman et al., 1991; Domeshek and Kolodner, 1992; Hua et al., 1992).

We believe that these requirements, taken together, reflect a realistic application context and that our approach is therefore of practical interest.

4 Indexing and Retrieval

We attempt to keep the problem specifications in the diverse modules as uniform as possible in order to emphasize the common logic underlying the modules. This uniformity also gives us the chance to develop a uniform indexing and retrieval mechanism to support case-based design across modules and problem decompositions: if a solution has been developed with the aid of a module, the associated problem specification can be used, upon the designer's request, to compute an index to be stored together with the solution as a case so that at a later time, when the same or a different designer defines a similar problem, the resulting problem specification can be compared with the stored indices to retrieve solutions to similar problems.

This uniform problem specification relies on two central constructs, design units and functional units. *Design units* are the basic spatial or physical entities that make up the representation of a design and are the primary focus of attention during form generation, which concentrates in specifying their shape, location and non-geometric attributes. A design unit is generally multi-functional, that is, has to accomplish more than one purpose. A *functional unit* collects all of the requirements that a design unit has to satisfy in a single construct.

A functional unit defines a design problem, namely that of shaping, placing and otherwise determining the properties of a design unit that satisfies the respective requirements in a given design context. A functional unit can contain *constituent functional units* (see the floor plan example used in the preceding section). The hierarchical decomposition of a functional unit into constituent units leads to a parallel decomposition of the resulting design problem into subproblems through several abstraction levels (see also the general AEC reference model GARM (Gielingh, 1988)).

There is no assumption that this specification is complete at any time in the design process. It is restricted formally to requirements that can be evaluated by the module and taken into account during automated form generation and case retrieval. It is restricted in its content to requirements that appear currently important to the designer. In general, design in SEED proceeds in an "open world" (Hinrichs, 1992).

A generic matching and retrieval mechanism with the required properties becomes conceivable, in principle, for the first SEED prototype because it will be implemented based on an object-oriented approach in which all major entities handled by a module are objects with attributes that can be compared with the attributes of other objects. We use the term object here in a conceptual, generic sense (independent of a specific programming language or implementation environment) to denote an entity characterized by attributes with values. We call an object *structured* if some of its attributes are lists of objects of the same type. An example are the constituents of a functional unit, which define a functional unit and problem hierarchy as described above.

The problem specification that exists in a module is a structured object in this sense. Comparisons of a current problem specification, which we call a *target index*, with a case index can then proceed, in principle, object-by-object and attribute-by-attribute (see Flemming, 1994 for details).

The degree to which a module can actively support the refinement or adaptation of a retrieved case crucially depends on the degree to which the structure of the target index and that of the retrieved case index match because this indicates also how closely the retrieved solution solves the current problem. We are particularly interested in cases whose index *refines* the target index; that is, there exists a right-unique mapping of the constituents in the target index into the constituents of the case index that associates strictly constituents in the same or in a compatible class and transitively preserves exactly the hierarchical dependencies between the constituents of the target index. As a concrete example, a problem specification of module 2 may specify a radiology department as a functional unit only in terms of its overall size, but a retrieved solution may contain the individual rooms in such a department as its constituents, which correspond to additional functional units in the attached index and are hierarchically subordinate to the radiology department. *Refinement may be a desired side effect of the retrieval of a case* because it saves time not only in terms of finding an initial solution, but also in terms of time spent on refining the problem specification itself.

As a consequence, the comparisons made during memory search should not discriminate against a case when its index refines the current problem structure; on the contrary, they may favor it for that very reason. That is, a case with a compatible index may be preferred over another case if it has more structure, everything else being equal. The latter restriction is important because a designer dealing with a one-bedroom apartment, for example, may not be interested in a 3-bedroom apartment, let alone an apartment building. Problem attributes like size restrictions, budget, etc. will prevent cases with such over-refinements from being retrieved.

5 Case Adaptation

When the designer asks a module to find good candidate solutions in the case base to solve the problem currently specified, the module tries to find solutions that allocate at least all the functional units currently specified. But in the case of refinement, such solutions contain design units associated with functional units that are not yet specified. After retrieval, the new design units become part of the current solution and the associated functional units (which are preserved in the index) become part of the current problem specification, where they are available for inspection and editing. That is, case adaptation may include adaptation of the problem specification.

A SEED module supports the interactive editing and expansion of problem specifications. The underlying operations are available after a case has been retrieved to incorporate refinements in the retrieved problem structure into the current one automatically or selectively in interaction with the designer. Commands are also available to expand or modify solutions according to a problem specification, and the underlying operations are again available to adapt a retrieved case to the current problem specification interactively or automatically.

6 Implementation Issues

Three of the main requirements have an impact on the case based implementation. First, the case base is expected to grow over time via designer interaction. Second, the case base is to be used by multiple modules and across module sessions. Third, the case base is to support the rapid generation of design representations. The first two requirements imply that the case base should be stored in a persistent format and have a common module access interface. The third requirement implies that the representation of cases should be closely aligned with the general SEED information representations so that cases can be easily moved to and from the case base.

The implementation of a case-based system must be supported by a knowledge representation system, which can be divided into a reasoning system and a storage system. The *reasoning system* performs the indexing, matching, and adapting of information within the case base, as well as the transfer of information to and from the SEED environment. We anticipate employing a reasoning mechanism that includes constraint propagation similar to the system described in (Hinrichs 1992). The *storage system* manages the persistency and representation of information, which comprises collections of attribute-value pairs aggregated as objects. In the literature, reasoning systems have been described in detail, but more effort is needed in the representation, storage, and manipulation of information within the case base. The storage system, along with the knowledge representation, must support the reasoning system in the indexing and adaptation processes. For case retrieval, the storage system must support at a minimum subclass queries, structure matching, as well as attribute and value matching. Subclass queries are

critical in matching specialized cases to general queries. For example, a target index may specify a *public-circulation-space*, which would be matched by a *corridor* because it is a specialization of *public-circulation-space*. The structure matching occurs via the constituent relationship between functional units, which is separate from the *IS-A* relationships defined in a taxonomy.

Given the above requirements, a natural strategy for implementation is the use of an object database system. The current SEED implementation relies on an object database system in other areas, which thus becomes a natural implementation mechanism. The main implementation issues to be resolved are whether to use the notion of an object provided by the database or to develop a separate object representation utilizing the database objects. Several benefits are gained using the first approach including the availability of database tools to help manage the case information. However, the indexing and adaptation process is probably too rigidly constrained with this approach. We therefore plan to implement a representational system for storing cases. A significant benefit of this approach is that it allows us to experiment with different indexing techniques without reconstructing the existing case base; the indexing system can be implemented in terms of database objects as well.

What has not yet been established is the method for embellishing the case objects with the appropriate hints for the reasoning system to index cases. We believe that the designer should not be required to intervene in this process, and we plan to investigate several indexing alternatives.

7 The Larger Context

We believe that case-based design, as envisioned here, should be viewed in the larger context of information management in team-based design. Empirical studies reveal a number of problems and bottlenecks in everyday work in such domains as bridge construction (Grønbæk, 1993) and in a variety of engineering design and manufacturing contexts (Levy et al, 1993). Problems have been identified in archiving and retrieving information, in interrelating information from multiple and diverse sources and media, and in searching for and finding information. They typically occur because there is a discrepancy between large, monolithic, vertical systems (applications) and daily work. This leads to costly overhead in information management and breakdowns in communication and understanding unless the related problems of usability, information structuring and management are addressed.

Since usefulness and usability are primary goals of the SEED project - *bringing research design methods and technology closer to practice* - these observations are relevant to SEED or any of its modules in general and apply particularly to the creation, elaboration, adaptation and reuse of cases. For instance, a designer may ask: "What was the case Fred was working on yesterday that he said would be a good starting place?", or "What were the cases that I considered six months ago when I was faced with a similar design problem, and how did I select from the alternative matches and adapt the one chosen?" These examples suggest that the content of cases should be enlarged, e.g. with annotations, that retrieval should also be possible based on clues that are less explicit and structured than the problem specifications in a module, or that cases be linked to project information outside of SEED itself. It is also true that a CAD system like SEED should not and pragmatically cannot encompass all general information management dimensions because (1) tools exist in a continuum of tools and activities, and it would be counter-productive to extend any one tool to manage these issues for all the others; and (2) it is not possible to foresee all types of information and tools that should be cross-referenced in a project: they are dynamically dependent on project-specific issues and phases, on the participants, tools, work process, design office context, and so on.

The studies cited above suggest a *uniform information management environment* that links heterogeneous information types and allows tools (like SEED) to be used and tracked within the environment. Since tools which are open and provide "hooks" for external manipulation are easy to embed in such an environment, these features should be taken into account in the system design from the start. Aside from a variety of approaches and technologies known under loose categories such as CSCW, hypermedia and concurrent engineering that may bear on these issues, we are aware of a few research efforts that specifically address information modeling and management for design: the SHARE (Toye et al., 1993) and *n*-dim projects (Levy et al., 1993; Subrahmanian et al., 1993). We plan at the present time to explore a relationship with *n*-dim because (1) it appears to be more comprehensive with respect to capturing design rationales and histories as side effects of information structuring, and (2) it is a local effort with some overlap of personnel involved. *n*-dim encompasses a methodology and

environment to support generalized information modeling (including the cross-referencing of diverse types of information such as models of the organization, process and artifact), the embedding of tools, multiple indexing of the same information, and design histories.

Acknowledgments. The development of SEED is sponsored by Battelle Pacific Northwest Laboratories, the US Army Corps of Engineers Construction Engineering Research Laboratories (USACERL), the National Institute of Standards and Technology (NIST), the Engineering Design Research Center (EDRC) at Carnegie Mellon University, the Australian Research Council and the University of Adelaide.

References

Coyne, R. F. (1991) ABLOOS. An Evolving Hierarchical Design Framework, Ph.D. diss, Dept. of Architecture, Carnegie Mellon University, Pittsburgh, PA

Coyne, R. F. and U. Flemming (1990) "Planning in Design Synthesis: Abstraction-Based LOOS" in *Artificial Intelligence in Engineering V, Vol. 1: Design* (Proc. Fifth Int. Conf., Boston, Mass.), J. Gero (ed.), New York: Springer, 91-111

Domeshek, E. A. and J. L. Kolodner (1992) "A Case-Based Design Aid for Architecture" in *Artificial Intelligence in Design `92*, J. Gero (ed.), Boston: Kluwer Academic Publishers, 497-516

Flemming, U. (1989) "More on the Representation and Generation of Loosely Packed Arrangements of Rectangles" *Environment and Planning B. Planning and Design* **16**:327-359

Flemming, U. (1994) "Case-Based Design in the SEED System" *Automation in Construction*, forthcoming

Flemming, U., R. Coyne, T. Glavin, and M. Rychener (1988) "A Generative Expert System for the Design of Building Layouts - Version 2" in *Artificial Intelligence in Engineering: Design* (Proc. Third International Conference, Palo Alto, Ca), J. Gero (ed.), New York: Elsevier, 445-464

Flemming, U., R. Coyne, R. Woodbury (1993) "SEED: A Software Environment to Support the Early Phases in Building Design" in *ARECDAO93* (Proc. IV Int. Conf. on Computer Aided Design in Architecture and Civil Engineering, Barcelona, Spain) 111-122

Gantt. M. and B. A. Nardi (1992) "Gardeners and Gurus: Patterns of Cooperation Among CAD Users" in *CHI `92 Conference Proceedings: Striking a Balance* (P. Bauersfeld, J. Bennett, G. Lynch, eds.), Reading, MA: Addison-Wesley

W. Gielingh (1988) General AEC Reference Model. ISO TC 184/SC4/WG1 doc 3.2.2.1, TNO Report BI-88-150

Grønbæk, K., M. Kyng, P. Mogensen (1993) "CSCW Challenges: Cooperative design in engineering projects" *Comm ACM* **36**:67-77

Heisserman, J. (1991) Generative Geometric Design and Boundary Solid Grammars, Ph.D. diss., Dept. of Architecture, Carnegie Mellon University, Pittsburgh, PA

Heisserman, J. and R. Woodbury (1993) "Generating Languages of Solid Models" in *Proc. Second ACM/ IEEE Conf. on Solid Modeling and Applications*, Montreal, Canada, May 19-21

Hinrichs, T. R. (1992) *Problem Solving in Open Worlds. A Case Study in Design*, Hillsdale, NJ: Erlbaum Associates

Hua, K., I. Smith, B. Faltings, S. Shih, G. Schmitt (1992) "Adaptation of Spatial Design Cases" in *Artificial Intelligence in Design `92*, J. Gero (ed.), Boston: Kluwer Academic Publ., 559-575

Kolodner, J. L. (1991) "Improving Human Decision Making through Case-Based Decision Aiding" *AI Magazine* **12**(2):52-68

Levy, S., E. Subrahmanian, S. Konda, R. Coyne, A. Westerberg, Y. Reich (1993) "An Overview of the n-dim Environment" Engineering Design Research Center Tech. Report EDRC-05-65093, Carnegie Mellon University, Pittsburgh, PA

Rosenman, A., J. S. Gero, R. E. Oxman (1991) "What's in a Case: The Use of Case Bases, Knowledge Bases, and Databases in Design" in *CAAD Futures `91* (Proc. Int. Conf. on Computer-Aided Architectural Design Futures, Zürich Switzerland), G. N. Schmitt (ed.), Wiesbaden, Germany: Vieweg, 285-300.

Subrahmanian, S., R. F. Coyne, S. Konda, S. Levy, R. Martin, I. Monarch, Y. Reich, A. Westerberg (1993) "Support System for Different-Time Different Place Collaboration for Concurrent Engineering" in *Proc. Second Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, IEEE Computer Society Press, Los Alamitos, CA, 187-191.

Toye, G., M. R. Cutkosky, L. J. Leifer, J. M. Tenenbaum, J. Glicksman (1993) "SHARE: A Methodology and Environment for Collaborative Product Development" in *Proc. Second Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Los Alamitos, CA: IEEE Computer Society Press, 33-47.

Footnotes

(1)

(2)
Professor, Department of Architecture and Engineering Design Research Center (EDRC), Carnegie Mellon University, Pittsburgh, PA 15213. The Engineering Design Research Center is an NSF-supported Engineering Research Center.

(3)
Research Associate, Department of Architecture and Engineering Design Research Center (EDRC), Carnegie Mellon University, Pittsburgh, PA 15213.

(4)
Research Assistant, Department of Architecture and Engineering Design Research Center (EDRC), Carnegie Mellon University, Pittsburgh, PA 15213.