Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2000), *14*, 415–430. Printed in the USA.
Copyright © 2000 Cambridge University Press 0890-0604/00 $12.50

SEED-Config: A case-based reasoning system for conceptual building design

HUGUES RIVARD[1] AND STEVEN J. FENVES[2]

[1]Centre for Building Studies, Department of Building, Civil, and Environmental Engineering, Concordia University, Montreal, QC, H3G 1M8, Canada
[2]Department of Civil and Environmental Engineering, Carnegie Mellon University, Pittsburgh, PA, 15213, U.S.A.

(RECEIVED February 25, 1999; ACCEPTED March 30, 2000)

Abstract

A case-based design functionality is a natural and intuitive addition to a design tool that can augment human capabilities and help designers remember and retrieve appropriate cases. SEED-Config, a design environment for conceptual building design, was developed to incorporate a case-based reasoning functionality to provide designers with initial potential solutions. The case representation in SEED-Config is the BENT information model, which records design knowledge, supports the hierarchical decomposition of design cases, offers multiple views, and encapsulates the outcome of the design in addition to the problem specification and the design solution. The case library was implemented in an object-oriented database management system to accumulate cases automatically and to provide efficient query facilities. The case retrieval aspect of SEED-Config offers three different methods to find the most useful cases stored in the case library: task-based, lineage-based, and customized. Case retrieval responds to the exploratory nature of the design process and supports versatile case retrieval by providing multiple paths to each case. The case adaptation aspect, which adjusts the selected case to the new problem to provide a complete solution, uses an adaptation method called derivational replay. The case-based design capabilities are completely integrated within the design environment from which the cases originate.

Keywords: Building Design; Case-Based Reasoning; Derivational Replay; Design Environment; Object-Oriented Database

1. INTRODUCTION

Experienced designers rely extensively on past experiences to solve new design problems. The problems they face remind them of similar prior experiences, or cases, that they have encountered before. Such previous cases are a source of inspiration on the basis of which more relevant solutions to the problems at hand can be developed (Fenves et al., 1995). By using concepts that have been proven effective in past situations, designers save both time and effort. Actually, experienced designers (or experts) can be recognized by the ease with which they recall just the right prior experiences to help them solve the current problem (Schank, 1982).

A case-based reasoning (CBR) system is a tool that can store previous experiences in a case library, find experiences that are similar to a given problem, and adapt the most appropriate one to solve that problem. Because designers may forget or remember the wrong cases, a CBR system could help them perform better reasoning by expanding their memories with cases that they may have forgotten or that originated from other designers (Kolodner, 1991). A collection of prior designs would thus be made available to them. A CBR system can provide support for storing and using the collective memory of a design firm. Such a system could become highly specialized, and even idiosyncratic, as it gathers design cases. In the long run, it would reflect a firm's design practice and style. This type of design tool is expandable, as new cases can be added, and it is modifiable through its adaptation mechanism, so that it can support changes in practice (Fenves, 1995). Hence, a CBR system could record a firm's expertise, facilitate knowledge exchange between

Reprint requests to: Hugues Rivard, Department of Building, Civil and Environmental Engineering, Concordia University, 1455 de Maisonneuve Blvd. West, Montreal, Quebec H3G 1M8, Canada. E-mail: rivard@alcor.concordia.ca

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

designers, and allow novice designers to tap into a large source of design experience.

This article presents a CBR system developed for conceptual building design (Rivard, 1997). The research project described here is part of SEED-Config. SEED (Software Environment to support the Early phases of building Design) is a multidisciplinary project that intends to provide computer support for the preliminary design of buildings (Flemming & Woodbury, 1995). The emphasis is on supporting early design exploration, that is, the fast generation of alternative design concepts and their rapid evaluation against a broad spectrum of criteria. SEED-Config is the module within SEED that supports the design of three-dimensional building models in terms of spaces, subsystems, and physical elements (Woodbury & Chang, 1995). It is intended to support decisions on the overall form of the building and the structural, enclosure, mechanical, electrical, and interior systems. The goal is to implement a tool that supports human designers by augmenting their creativity and expertise (Fenves, 1995).

CBR systems have four major aspects: case representation, case accumulation, case retrieval, and case adaptation. **Case representation** addresses the issues of how the information is represented, recorded, and accessed. **Case accumulation** provides the support for learning; it is responsible for storing solutions to new problems as cases in the case library. The next two aspects form the problem-solving process of CBR systems: **Case retrieval** is responsible for finding the most useful cases stored in the case library that can address the current problem, and **case adaptation** transforms the selected case into a solution for the problem at hand.

This article describes the four major aspects of the CBR functionality of SEED-Config in the above order. This is preceded by a list of requirements and by a review of a selection of CBR systems found in the literature.

2. REQUIREMENTS FOR BUILDING DESIGN

A list of seven requirements was formulated to guide the development of the CBR functionality of SEED-Config. This list defines the features required in a CBR system for the elaboration of a user-friendly conceptual building design environment. Some of these requirements originated from SEED's specifications, whereas others originated from observations of the building design environment.

2.1. Problem decomposition

Buildings are rarely similar enough to justify the construction of the same design twice. Although the complete design of a building cannot be reused as a whole, parts of the design such as concepts, details, styles, elements, and subsystems may be reused. Complex design problems are typically broken down by designers into sets of subproblems, which can, in turn, be further divided until smaller and more manageable tasks are obtained. This "divide-and-conquer" strategy results in a hierarchical decomposition of the design problem. An example of a hierarchical decomposition for a building project is shown in Figure 1. The retrieval of past solutions must be supported at any level of such hierarchical problem decompositions (Flemming & Woodbury, 1995). This requires that the case representation provide easy access to parts of a case and that each part be indexed independently from the global case by the indexing scheme (Kolodner, 1993).

2.2. Integration of multiple views

Each designer involved in the building design process (e.g., architect, mechanical engineer, interior designer, etc.) has a different view of the final product. Hence, a CBR system for building design should integrate the multiple views of a building. This requirement ensures the retrieval of cases with multiple functions. For instance, a case for an exterior wall could have a structural function (i.e., load-bearing capabilities) as well as an enclosure function (i.e., protection of the indoor environment from outdoor weather conditions). Even though SEED-Config currently addresses only three disciplines (i.e., massing, structural, and enclosure design), the intent is to append additional disciplines in the future.

2.3. Capture of evaluation

The outcome of a case represents the results of implementing the solution in the actual world by reporting the performance of the design and the degree to which it fulfilled the goals set forth in the problem statement. The outcome informs the designer whether a case should be selected. Cur-

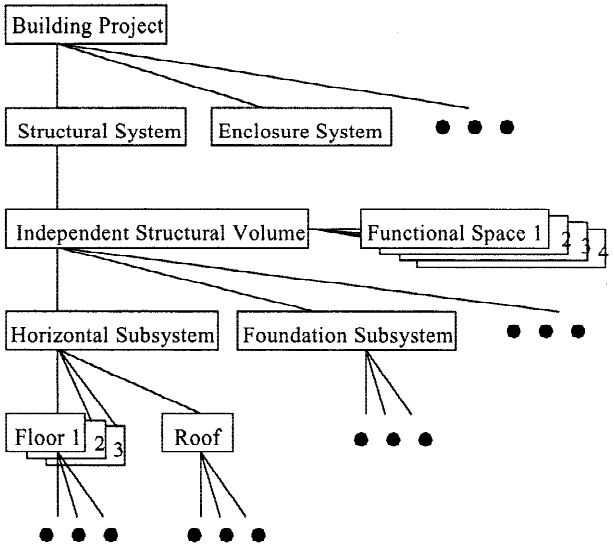


Fig. 1. Example of a hierarchical decomposition of a building.

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

rently, lessons learned on a construction project have limited or nonexistent feedback to subsequent projects (Sanvido et al., 1992). There is a need for communicating concerns and consequences from the construction or implementation phase directly to those making design decisions. Cases would provide a logical place to record such concerns and consequences, because those outcomes would automatically be made available to the designers when they intend to use a retrieved case, thus becoming part of the collective memory of the design firm.

2.4. Automatic case accumulation

The main purpose of SEED-Config is to aid designers in designing. Any overhead beyond the normal design effort for indexing or processing before storing a case should remain hidden from the designers as much as possible, so that cases are accumulated as a side effect of normal design activities (Flemming & Woodbury, 1995). The idea is to eliminate any effort that may make designers reluctant to add potentially useful cases to the case library.

2.5. Design exploration

The building design process is exploratory in nature. Initially, the problem specification is incomplete and the relevant features of the design problem manifest themselves only as the design proceeds. Then, the domain of design possibilities is explored by synthesizing, evaluating, and comparing several design alternatives until a satisfactory solution emerges. Hence, a CBR system should support the fact that only incomplete problem specifications are available initially, that specifications may evolve dynamically, and that several design alternatives may need to be considered (Maher et al., 1995).

2.6. Versatile case retrieval

People can remember useful cases quickly because their long term memory resembles a large library where the information is stored by subject, is generously cross-referenced, and has a sophisticated index that gives direct access through multiple entries to the subjects (Simon, 1996). Similarly, a CBR system should be equipped with a versatile case retrieval mechanism that provides several ways of accessing and retrieving cases. For instance, a query could entail a problem statement, a design solution, a design evaluation, a design process used, or a combination of these.

2.7. Integrated design environment

CBR systems for design can be categorized as either stand-alone systems, which uniquely rely on the CBR methodology to design, or as integrated design environments, which integrate CBR with other design generation and editing capabilities. Integrated design environments are preferred for two reasons. First, designers can adapt retrieved cases using the form generation and editing capabilities of the same design environment used to create the initial designs, and second, adding new cases in stand-alone systems requires significant effort (Flemming, 1994; Fenves, 1995). CBR systems are also either automatic or interactive (Kolodner, 1993). Automatic CBR systems are capable of synthesizing a solution by themselves. Interactive systems require input from designers throughout the process. The latter are more appropriate for design because design problems are usually large and complex, value or aesthetic judgments are necessary, considerable knowledge is used, the knowledge involved is difficult to define, and designers may be legally responsible for the solutions proposed.

3. REVIEW OF CASE-BASED REASONING SYSTEMS

Several CBR systems have been developed for civil engineering and architecture. A selection of these systems is presented below in a chronological order. This selection is not intended to be exhaustive, but rather as a sampling of systems that were found to be relevant to this research. Each of these systems is evaluated in terms of the SEED-Config requirements. For additional information, readers are referred to a detailed review of CBR systems for building design that compares different systems based on the issues of case representation, retrieval, presentation, and adaptation (Watson & Perera, 1997).

STRUPLE (STRUctural PLanning from Experience) is an early attempt to implement a CBR system in the domain of building design (Zhao & Maher, 1988). It is a stand-alone system that proposes structural subsystems (at the top level of abstraction only) from past experiences based on a problem description. The goal is to reduce the design space to a subset of structural subsystems and then to pass it on to a synthesis program such as HI-RISE (Maher, 1984).

DDIS (Design-Dependent and -Independent System) is an integrated design environment for designing beam-columns. It integrates case-based and rule-based knowledge, called design-dependent and design-independent knowledge, using a blackboard architecture (Wang & Howard, 1991). Design plans are sequences of design steps, each of which is modeled as a rule in the knowledge base. A design problem is solved by trying to reenact a selected design plan on the current situation. DDIS supports problem decomposition, and new design cases are automatically processed, indexed, and added to the case library. DDIS does not, however, support design exploration explicitly, and the user is responsible for selecting the most relevant design from memory because no retrieval mechanism is provided.

ARCHIE is an interactive system for conceptual architectural design (Pearce et al., 1992). It provides sug-

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

gestions and warnings during design synthesis and evaluation. It can retrieve cases similar to the current problem based on a problem description and a solution description, both of which can change and evolve. Archie is not integrated with a design environment but it lets architects explore previous cases for ideas and inspirations. The library of design cases was obtained from the literature and entered in the system before it could be used. New cases cannot be automatically added.

CADSYN integrates design cases and generalized knowledge in a design synthesis environment that assists designers during the conceptual structural design stage (Maher & Zhang, 1993). CADSYN relies on a decomposition hierarchy, which contains generalized knowledge about the design of building structures. Design methods attached to each node of the hierarchy determine the attribute values. The decomposition hierarchy can be used to generate a new design or to adapt an inconsistent solution. CADSYN adds new cases automatically but does not specifically support design exploration.

CASECAD is a stand-alone system that assists structural designers during conceptual design (Maher & Balachandran, 1994). Existing building structures are stored as cases with symbolic data (used to index the case) and graphics (used to visualize the case). CASECAD's advantages are that it can decompose problems into hierarchies, record the outcome, and manage multimedia data (Maher et al., 1995). It also provides a flexible indexing system with multiple access paths to cases. Its drawbacks are that adding new cases requires significant effort, design exploration is not explicitly supported, and adaptation is currently not implemented.

CADRE is a CBR system in which a case is a geometric model of both the structural system and the architectural layout of a building (Bailey & Smith, 1994; Bharat et al., 1995). CADRE provides two transformation adaptation methods. Dimensional adaptation involves finding a solution for a set of linear and nonlinear constraints on the parameters used to describe the building, whereas topological adaptation modifies the topology so that a new dimensional model with its corresponding constraints can be generated. The constraints required for adaptation are either obtained interactively from the user or generated automatically from the geometrical model. CADRE addresses the multidisciplinary nature of building design, but does not provide case retrieval, recording of outcomes, and support for problem decomposition or design exploration.

CASETOOL is an extension to an expert design system, called DEKBASE (for Development Environment for Knowledge-Based Systems in Engineering) (Krishnamoorthy et al., 1993). Expert systems are used to establish the search conditions, to set weights for ranking, to adapt cases, to verify compatibility between elements, to critique designs, and to control the CBR process (Kumar & Krishnamoorthy, 1995). CASETOOL is integrated with a design environment, decomposes problems, accumulates cases automatically, and records case evaluations, but does not support design exploration.

CBDEC (Case-Based Design of Envelope Components) is a CBR system integrated in a knowledge-based environment to assist in the design of the building enclosure (Gowri et al., 1995). It is an extension of an earlier implementation of the building enclosure design process in an expert system called BEADS (Gowri, 1990). A breadth-first search is used to retrieve a set of cases that are then filtered using a pattern-recognition technique to evaluate the similarity of their physical signature to the current situation. CBDEC does not record outcomes, but supports problem decomposition, is integrated with a design environment, and indexes cases automatically before they are added to the case library.

CADREM is a CBR system that proposes structural layouts for buildings (Kumar & Raphael, 1997). CADREM uses a case-retrieval approach called retrieval based on examples (RBEX) (Raphael & Kumar, 1996). The retrieval mechanism tries to match the main task specified in the problem specification with similar tasks specified in design methods within cases. Adapting past solutions to new problems is done by reusing retrieved design methods. CADREM is an integrated design environment that decomposes problems into hierarchies of subproblems; however it does not specifically support design exploration and does not record design evaluation.

None of the CBR systems presented here satisfy all the requirements for SEED-Config. A summary of the evaluation of these systems is shown in Table 1. The table does not include STRUPLE because it is only a front end to an expert system. Most systems support problem decompositions (except for ARCHIE and CADRE), accumulate new cases automatically (except for ARCHIE, CADRE, and CASECAD), and are integrated design environment (except for ARCHIE and CASECAD). There seems to be a consensus that hierarchical problem decomposition is necessary in breaking down a complex system into simpler problems to allow searches to be done at the system level, as well as at the level of its parts. CADRE is the only system that accounts for the multidisciplinary aspects of design. Only three systems, ARCHIE, CASECAD, and CASETOOL, record design evaluation. Only ARCHIE supports design exploration as specified in the SEED-Config requirements and only CASECAD provides versatile case retrieval mechanisms.

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

Table 1. *Assessment of some CBR systems with respect to SEED-Config requirements*

	Case-based reasoning systems							
SEED-Config requirements	DDIS	ARCHIE	CADSYN	CASECAD	CADRE	CASETOOL	CBDEC	CADREM
Problem decomposition	**Yes**	No	**Yes**	**Yes**	No	**Yes**	**Yes**	**Yes**
Multiple view integration	No	No	No	No	**Yes**	No	No	No
Capture of evaluation	No	**Yes**	No	**Yes**	No	**Yes**	No	No
Automatic case accumulation	**Yes**	No	**Yes**	No	No	**Yes**	**Yes**	**Yes**
Design exploration	No	**Yes**	No	No	No	No	No	No
Versatile case retrieval	No	No	No	**Yes**	No	No	No	No
Integrated design environment	**Yes**	No	**Yes**	No	**Yes**	**Yes**	**Yes**	**Yes**

4. CASE REPRESENTATION

The first of the four major aspects of CBR systems is the representation of cases. A case describes a past problem-solving situation. Case representation organizes the information that defines a problem along with its solution, its outcome, and, optionally, the manner in which the problem was solved. This aspect of a CBR system delimits the capabilities of the system and influences other aspects such as indexes, organizational structures, retrieval algorithms, and adaptation mechanisms.

Cases and designs in SEED-Config are represented according to a generic information model called the Building Entity and Technology (BENT) model (Rivard & Fenves, 2000). The information model consists of a set of basic constructs, built on top of an object-oriented model, enabling the organization of data about design cases in a logical and structured manner. The goals of this information model are to store design data as they are generated, to share data among design participants, and to support CBR.

In the BENT model, the building is represented as an assembly of entities. A building entity is something that can be distinctly identified in a building design, and about which data are accumulated (Eastman & Fereshetian, 1994). A building entity represents a concept meaningful to designers and can be a system, a subsystem, a part, a feature of a part, a space or a joint (Gielingh, 1988). The BENT model represents each building entity as a generic container that encompasses its geometry, taxonomy, properties, relationships, and the design knowledge that generated it. Each of these aspects is clearly delineated in the model so that efficient and independent tools can be used to manipulate them. Figure 2 shows an idealized representation of a building entity called "First Floor," along with portions of its data. Each box is an instance of a class identified in bold letters in the upper section of the box. The constructs of the building entity are described below. For more information, please refer to Rivard & Fenves (2000).

4.1. Geometric description

A building entity is geometrically described by a unique primary spatial representation, which is the entity's high-level description used principally for reasoning about its geometry and its topological relationships with other entities, and by a few secondary geometric representations, which are domain specific. This representation scheme relies on the premise that topological relationships of physical entities are invariant with respect to their domain-specific secondary representations (Zamanian & Fenves, 1994). The primary spatial representation of the entity "First Floor," shown in Figure 2, is a plane that defines its extent as well as its location in space. An example of a secondary geometric representation for this entity is a steel reinforcement bar layout.

4.2. Taxonomy

A classifier instance is a label assigned to classify a building entity. It also serves as an index for querying the case library and retrieving cases. Classifier instances provide access to building entities independently from the design sequence that generated them. A given building entity typically has several classifier instances assigned to it, each of which offers a different means to retrieve that entity. The building entity "First Floor," shown in Figure 2, has been assigned a classifier instance, which specifies the structural material to be used in its construction as "Cast-in-place Concrete."

4.3. Properties

An attribute-value pair is a named property of a building entity. In the BENT model, the attribute-value pairs are organized in two hierarchical levels. At the highest level, data are grouped into three subsets. The functional unit (FU) includes properties representing intended purposes, requirements, and constraints on the entity; it can be seen as a design-problem statement (Gielingh, 1988). The design unit (DU) includes properties representing all the material and shape characteristics that define the actual design of the entity; it can be seen as a solution alternative to the design problem. The evaluation unit (EU) represents the behavior of the entity and includes the response to stimulation associated with different design conditions. It can also be used to record the concerns and consequences of a design that has been implemented, thus providing support for the re-

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

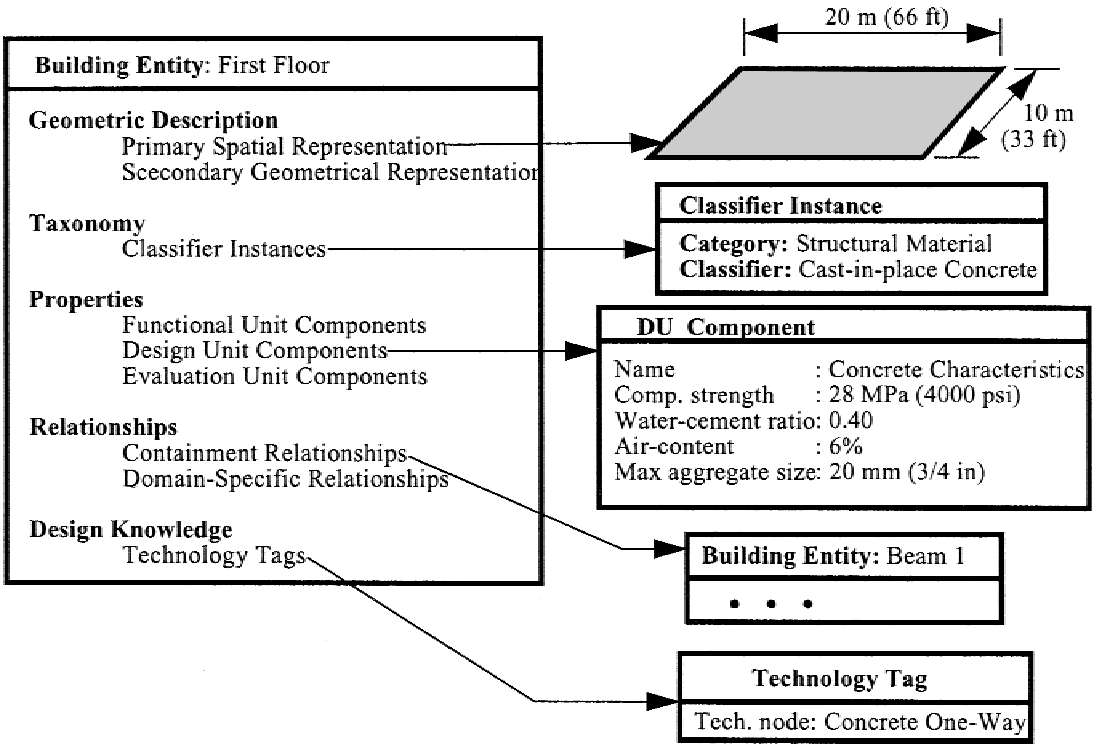


Fig. 2. An example of a building entity and a portion of its data.

quirement of capture of evaluation specified in Section 2. At the lower level of data aggregation, the attribute-value pairs are combined into small cohesive subsets, each of which is called a component (Phan & Howard, 1993), so that closely related attributes may be found together. For instance, a design unit component for structural engineering called "Concrete Characteristics" (see Fig. 2) may contain the following attribute-value pairs: compressive strength, water–cement ratio, air content, and maximum aggregate size. Components allow building entities to be refined in staged steps by adding sets of attribute-value pairs to the entity as they are generated in the design process. In this manner, there is no need to define every attribute that may be needed in a building entity at the outset. Furthermore, the complexity of the building entity can be abstracted to the user with partial views displaying only the relevant components. The sharing of components among different building design domains satisfies the requirement of multiple view integration.

4.4. Relationships

Relationships among building entities fall into two categories: containment relationships (also known as aggregation relationships), which capture the links between an entity and its parts; and domain-specific relationships (DSRs), which describe other relationships of interest. A composite object is modeled as a building entity linked by containment relationships to other building entities that represent the parts. The containment relationship is used to model the hierarchical decomposition of design problems as specified in the problem decomposition requirement of Section 2. DSRs store other essential relationships such as "Support" and "Connect" in structural engineering. Spatial relationships, such as "above," "adjacent," "next to," and "spatially contained in," are not stored explicitly in this information model because they can be obtained directly from the geometric modeler. The building entity "First Floor" of Figure 2 is decomposed into a set of subentities comprising a slab and beams linked together through containment relationships (only one beam subentity is depicted in Figure 2). A wall entity located below could be linked to the "First Floor" entity by a DSR "Support" to define a load path (not shown in Figure 2).

4.5. Design knowledge

A technology tag is a reference to a particular reasoning step or operator, called technology node, that was used in designing the building entity. Technology nodes encapsulate design knowledge. Each node represents a known design alternative, the constraints that determine its applicability, and the computational steps necessary to assign values to the attribute(s) defining that alternative. Technology nodes are arranged into a technology graph representing all possible design alternatives (Rivard et al., 1996; Gomez, 1998). A designer may select a technology node and apply it to the

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

current building entity when it is applicable in the current context. Technology nodes refine the design of a building entity by adding design unit components, classifier instances, and domain-specific relationships to the entity, or by creating new entities contained in the current building entity. Figure 2 shows a technology tag indicating that the "First Floor" entity has been designed with the technology node "Concrete-One-Way," which, among others, determines the supporting elements and defines the direction of the design spans. The technology concept is further elaborated in Gomez (1998), in terms of the development of a representation for technology nodes, the elaboration of its functionalities, and the creation of a user interface that supports the generation of technology nodes interactively. Of the systems surveyed in Section 3, DDIS, CADSYN, CASE-TOOL, and CADREM all have knowledge representations similar to the one used in SEED-Config.

5. CASE ACCUMULATION

The second major aspect of CBR systems is case accumulation. CBR systems have the ability to learn by accumulating new solutions as cases in the case library. They are thus able to solve problems to the extent of coverage of their knowledge, but as their knowledge grows through the addition of cases, the range of problems they can solve expands (Kolodner, 1993). This section describes the case library and the archival process implemented in SEED-Config.

5.1. The case library

A case library is the memory of the CBR system. It must provide powerful and efficient mechanisms for searching and accessing cases. The case library is implemented in an object-oriented database management system (OODBMS) to take advantage of the object-oriented schema constructs, persistence, and query capabilities it offers. OODBMSs are a fusion between the semantic richness of object-oriented languages and the practical features of relational database management systems. The OODBMS UniSQL was selected by the SEED team because it best satisfies the requirements specified for the database system, fully supports relational database constructs, has multi-database capabilities, and is compliant with SQL, the standard relational language supported by most database systems on the market (Snyder et al., 1995). UniSQL is actually an extended relational database management system, because it provides object-oriented concepts on top of the relational data model. It offers a superset of SQL, called SQL/X, that has additional features for dealing with complex objects, values, and methods. It also comes with a C++ interface that provides access to the database. The BENT model is implemented as a database schema in UniSQL by defining every class in the model in SQL/X. This database schema defines the case library. A translation mechanism is necessary because of important differences between the implementations of the BENT model in UniSQL and in C++ (for the SEED-Config design environment).

5.2. Archiving new cases

A building entity by itself is a self-contained case because it includes its problem statement (the functional unit), the design solution (the design unit), the design process (the technology tags that provide a reference to the design knowledge used), and the outcome of the design (the evaluation unit). Furthermore, a case is already indexed as the technology nodes that were applied to the entity during the design process have assigned classifier instances to it.

To add a new case, the designer selects a building entity in the hierarchical decomposition and instructs SEED-Config to archive it in the case library. These are the only two actions required, thus satisfying the main requirement pertaining to case accumulation, established in Section 2, that stipulates minimal effort on the part of the designer. The designer can select the building entity at any level of the hierarchical decomposition. To archive the whole project, the designer selects the root of the project (i.e., the entity located at the top of the hierarchical decomposition). The application then copies the selected building entity and all its constituents as a case into the case library.

The archiving algorithm consists of a main function that recovers the initial data and manages the database, and a recursive function that copies building entities. The main archiving function is initiated with the building entity selected and performs the following actions. First, the functional unit component "Project Description" is obtained from the root of the project. This component is attached to the case and all its subentities. Its purpose is to describe the project for which the case was developed at retrieval time and it contains information such as the name of the project, the name of the owner, the address of the project, and the date the project was begun. Next, the database is opened and initialized. The "Project Description" is then saved in the database, and the recursive function "SaveBE" is called to archive the selected building entity and all its subentities. Finally, the transaction is committed and the database is closed.

The purpose of the function "SaveBE" is to copy data from the C++ implementation of the BENT model into the UniSQL database schema. First, the function saves the primary spatial representation of the building entity into a summarized representation containing its dimensionality (i.e., point, line, plane, or volume) and extent. Next, the function copies the technology paths (i.e., sequence of technology tags), thus storing the design knowledge used in refining the building entity. It also stores the classifier instances (which act as indexes), the secondary geometrical representations, and all the functional unit, design unit, and evaluation unit components. Finally, the function stores DSRs. Only the DSRs that relate subentities of the case root are stored in the case library. Figure 3 illustrates which DSRs from a

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

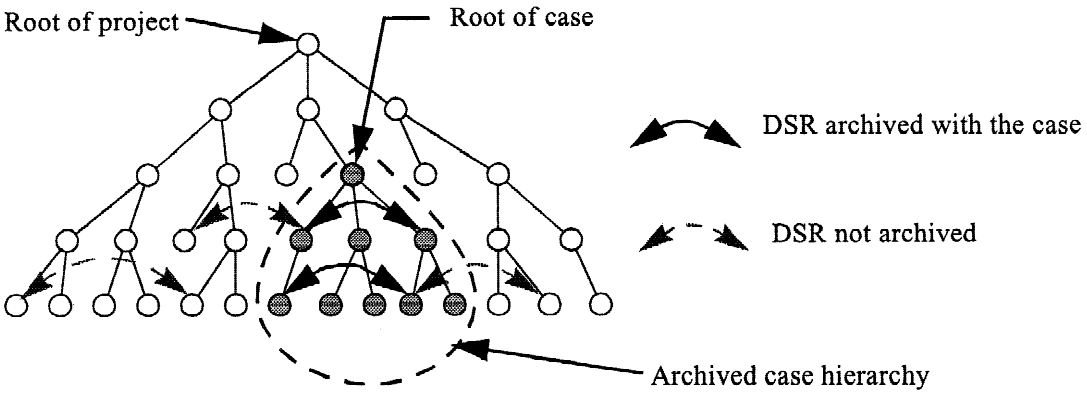


Fig. 3. Archiving a case with domain-specific relationships from within the hierarchical decomposition of a design problem.

project are stored in the case library. The DSRs with entities lying outside the case must be reestablished during the adaptation process. Finally, the function is called recursively for all the subentities of the building entity.

A case is considered incomplete until its outcome is recorded. In the building industry, the outcome can only be assessed several months, or even several years, after the design has been finalized, because it relies exclusively on feedback from people involved in its construction, use, maintenance, and renovation. SEED-Config allows the storage of incomplete cases and provides facilities to add their outcomes once they become available.

Most CBR systems described in the literature need to be seeded before they can be used. The seeding of the case library involves the insertion of an initial set of cases that forms the necessary foundation for solving the most frequent or important problems (Kolodner, 1993). This foundation is then extended as new problems are encountered, resolved, and added to the case library. SEED-Config does not require an initial set of cases because it is a self-sufficient design environment. Of course, the CBR capability of SEED-Config cannot be used at first because of its empty case library, but as designs are generated using SEED-Config's design capabilities and added as cases in its case library, the CBR system becomes more and more powerful.

6. CASE RETRIEVAL

The third major aspect of CBR systems is case retrieval. Case retrieval involves the formulation of a query based on the current design problem statement and the search of the case library. The result of the retrieval is a set of the most useful cases that can potentially help the designer in solving the current problem. This section describes the implementation of case retrieval in SEED-Config. There are four subsections: the first focuses on the overall retrieval process; the second describes the available options for preparing queries; the third discusses the use of semantic networks in generalizing and specializing queries; and the final subsection evaluates the approach implemented.

6.1. The overall retrieval process

The overall retrieval process implemented in SEED-Config is depicted in Figure 4. The control flow of the retrieval process is informal in the sense that the designer is in control of all steps. The opposite would be the formal approach that handles the retrieval automatically (Maher & Pu, 1997). The informal approach was preferred because it provides greater control and flexibility during the search definition. The steps in Figure 4 that require inputs from the designers are rendered as rectangles with dashed lines. Some automation and graphical user interfaces are provided to facilitate the designer's work.

The designer initiates the CBR process by first selecting a building entity (typically, the building entity currently being designed) and then activating the case retrieval process. Two search windows are automatically initialized based on the current problem statement of the selected building entity. These dialog windows are provided to help the designer in generating a query statement and are described in detail in the next subsection. Each allows the designer to switch from one to the other or to reset it to its initialized content. The minimum information required for case retrieval is the design problem statement, which is obtained from the functional unit of the building entity. The problem statement for a structural entity describes the type of the element (e.g., beam, column, slab, or frame), the loads, boundary conditions, constraints, and so on. The designer can modify the problem statement provided by adding or editing parameters in a search window. Once the designer has approved the specification in one of the search windows, a SQL/X query is automatically generated. The designer can edit the resulting query. Once the query is completed, the UniSQL database is opened and the query is processed. If no case is found, the retrieval process is restarted until a case is found or the process is cancelled. If the query finds a set of cases, they are presented in a dialog window. The designer is responsible for browsing through the cases and for selecting the most appropriate one to solve the problem at hand. If a case is selected, it is translated

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

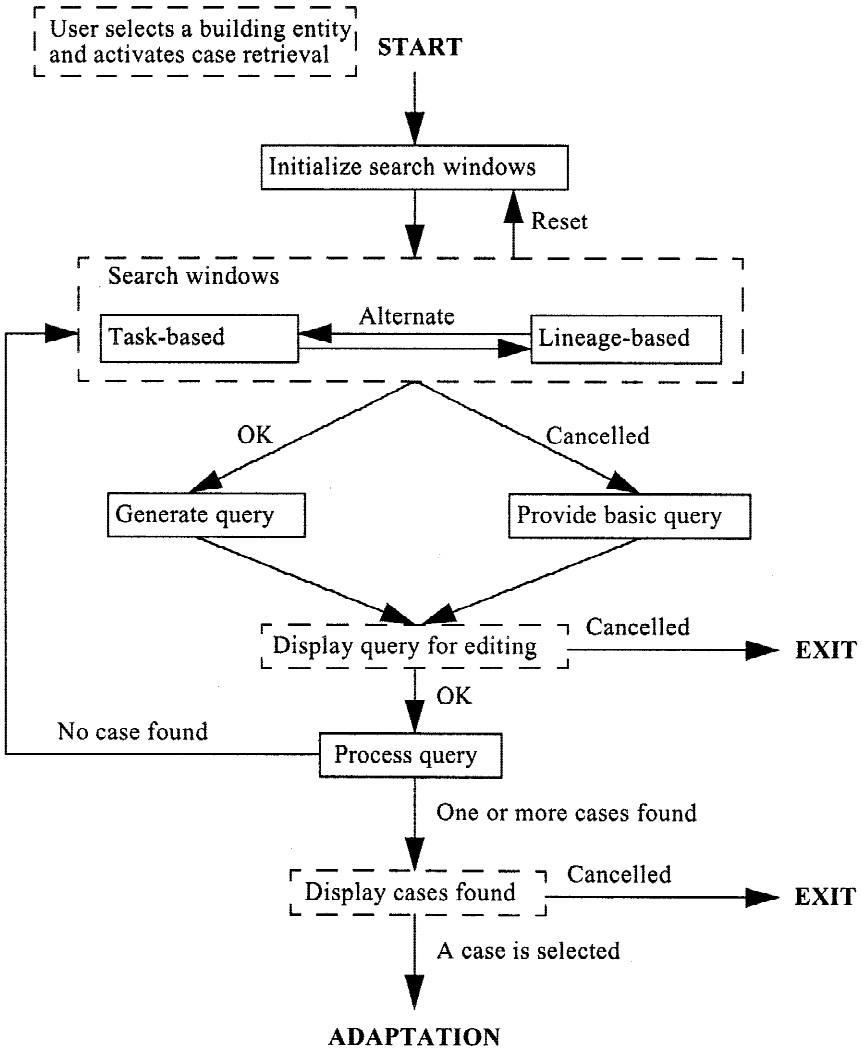


Fig. 4. Control flow for retrieving a case.

back into the C++ BENT model and transferred to the adaptation process described in the next section.

The control flow supports the typical trial and error aspect of situation assessment. Situation assessment is the analysis of the current problem statement to ensure the retrieval of useful cases that are similar to the problem at hand. Initially, the problem is usually not fully defined and inferences from the designer are required to complete the problem statement. If the result of the retrieval is inappropriate, the situation can be reassessed. Hence, situation assessment is an incremental process that provides increasingly more accurate problem statements as the understanding of the current situation is refined (Kolodner, 1993).

6.2. Case query options

The designer has three options for preparing a query: based on task, based on lineage, or customized. The first option focuses on the building entity selected, whereas the second one supports retrieval based on the hierarchical decomposition of the case. These two options provide dialog windows to assist the designer in automatically generating a query statement and are exclusive (i.e., only one or the other can be used in a retrieval). The third option, which lets the designer prepare the query directly in SQL/X, can be used in conjunction with one of the first two. Each of these options is described below.

6.2.1. Task-based search

A task-based search focuses on the specific building entity in the hierarchical decomposition that constitutes the designer's current task and does not involve related entities. This is the search window that appears by default.

The search parameters of the task-based search window, shown in Figure 5, are grouped into three sections: primary spatial representation, classification, and component. The

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

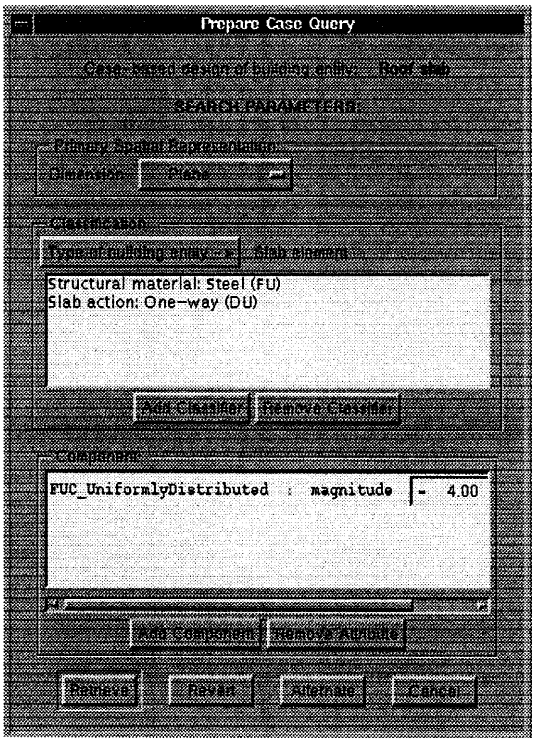


Fig. 5. Task-based search dialog window.

first section defines the dimensionality of the case to be retrieved and can take one of five values: point, line, plane, volume, or no restriction. The second section defines the set of indexes that classifies the case. The type of building entity is displayed first, and the view underneath displays the list of classifier instances associated with the building entity. Each classifier instance is shown with its category first, followed by its classifier name, and its unit type. The third section specifies a list of attribute-value pairs. Each attribute is shown with the name of its component, followed by the attribute name, and a field that is used to specify its value. This field can be used by the designer to set ranges of acceptable values.

The functionality of this window is explained through an example. A structural engineer is designing a roof. His current task is to design the slab. When he activates the retrieval process, the window is initialized based on the selected building entity and appears as shown in Figure 5. The dimensionality is obtained from the primary spatial representation of the entity and is set to plane. The type of building entity and the classifiers are set according to the classifier instances currently assigned to the building entity. This entity has been partially classified as depicted in the second section with the type of building entity "Slab element" and two classifier instances: "Structural material = Steel," and "Slab action = One-way." The third section shows that a functional unit component has been assigned to the entity defining a uniformly distributed load with magnitude of 4.0 kPa. Although only the functional unit components of the selected building entity are used to initialize the search window, the designer can add design and evaluation unit components.

The search parameters provided by default may either be expanded or restricted. Continuing the example, the structural engineer may expand the search by editing the field value of the load magnitude to be between 3.0 and 5.0 kPa instead of equal to 4.0 kPa. He may restrict the search by selecting the type of decking to be of corrugated steel by adding the classifier "Type of decking = Corrugated steel." The engineer may restrict the search further by adding an evaluation unit component for the construction cost with a limit of $70 per square meter or less. When the engineer presses the "Retrieve" button, the search specifications are automatically translated into a SQL/X query and presented in the Query Edit window. This window allows the designer to inspect and modify the SQL/X query generated before proceeding with the retrieval process as described earlier.

6.2.2. Lineage-based search

A lineage-based search takes into consideration related entities in the hierarchical decomposition of the case. This type of search relies on containment relationships, hence the term lineage for parent–child relationships. Example queries restricted by building entities that are related through containment relationships are: 1) find a roof design in a building that encloses an indoor swimming pool; 2) find a roof design made primarily of steel and that has open-web steel joists; and 3) find a roof design of a building with floors made primarily of steel. The first example is shown in Figure 6 with only the relevant portion of the hierarchical decomposition. The **target entity** is the building entity currently being designed; here the roof. The **related entity** is the building entity in the hierarchical decomposition used to limit the search; here the functional space with its occupancy category limited to swimming pool. In this example, the target entity is connected to the related entity through three containment relationships according to the hierarchical decomposition of Figure 1. The window for lineage-based search has two sections: the first section defines the target entity, which is initialized with the entity currently being designed, and the second section is used to specify the related entity.

When the designer has completed defining the query in the search window, a function automatically generates the corresponding SQL/X query. The function first computes the levels of the target and related entities in the hierarchical decomposition and determines which one is higher in the hierarchy. In the example, the higher one is the related entity "Functional space" and the lower one is the target entity "Roof." Next, the function finds the **root of the query**, which is the closest parent of both entities in the hierarchical decomposition (see Figure 6). The algorithm checks if the higher entity is the parent of the lower one. If it is, then it is also the root. If it is not, then the algorithm moves up to the next level until a common parent is found. In the case when

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

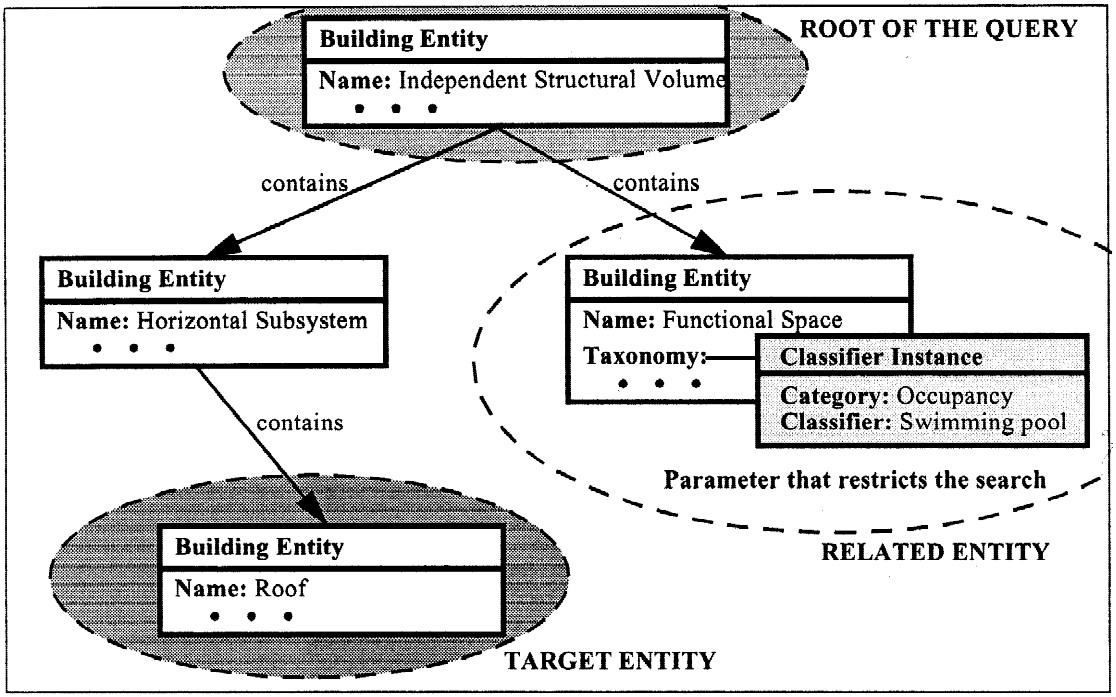


Fig. 6. A lineage-based search.

the two entities are at the same level, one of them is selected arbitrarily, and the algorithm moves up its lineage. Once the root of the query is determined and the numbers of levels between the root and both the target and the related entities are computed, a SQL/X query can be generated.

Success of the lineage-based search depends on the granularity of the cases stored. Because cases can be stored at different levels of the hierarchical decomposition, cases with roots at low levels of the hierarchy will not be accessible to lineage-based searches requiring a root located higher in the hierarchy, because archiving a portion of the building decomposition detaches the case from the context of the project.

6.2.3. Customized search

Customized search is used to define SQL/X queries that cannot be automatically generated by the two search windows provided. With this option, the designer can either write a specialized query or edit the queries automatically generated in the Query Edit window. Customized search can be used to create four types of specialized queries: queries on applied technologies, on geometrical extents, on domain-specific relationships, and on multifunctional cases. An example illustrating each one is given below:

- The technology nodes used in expanding an entity represent a good alternative means of selecting a set of cases because they record the design process as well as the knowledge involved. For instance, the designer can look for cases that have been designed with a specific technology node such as “Open web joist.”
- Because the case accumulation process records a summarized geometrical representation of the building entity, it is possible to create queries based on dimensionality and extent. For instance, the designer can look for the design of a concrete floor of 10 by 20 meters.
- A query can focus on the domain-specific relationship that exists between two entities. For instance, the designer can search for the design of a roof that is supported by a concrete wall. In this example, the query searches for a DSR of type “Support,” whose “supporting” role is filled by a “wall” entity classified with the classifier “Structural material = Concrete,” and whose “supported” role is filled by the “roof” entity, the target of the query.
- Because the case representation, described in Section 4, supports the retrieval of cases with multiple functions, the designer is able to retrieve an entity that complies with two or more functions. For instance, a structural designer can look for a case representing the design of an exterior wall that has both structural and enclosure functions and that has specified thermal properties.

6.3. Generalization and specialization of queries

The classifying and indexing approaches of SEED-Config provide a method for generalizing and specializing queries. SEED-Config uses the faceted classification scheme. In this scheme, a building entity has associated with it a set of categories (or facets), in which each category defines a spe-

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

cific aspect of the entity. A category consists of a group of classifiers arranged hierarchically. An example of a category in structural design is "Structural Material," which has four classifiers: "Steel," "Concrete," "Timber," and "Masonry." Each of these classifiers can be specialized further, for instance, "Concrete" can be specialized into "Cast-in-place Concrete" and "Precast Concrete." Hence, classifiers are arranged in a subsumption hierarchy. Subsumption is the process of recognizing a narrower concept (or classifier) to be part of a broader one. Classifiers are more specific as one travels down and more general or abstract as one travels up the hierarchy. A building entity is classified by selecting one classifier from each of its associated categories and by assigning the corresponding classifier instance to the entity. Each classifier instance, described in Section 4, corresponds to one classifier from the classification. A software module of SEED-Config, called the Classification Reference Manager, allows the designer to define, modify, manage, infer about, and query the classification that defines the domain of all legal categories and classifiers.

Classification hierarchies are in fact semantic networks, which provide an efficient mechanism for supporting reasoning about the generalization or specialization of concepts. This is an essential capability for widening the space of applicable cases when a query results in a limited number of matches. Continuing the example shown in Figure 6, the initial query is specified as "Occupancy = Swimming pool." The category "Occupancy" and its partial classification hierarchy is shown in Figure 7. If the query is unsuccessful because there are no such cases in the case library, the query can be generalized through the semantic network (i.e., the classification hierarchy), to sport facilities or even to halls, possibly finding the design of an auditorium that could be reused. Hence, a second, more general query can be specified as "Occupancy = Hall," which subsumes "Sport facility," "Entertainment," and their respective specializations. This capability applies to all three options available for defining a query.

6.4. Evaluation of the case retrieval approach

The case retrieval process described above satisfies the requirement for design exploration, stated in Section 2, by allowing the search specifications to change and expand as design progresses. The incomplete search specification used to initialize the search windows can be used to search for cases or can be expanded to get more specific searches. Hence, a retrieved case satisfies the requirements set forth in the initial search specifications as well as the specifications that emerge during the design process.

The requirement for versatile case retrieval is satisfied by providing multiple paths to each case, because the entire design is recorded in a case and can be queried. Case queries are not limited to design problem statements (functional units) only. The design solution, design process steps, outcome, problem decomposition, relationships, and aspects of the geometry can all be searched. Furthermore, classifier instances (i.e., indexes) are obtained from classifier hierarchies that allow generalization and specialization of queries. All of these means provide flexibility to the designer in searching the design knowledge stored in the case library. The goal is to ensure that cases are accessible every time they are needed.

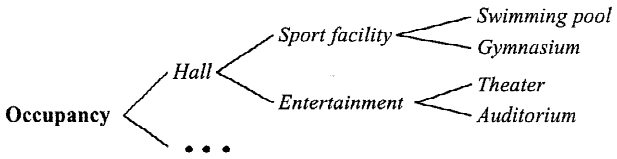


Fig. 7. The "Occupancy" category and its partial classification hierarchy.

The search windows described above represent a first attempt at providing the basic functionality for preparing case queries. They are proofs of concept showing that search windows could be developed to assist the designer in preparing task-based and lineage-based queries. This initial attempt could be improved in several ways: the two search windows could be merged into a more general search window; the SQL/X query could be kept hidden, thus concealing a level of complexity; and customized searches could be integrated into a general search window by supporting the specification of queries based on technology nodes, geometric extents, domain-specific relationships, and multifunctional cases.

Currently all cases that satisfy the query are retrieved and displayed in alphabetical order in a window. To provide better assistance to the designer, the cases could be evaluated and ranked to suggest which ones are considered best suited. Ranking is the process of ordering partially matching cases according to the quality of the match or the usefulness of the case (Kolodner, 1993). Because SEED-Config deals mostly with hierarchies of building entities, Flemming's hierarchical matching method would be ideal (Flemming, 1994). It compares the hierarchy of the current problem statement with the hierarchy of the retrieved case by considering each of the constituents and determining an overall matching score. The retrieved cases are then ordered based on their overall score. The goal of using such a ranking mechanism is not to select automatically the best matching case, but to rank the set of retrieved cases for the designer to browse. It is the designer's responsibility to select the most appropriate case for adaptation.

7. CASE ADAPTATION

The last of the four major aspects of CBR systems is case adaptation. Once a suitable case has been identified, it needs to be adjusted (or adapted) to the current problem to provide a complete solution. This adaptation is required because old situations never exactly match new ones. Case adaptation solves the current problem by either reusing parts of the past solution or by adjusting the past solution so that

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

it fits the current context. This section first discusses the adaptation method selected for SEED-Config. This is followed by a description of how the adaptation process is implemented.

7.1. The adaptation method

Most adaptation methods described in the literature automatically fix a retrieved case to fit the new situation. In contrast, recomputing a new solution using the same procedures that were used to obtain an old solution is a more appropriate and intuitive adaptation method for designers. Students, for instance, typically reuse procedures; whenever they face a new problem, they try to find a similar example in their textbook and follow the procedure described to solve the assigned problem. This adaptation method is called derivational replay, which reenacts the set of reasoning steps used to produce the retrieved case in the new situation (Carbonell, 1986). Each reasoning step is validated in the new situation and reenacted when found to be applicable. This adaptation method is particularly useful when the solution of a case is computed and depends on some parameters that vary from one problem to another.

Implementing derivational replay in SEED-Config is straightforward because of the way design knowledge is recorded in technology nodes, described in Section 4, which are arranged in a technology graph. Technology nodes record complex design decisions, represent every step of the design process, and can be reenacted, whenever needed, in a new context. A building entity (i.e., a case in SEED-Config) refers to the set of technology nodes that were used to design it. This set represents a path in the technology graph. This path, which records the set of applicable alternatives that were selected by the designer, can be reenacted in a new design situation. During case adaptation, past design decisions are reassessed in the new design context by examining each technology node in the path. If the range of applicability of a technology node is satisfied in the new design context, then the technology is applied to the current building entity, thus extending it, and matching continues to the next level in the path (Rivard et al., 1996). Otherwise, if a constraint of a technology node is violated, that technology and all subsequent technologies in the path are discarded. Once the adaptation is carried as far as it can be, SEED-Config offers three methods to replace technologies eliminated from the path and to complete the design of the building entity: manual, where the designer instantiates the various constructs directly (e.g., building entities, components, and classifier instances); interactive, where the designer selects applicable technology nodes and applies them to the current building entity; and case-based design, where a new case retrieval is initiated at this more detailed level. These three design methods are fully integrated in the same design environment, thus satisfying the corresponding requirement specified in Section 2.

An example of a design path in a technology graph for the design of structural slabs is illustrated in Figure 8 (Rivard et al., 1996). The design path consists of the bold technology nodes and links in the graph; it represents the set of decisions taken by the designer in the retrieved case. Technology nodes and links represented with dashed lines were not applicable when the original designer was in his/her design session. The remaining nodes represent alternatives that were applicable but were not selected. When this path is retrieved and reenacted in the new design context, "Concrete Double-Tee" becomes inapplicable due to the violations of some constraints. Therefore, to conclude the adaptation process, the designer only needs to complete the design by selecting another alternative, say, for example, "Concrete-Hollow-Core."

7.2. Implementation of the adaptation process

We can distinguish two kinds of technology nodes: nodes that have built-in procedures to generate all the required values (e.g., the "Steel one-way" technology automatically determines the design span as the shorter span of a slab), and nodes that need to request data from the designer (e.g., the "Deck-on-joists" technology requests the joist spacing from the designer). The second type of technology node, labeled interactive, has two design modes depending on when it is used. During a normal design session, the technology node interacts directly with the designer. During a case-adaptation session, the technology obtains the data from the case instead of interacting with the designer. The advantage of the second mode is that it spares the designer's time as it reuses previous design values. The designers can always change those values afterward.

The adaptation process is illustrated with an example that is depicted in Figure 9. For simplicity's sake, the example focuses on the instantiation of design unit components and subentities only, and does not discuss the instantiation of other constructs. A structural engineer is designing the roof of an indoor swimming pool. The building entity representing the roof has already been created and two technology nodes have been applied to it: "Horizontal" and "Steel-

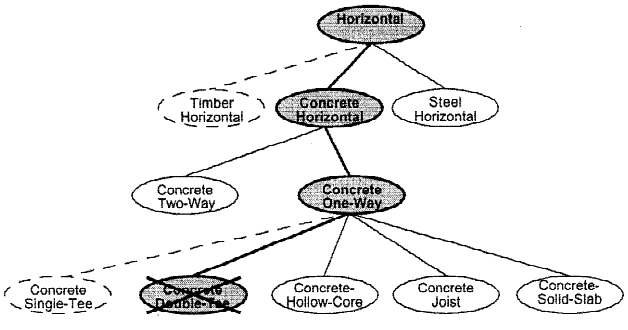


Fig. 8. A design path in a technology graph (Rivard et al. 1996).

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

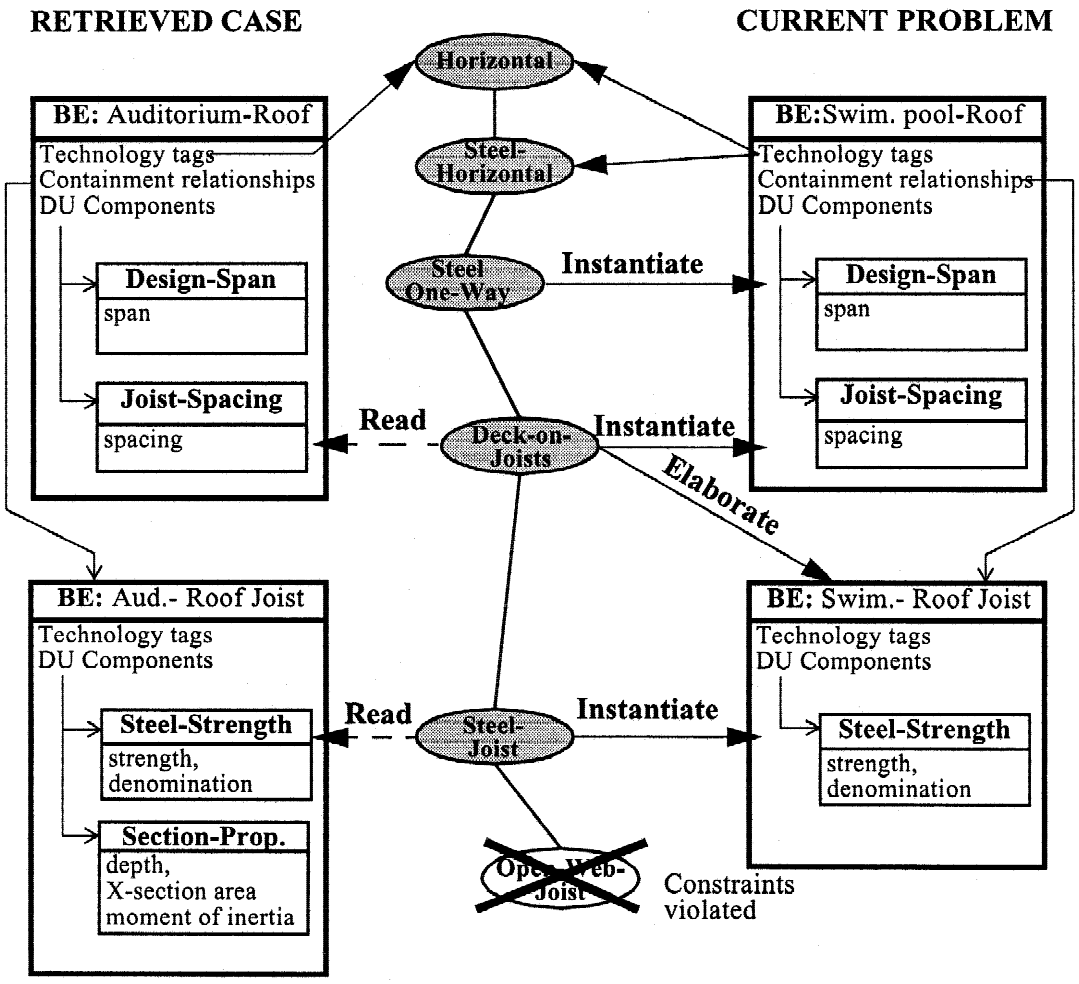


Fig. 9. Example of case adaptation.

Horizontal." The structural engineer concludes the case retrieval process by selecting a case representing the design of an auditorium roof. The adaptation process iterates through the design path of the retrieved case until it finds "Steel one-way" as the first technology node that has not been applied to the current building entity. Once "Steel one-way" is found to be applicable, it is applied to the current building entity, thus automatically instantiating the "Design-Span" component. The process then moves to the next technology node in the path and finds "Deck-on-joists" to also be applicable. This technology node, which normally requires the spacing from the designer, reads the value from the retrieved case and instantiates the "Joist-Spacing" component. The technology node creates a set of joists and a slab as subentities of the roof. Only one joist is shown in Figure 9. The adaptation process moves to the next technology node and determines that "Steel-Joist" is also applicable in the current design context. The technology is applied and obtains the strength of the steel from the retrieved case to instantiate the "Steel-Strength" component in the current entity. The process ends when a constraint of the technology node "Open-Web-Joist" is violated. As explained above, the structural engineer can complete the design of the roof either by initiating a new case-based retrieval at this more detailed level or by selecting a different branch in the technology graph.

Hence, a case in SEED-Config contains both design information and references to procedures. The design information stored in the building entity is used to retrieve cases and to obtain data usually acquired interactively. The procedures are technology nodes that support the reenactment of the design sequence and represent the reusable part of a case. The capability of reusing both design information and procedures is in sharp contrast with most other CBR systems that reuse either one or the other but not both.

8. CONCLUSIONS

SEED-Config is a design environment for conceptual building design with a case-based reasoning functionality so that designers do not have to start from scratch for each new design. The development of this system began with the creation of a list of seven requirements for CBR systems intended for conceptual building design. None of the CBR systems reviewed from the literature satisfied all these re-

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

quirements. This list guided the development of the case-based reasoning functionality.

The case representation in SEED-Config is the BENT information model. It supports the hierarchical decomposition of design cases. The parts of a case are as simple to access as the actual global case, because each part is stored as a whole and is indexed independently. The case representation also supports multiple views by aggregating small cohesive sets of attribute-value pairs into components which may be shared among different views. Moreover, the case representation encapsulates the outcome in addition to the problem specification and the design solution.

A case library was implemented in an OODBMS for accumulating cases and for providing efficient query facilities. The case accumulation aspect was developed to provide automatic archiving capabilities. The only effort required from the designer is the selection of a building entity in the hierarchical decomposition and the activation of the process through a menu command.

The case retrieval aspect of SEED-Config offers three different methods to create a query based on the current problem statement: task-based, lineage-based, and customized. The case retrieval process described here supports the exploratory nature of the design process by supporting incomplete initial problem statements and by allowing them to change and expand as design progresses. The case retrieval process can rely on multiple paths to find a case. Case queries are not limited to problem statements because the design solution, the design process steps, the evaluation, the relationships, and aspects of the geometry are all searchable. This provides flexibility for the designer in querying the design knowledge stored in the case library.

The case adaptation aspect was elaborated with an adaptation method called derivational replay that reenacts the set of reasoning steps used to produce the retrieved case in the new situation. The case-based design capabilities are completely integrated with the design environment from which the cases originate.

The outcome of this research was a prototype implemented to demonstrate the approach. The prototype was a proof of concept and was tested in laboratory conditions only. It successfully demonstrated that the case-based design capabilities of SEED-Config satisfies the seven specified requirements. However, additional research efforts are required to further assess the performance of the system, to perfect the user interface, and, finally, to evaluate the approach with user studies and on-site prototype evaluations.

ACKNOWLEDGMENTS

The authors wish to thank the Natural Sciences and Engineering Research Council of Canada, the “Fonds pour la formation de chercheurs et l’aide à la recherche (Fonds FCAR)” of the province of Quebec, and the Engineering Design Research Center of Carnegie Mellon University for financially supporting this research project.

REFERENCES

Bailey, S.F., & Smith, I.F.C. (1994). Case-based preliminary building design. *Journal of Computing in Civil Engineering 8(4)*, 454–468.

Bharat, D., Schmitt, G., Shih, S.-G., Bendel, L., Faltings, B., Smith, I., Hua, K., Bailey, S., Ducret, J.-C., & Jent, K. (1995). Case-based spatial design reasoning. In *Advances in Case-Based Reasoning*, (Haton, J.-P., Keane, M., and Manago, M., Eds.) pp. 198–210. Springer-Verlag, New York.

Carbonell, J.G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In *Machine Learning: An Artificial Intelligence Approach*, (Michalski, R.S., Carbonell, J.G., and Mitchell, T.M., Eds.), pp. 371–392. Morgan Kaufmann Publishers Inc., Los Altos, California.

Eastman, C.M., & Fereshetian, N. (1994). Information models for use in product design: A comparison. *Computer-Aided Design 26(7)*, 551–572.

Fenves, S.J. (1995). Successes and challenges in computer-aided structural engineering. *Proc. Sixth Int. Conf. on Computing in Civil and Building Engineering*, (Pahl, P.J., and Werner, H., Eds.), pp. 13–20. A.A. Balkema Pub., Rotterdam, Netherlands.

Fenves, S.J., Rivard, H., Gomez, N., & Chiou, S.-C. (1995). Conceptual structural design in SEED. *Journal of Architectural Engineering 1(4)*, 179–186.

Flemming, U. (1994). Case-based design in the SEED system. In *Knowledge-Based Computer-Aided Architectural Design*, (Carrara, G. and Kalay, Y., Eds.), pp. 69–91, Elsevier, New York.

Flemming, U., & Woodbury, R. (1995). Software environment to support early phases in building design (SEED): Overview. *Journal of Architectural Engineering 1(4)*, 147–152.

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

Gomez, N. (1998). *Conceptual structural design through knowledge hierarchies*. Ph.D. Thesis. Carnegie Mellon University, Pittsburgh, PA.

Gowri, K. (1990). *Knowledge-based system approach to building envelope design*. Ph.D. Thesis. Concordia University, Montreal, Canada.

Gowri, K., Iliescu, S., & Fazio, P. (1995). A case-based model for building envelope design assistance. *Proc. of the Second Congress of Computing in Civil Engineering*, (Mohsen, J.P., Ed.), pp. 171–178. ASCE, Washington D.C.

Kolodner, J.L. (1991). Improving human decision making through case-based decision aiding. *AI Magazine*, *12(2)*, 52–68.

Kolodner, J.L. (1993). *Case-Based Reasoning*. Morgan Kaufmann Publishing Co., San Mateo, California.

Krishnamoorthy, C.S., Kumar, H.S., Rajeev, S., & Suresh, S. (1993). Knowledge based system with generic tools for structural engineering design. *Structural Engineering Review 5(2)*, 121–131.

Kumar, H.S., & Krishnamoorthy, C.S. (1995). A framework for case-based reasoning in engineering design. *AI EDAM 9*, 161–182.

Kumar, B., & Raphael, B. (1997). CADREM: A case-based system for conceptual structural design. *Engineering with Computers 13(3)*, 153–164.

Maher, M.L. (1984). *HI-RISE: A knowledge-based expert system for the preliminary design of high rise buildings*. Ph.D. Thesis. Carnegie Mellon University, Pittsburgh, Pennsylvania.

Maher, M.L., & Balachandran, M.B. (1994). Multimedia approach to case-based structural design. *Journal of Computing in Civil Engineering 8(3)*, 359–376.

Maher, M.L., Balachandran, M.B., & Zhang, D.M. (1995). *Case-Based Reasoning in Design*. Lawrence Erlbaum Associates Publishers, Mahwah, New Jersey.

Maher, M.L., & Pu, P. (1997). Introduction to the issues and applications of case-based reasoning in design. *Issues and Applications of Case-Based Reasoning in Design*, (Maher, M.L. and Pu, P., Eds.), pp. 1–10. Lawrence Erlbaum Associates Publishers, Mahwah, New Jersey.

Maher, M.L., & Zhang, D.M. (1993). CADSYN: A case-based design process model. *AI EDAM 7(2)*, 97–110.

Pearce, M., Goel, A., Kolodner, J.K., Zimring, C., Sentosa, L., & Billington, R. (1992). Case-based design support: A case study in architectural design. *IEEE Expert 7(5)*, 14–20.

Phan, D.H.D., & Howard, H.C. (1993). *The primitive-composite (P-C) approach—A methodology for developing sharable object-oriented data representations for facility engineering integration*. Technical Report 85 (A and B), CIFE, Stanford University, CA.

Raphael, B., & Kumar B. (1996). Indexing and retrieval of cases in a case-based design system. *AIEDAM 10*, 47–63.

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press

Rivard, H. (1997). *A building design representation for conceptual design and case-based reasoning*. Ph.D. Thesis. Carnegie Mellon University, Pittsburgh, PA.
Rivard, H., & Fenves, S.J. (2000). A representation for the conceptual design of buildings. *Journal of Computing in Civil Engineering, ASCE, 14(3)*, 151–159.
Rivard, H., Gomez, N., & Fenves, S.J. (1996). An information model for the preliminary design of buildings. *Proc. of the CIB W78/ TG10 Workshop Construction on the Information Highway* (Turk Z., Ed.), pp. 435–448, CIB Proceedings, Pub. 198, Rotterdam, Netherlands.
Sanvido, V.E., Fenves, S.J., & Wilson, J.L. (1992). Aspects of virtual master builder. *Journal of Professional Issues in Engineering Education and Practice 118(3)*, 261–278.
Schank, R.C. (1982). *Dynamic memory: A theory of reminding and learning in computers and people*. Cambridge University Press, Cambridge.
Simon, H.A. (1996). *The sciences of the artificial*, 3rd ed., The MIT Press, Cambridge, Massachusetts.
Snyder, J., Aygen, Z., Flemming, U., & Tsai, J. (1995). SPROUT—A modeling language for SEED. *Journal of Architectural Engineering 1(4)*, 195–203.
Wang, J., & Howard, H.C. (1991). A design-dependent approach to integrated structural design. *First Int. Conf. on Artificial Intelligence in Design*, (Gero J.S. Ed.), pp. 151–170. Butterworth-Heinemann, Oxford.
Watson, I., & Perera, S. (1997). Case-based design: A review and analysis of building design applications. *AIEDAM 11(1)*, 59–87.
Woodbury, R., & Chang, T.-W. (1995). Massing and enclosure design with SEED-Config. *Journal of Architectural Engineering 1(4)*, 170–178.
Zamanian, K.M., & Fenves, S.J. (1994). A referential scheme for modeling and identifying spatial attributes of entities in constructed facilities. *Research in Engineering Design 6(3)*, 142–168.
Zhao, F., & Maher, M.L. (1988). Using analogical reasoning to design buildings. *Engineering with Computers 4*, 107–119.

Hugues Rivard is an Assistant Professor of Building, Civil, and Environmental Engineering at Concordia University. He is a member of the Centre for Building Studies. He completed his Ph.D. in Computer-Aided Engineering in the Department of Civil and Environmental Engineering at Carnegie Mellon University in 1997. His main research interest deals with the conceptual building design process and how it can be assisted by computers.

Steven J. Fenves is University Professor Emeritus of Civil and Environmental Engineering at Carnegie Mellon University. He received his degrees in Civil Engineering from the University of Illinois. He taught at the University of Illinois (1958–1971) and at Carnegie Mellon University (1972–1998). He is currently a Senior Research Associate at the National Institute for Standards and Technology. Dr. Fenves' research has dealt with computer-aided engineering, design standards, engineering databases, knowledge-based systems, and design environments. Dr. Fenves is the author of six books and over 300 articles. He is a member of the National Academy of Engineering and an Honorary Member of the American Society of Civil Engineers.

https://doi.org/10.1017/S0890060400145068 Published online by Cambridge University Press