Representing Design Decisions: An Object Oriented Approach

Filiz Ozel
Arizona State University
fozel@asu.edu

During the course of a design project numerous design decisions are made, usually with little attention paid to documenting them or keeping track of them. Systematic documentation and representation of design decisions can not only be invaluable in learning from past design experiences, but can also be good tools in teaching architectural design. By using abstraction and analogy to analyze a design precedent, a problem/solution hierarchy can be built where similarities and differences between the precedent problem and the target problem, goals, constraints and solutions are identified for each level of the hierarchy. Each one of these can be represented as objects in an object oriented programming environment, allowing the construction of a hierarchical structure. This model was incorporated into a computer-sustained learning system called "DesignRep" which was created by using RedBook (Aquaneering Co.), object-oriented development environment.
Introduction

Students of architectural design must master a very wide range of issues and be able to use them effectively as their design skills develop. With the emergence of computer technology as a medium, not only the representation of final designs in the form of 2D graphics and 3D models, but also the representation, dissemination and use of design knowledge through the use of computers are revolutionizing the way architects access and use information. While the design of an architectural artifact can be greatly affected by the design representation employed during conceptualization, the use of design knowledge can be equally affected by how this knowledge is represented, modeled and presented to the designer. This becomes even more critical when the designer is a student who is yet to develop skills in using design knowledge effectively as s/he attempts to master it and apply it to a given problem.

Traditionally, design knowledge has been represented in terms of:
- rules and regulations (as in building codes, etc.)
- rules of thumb that are formulated by the daily experiences of the designer
- prototypes such as building typologies and the knowledge found in Architectural Graphics Standards
- precedents (i.e. past cases) of architectural design that are time tested and are considered examples of good design.

Among these, using precedents as a source of design knowledge can provide a more holistic approach to design teaching. At the root of the tradition of relying on architectural precedents lies the idea that these designs contain knowledge that can be extracted and applied to new situations, to new designs. The nature of this knowledge can be in the form of final design artifacts on the one hand and in the form of history of design decisions on the other hand. Typically, in using precedents to teach architectural design, more emphasis is placed on learning from the artifact mainly because this piece of information is more readily available.

Many times, since no documentation is available on design decisions, students of design try to understand the underlying design decisions and how they led to the design solution at hand by studying the final artifact. In this process, a mechanism that represents design decisions could aid the student in systematically documenting design decisions and in understanding the relationship between the sub-problems of a design problem and the solutions to them in a precedent case.

Coyne, et al. (1990) stress the importance of learning models, and continue to indicate that “Learning is said to occur when a system constructs or modifies its representations.” People typically build representations of new knowledge by using some of the existing models they already possess. Coyne et al. (1990) point out that a sign of intelligence is that if a person learns a lesson from an experience they had and apply what s/he has learned to new situations effectively. “We do not merely extract the relations that are explicit in a particular situation, but generalize and abstract concepts so that we can apply them in different but analogous situations.” (Ibid., p.460). At the root of the learning process through studying past design decisions lies the concepts of generalization, abstraction and analogy which allow the student to comprehend the decisions made in a past case more holistically and apply this knowledge to a new case. In this article, the author presents a conceptual model that explores the potential of abstraction and analogy in design decision representation and continues to introduce a computer assisted learning system that implements this conceptual model. In this process an object oriented approach is taken due to the hierarchic nature of design problem-sub-problem relationship.
the mechanisms for
design decision representation

The basis of modeling and representing design decisions is analogous, where the abstraction common to two analogous problems is identified in order to know what can be transferred from one problem to another. Two fundamental components of this representation are:

- Recalling some sample design decision cases in previous situations, i.e. precedents,
- Extracting concepts that can be used in a new situation where these concepts are ranked in the order of similarity to the current problem, weighing similar goals more heavily than similar solutions.

In this process, the first step is to create an abstract schema that represents what the source (precedent problem) and the target case (current problem) have in common. In architectural design, while the architectural program of a current design problem can be the basis for an abstract schema, identifying the theoretical underpinnings of a precedent case and making an analogy to the target case at this level of understanding provides a better foundation for extracting knowledge and learning from a precedent case.

In this process, the abstract schema consists of:

- Problem schema, created by mapping the precedent problem to the current problem.
- Solution schema, this is created by using the precedent case. Then the solution is applied to the design problem at hand. This is usually called "abstraction analogy". (Shinn, Hong S., 1988, Proceedings of Case-based Reasoning Workshop, p.579). Abstraction analogy can lead to the transfer of reasoning mechanisms and/or generalized results from a precedent case to a current problem at hand. Both types of knowledge are formulated in the form of an abstract solution.

According to Dradon (1983) and Ross (1986), two contradicting views coexist in the process of analogical transfer:

- Direct transfer of knowledge between a precedent case and a current case, where correspondences between them are established and knowledge is directly interpreted within the domain of the current problem.
- Indirect transfer through a common abstraction.

In the past, most artificial intelligence systems have subscribed to the first view and used the direct transfer of knowledge method (Winston, 1980). Whereas, recently researchers began to explore the indirect method. Shinn (1986) quotes Genesereth (1986): "...the problem of understanding an analogy becomes one of recognizing the shared abstraction."

In architectural design, there has long been a tradition of analogy, a shared abstraction that can be applied to a current design problem by extracting it from a precedent design problem. To the extent a designer or a design student can create an abstract schema of a design problem, she can successfully identify the underlying reasoning processes that have led to a given design solution. This in fact underscores the importance of the differentiation between analogical abstraction of a given solution and analogical abstraction of a given problem. The former can be placed within the realm of direct transfer of knowledge, whereas the latter would correspond to the indirect transfer of knowledge.

In fact these two approaches lead to two completely different models of problem solving and learning. In using precedent design decisions to teach architectural design, this distinction emerges as a very real issue that must be carefully examined. The direct transfer view implies that generalization and learning occur after problem solving, on the other hand the indirect view suggests that they occur during problem solving. In computer assisted learning environments that rely on precedents, with direct transfer method solutions will have to be readily provided, whereas with the indirect method the history of the design decision and
reasoning process must be provided or request
ed from the student so that learning can occur
during problem solving. The transfer of a pre-
vious design solution—possibly with some minor
modifications—shortcuts the reasoning involved
for a similar problem by reducing its search for
a solution. On the other hand, this also greatly
undermines the learning process, since it does
not lead to generalizations and does not devel-
oping the ability to transfer this knowledge to new
design problems. Examples of direct transfer in
architectural design processes are:

- Using building typologies,
- Transferring the architectural vocabulary
  of an existing building,
- In architectural technology adopting stan-
dard construction details or structural solutions
- Transferring a technological solution
directly from a precedent building, etc.

Genna and Zveik (1995) propose to use anal-
ogy to obtain solutions to design problems by
collecting and storing hypotheses about vari-
able constraints that were satisfied in prece
dent cases. Shinn (1989) indicates that two
different types of knowledge are transferred
during analogical reasoning:
- the methods used in solving a precedent
  problem and
- the results (i.e. the solution).

This is in fact similar to the concept of
direct and indirect transfer discussed earlier.
Additional arguments brought by Shinn on
the topic can have interesting ramifications in
developing computer assisted learning tools that
employ methods that document and represent
past design decisions. He argues that when
result transfer (direct) is not appropriate, rea-
soning can be transferred (indirect), and that
these two methods can be applied at different
stages of the problem solving process. By stag-
ing the learning process in terms of direct trans-
fer (solution transfer) during the early stages of
the learning process and indirect transfer (rea-
soning transfer) during its later stages, design
students can gain self-confidence by being
equipped with a larger domain of references
they can use for design decision making. This
becomes very important in self paced, self guid-
ed environments where computers are used to
teach architectural design.

Abstraction analogy integrates these two
types of analogical knowledge transfer.

- In the first step of the process, an abstract
  schema is created by analogically mapping
  the precedent problem to the current problem.
- Then a solution schema is created by using
  the precedent case.
- This solution schema is applied to the cur-
  rent design problem.
- An additional step refines the solution
  obtained by analogy to fit the constraints of
  the current problem that were not covered.

representing design decisions in architecture

Rosensman et al. (1992) classify design knowl-
edge as 1) compiled or general knowledge and 2) a
 case knowledge. They continue to define com-
piled knowledge as the one “that is obtained
from a number of individual experiences,
...which may be thought of as rule-like...”
/Schmitt, G., Ed., p.385. General knowledge can
be considered almost as a set of rules of thumb
that tell us when to use what. This is in fact the
typical model followed in representing knowl-
edge about architectural technology. For example,
one can easily identify a generic roof detail as a
solution to the problem of controlling rain water
in buildings by looking it up in a reference book
such as Time Saver Standards (Callender, 1987) or
in a textbook on the topic. Although such knowl-
edge can tell us how to solve a generic problem,
it does not put a given technological problem
within the broader context of other issues that
might have impacted the derivation of a specific
design solution to this technological problem.
For example, formal design considerations, cost,
regional climatic idiosyncrasies may be additional factors that can play a role in how one resolves the problem of rainwater control. Thus, compiled knowledge does not provide a holistic approach to presenting and representing architectural knowledge.

On the other hand, when the history of design decisions is documented to study a single precedent in depth with all the factors that might have affected a given design sub-problem and how it was resolved, each sub-problem and its specific solution is placed within the context of a larger design problem and its solution. In learning from such an experience, the problem schema must show the similarities as well as the differences between the precedent problem and the target problem. The solution schema for a precedent design problem must contain not only the solution, i.e. the final design, but also a history of the decisions made by the designer, including other options that might have been considered during the design process and the reasons for not adopting those other options. The logic and reasoning behind these decisions can be transported to new designs, usually after some modification. Furthermore, if the final design is realized as a built structure, researchers also have the opportunity to explore the nature of the designer’s intent in employing an architectural design solution and its manifestations in terms of building performance. Providing such cases in a computer assisted learning environment can not only allow the students to explore the reasoning behind a design decision but also to judge how well this reasoning has worked. This in fact introduces a third step into the process of abstractional analogy, the process of exploring the effectiveness of the reasoning process. Such a feedback mechanism is needed in order to reassess the reasoning process that was used for a given case.

As a summary, computer assisted learning environments that represent and document design decisions in past cases should contain the following parameters:

- problem schema
  - Definition of the problem in terms of goals and constraints: problem abstraction
  - Analogical abstraction of the problem by identifying similarities as well as differences between the precedent and the target problem,

- solution schema
  - Definition of the solution with two parts: a solution diagram and a reasoning history. The physical properties and geometries of objects that facilitate such goals and objectives with given constraints as well as intermediate solutions must be included.
  - The evaluation of the reasoning history based on how well the solution have met the criteria set in the precedent design problem, and thus possible modifications to the reasoning process when applied to the current design problem.

Thus each case must be represented in terms of a problem description, a history of the decisions that were made, a solution and an assessment of the solution. Rosenman et.al. (1992) stress the importance of including a history of the justification of the decisions made and the reasons for not selecting other alternatives. Especially in those cases such as building code decisions or the selection of materials where tradeoffs exist between different alternatives, documenting the history of the design decisions can be very instrumental in transporting such considerations to other designs.

- case selection
In a general mapping procedure, the current design problem reminds us of a case where one or more aspects of the precedent problem are similar to the current one. While in design oriented case selections the process can be more subjective such as preference for an architect’s
work or preference for an architectural vocabulary, case selection for architectural technology can be more objective and should be based on explicit criteria. This is even more true when case selection must be done for the development of a computer-assisted learning tool. Among these criteria can be factors that are broader in nature such as the original design goals and objectives of the selected precedent, the theoretical framework of the design concept used, the role of architectural technology in the design decisions made by the designer in the selected case, etc. On the other hand there must also be problem-specific criteria such as climatic factors, type of construction used, cost, structural concerns, etc. depending on the area of architectural design that is intended to be covered by the computer-assisted learning tool.

**Problem abstraction**

Once the cases are selected, one must identify the structure of the problem, i.e., a common abstraction of two analogous problems must be made. This is typically accomplished by identifying the similarities between the problems which can be in terms of the goals and the constraints of the precedent problem. Next the goals are mapped to the target problem, if the goals fully match then one proceeds to map the constraints. In case of a partial match, only those goals that match are considered for possible transfer to the target problem. Furthermore, those constraints that are not related to the matched goals are not included in the pool of matched constraints. Issues such as site conditions which are identified as contextual issues by Ouman (1995) are actually constraints that act upon a given design problem and must be incorporated into the process of constraint mapping.

---

Shim establishes the similarities in his case reasoning system by means of an abstraction hierarchy. Given a pair of concepts, a common abstraction is found by simply identifying their immediate ancestor (or parent) within the hierarchy. If the existing hierarchy does not contain an immediate ancestor, an abstract object is created by introducing a new symbol into the system. This new object possesses the properties that are common to both concepts.

Breaking down the target problem into smaller components can also affect the problem matching process. The precedent problem and solutions to these problems can be repre-

![Diagram](image-url)

*Figure 1: Hierarchical organization of problem structure for the representation of design decisions. At each level, design criteria, constraints, and typical next precedent solutions must be included.*
sented in a hierarchical structure, i.e. it may be decomposed into sub-problems at multiple levels of granularity. Such a recursive representation can also assist in the transfer of knowledge to the current design problem at different levels of abstraction. For example, if the current problem is the design of a high rise building, while mapping the problem at this level of granularity will lead to the selection of cases from among other high rise building designs, a further breakdown of the target problem can lead to the selection of other cases. Let’s assume that “facade design” was identified as a sub-problem, then the criteria for the selection of the cases can be modified to include buildings that are not necessarily high rises. On the other hand, the appropriateness of the cases selected based on a sub-problem must be carefully examined. Usually the constraint mapping process will fail if an inappropriate precedent problem is selected. For example, if the precedent problem of “designing a structural system for a medium rise structure” is mapped to the target problem of “designing a structural system for a high rise building,” it is very likely that the constraint mapping process will fail.

As an example for a problem schema, the abstraction hierarchy built during the mapping of the precedent design problem of “Montessori School Building” to the target design problem of “School of Architecture Building” given in a graduate studio (Mostoller & Ozel, 1999) will be as follows:

- (a) Activity: (b) Education (Main Design Problem)
  - (c) Education: (d) Individual level space requirements (Sub-problem)
    - (e) Group level space requirements (Sub-problem)
      - (f) Contextual space requirements (Sub-problem)
        - (g) The relationship of spaces (Sub-problem)
          - (h) Individual Level space requirements

- (Sub-problem)

- (1.x) Provide workspace for each person (Constraint)
- (1.x.2) Provide a flexible individual space within the group space (Constraint)

- (2.x) Group level space requirements (Sub-problem)
  - (2.x.1) Group several individual spaces into a group space (Constraint)
  - (2.x.2) Relate the group spaces to the next level of group spaces (Constraint)
  - (2.x.3) Create a complete hierarchy of group level spaces (Constraint)

- (3.x) Contextual space requirements (Sub-problem)
  - (3.x.1) Create multiple spaces at different sizes for specialized functions (Constraint)
  - (3.x.2) Create transitional spaces between group spaces and contextual spaces (Constraint)

- (4.x) The relationship of spaces (Sub-problem)
  - (4.x.1) Relate individual spaces to group spaces (Constraint)
  - (4.x.2) Relate the root level of the group spaces to the exterior spaces (Constraint)
  - (4.x.3) Relate the lowest level of the group spaces to individual spaces (Constraint)
  - (4.x.4) Relate contextual spaces to exterior spaces (Constraint)
  - (4.x.5) Create transitional spaces between all levels of spaces (Constraint)

solution abstraction

A solution abstraction is created by abstracting the source solution at the same level of abstraction as the problem schema. This also depends on whether a reasoning history exists or not.

- if a reasoning history does not exist, then the result is transferred by generalizing the result to fit the problem schema. In using architectural precedents to abstract a solution usually only a partial reasoning history exists since it is rare that design decisions are recorded as a project develops. Therefore, generalization is the typical mode of knowledge transfer for most precedent based knowledge generation.
When there is a reasoning history, for each of its steps the current state of the schema is tested to see if it meets the preconditions. If the preconditions are not met in terms of the problem schema, then the case is abandoned for another case. For example, in the Montessori School - School of Architecture design mapping process, if the step for the mapping of spatial requirements has not met the preset criteria for the target problem, then the precedent case would have been abandoned.

On the other hand, a history of design decisions as it applies to problems in architectural technology can be more readily developed, and is a good candidate for teaching the role of architectonics can play in design decisions. This is usually formulated in terms of the design criteria for the selection of the technology used and the selection of the specific system that would meet the specified criteria, since usually there will be more than one solution that can fulfill a given criteria.

A typical model incorporated into books and reference material that provide compiled design knowledge in the area of architectural technology is as follows:

- identify a sub-problem
- identify the set of design criteria
- present a solution set that would meet each criterion
- identify the criteria for the selection of a specific solution from a solution set

An example of this exists in a widely used reference book (especially by design students): “The Architect’s Studio Companion: Technical Guidelines” (Allen & Iano, 1989). Similar to many other books on architectural technology, this book contains compiled or general knowledge about specific technological sub-problems. In the book, major chapters actually refer to sub-problems, among which are structural systems, mechanical systems, egress, etc. The next step is to identify some possible design criteria, although this is not done consistently for every sub-problem in this reference book. For example, the title “Designing Spaces for Mechanical and Electrical Services” is identified as a sub-problem. Within the hierarchy of sub-problems which is also called “an issue” by some researchers (McGill et al., 1990; Osman & Osman, 1993), additional levels exist such as “Selecting the heating and cooling systems appropriate for large buildings”. This means before a lower level sub-problem is resolved, one cannot resolve a sub-problem that is higher in the hierarchy.

Then the criteria for each sub-problem is identified. In this example, ten different criteria are identified for the sub-problem “Select an appropriate heating and cooling system for large buildings” among which are “minimize operating cost” “maximize quality of air quality and air velocity”, etc. (Allen & Iano, 1989). On the other hand the criteria for the sub-problem “design the spaces that will house these systems” which is higher in the hierarchy are not identified in this reference book. This is partially because this sub-problem can heavily rely on the design concept employed for the whole building. In fact, the higher a sub-problem is in the hierarchy the more holistic an approach must be taken in presenting the problem and its possible solutions. Therefore, precedent based representation of design decisions can be very effective in providing design knowledge about the problems that are closer to the top of the hierarchy.

The process of abstractional analogy must start at the top of the “problem/sub-problem” hierarchy. For example, by specifically identifying large buildings as the domain of the sub-problem “Selection of heating and cooling systems for large buildings”, Allen and Iano (1989) are actually identifying a design problem that is higher in the hierarchy: the problem of “Designing a large building”. Clearly, “Selection of heating and cooling systems” is only one of many sub-problems associated with the design
of a large building, thus only represents a branch of the problem hierarchy.

The conceptual model for the representation of design decisions in teaching architectural design that is proposed by this researcher contains the following six level process:

- Identify the schema for the precedent design problem in terms of goals, objectives and constraints.
- Identify all sub-problems that are a child of the problem identified above and repeat this process until no lower level sub-problems can be identified.
- Identify the design criteria for the sub-problems at each level of the hierarchy.
- Identify the design decision for the precedent solution at each level of the hierarchy.
- Identify the criteria for the selection of a solution from the solution set.
- Identify a solution schema based on the precedent for each sub-problem. The solution schema must contain topological and geometric definitions of the precedent solution as well as any incidental factors that might have affected the selection of the solution.

**Computer-assisted learning environment**

Based on the conceptual model developed here to analyze architectural designs through the representation of design decisions for a precedent case, a template was prepared using an object-oriented development environment called ToolBook (Asymetrix Co.). The system provides a framework for creating an abstract schema for a given case by focusing both on the precedent design problem and the precedent design solution. It is formulated as a tool that would assist the student designer to represent architectural knowledge that is extracted from past cases. **(Figure 2)** Although it can be used to create a library of cases that are selected, analyzed and stored by the instructor, it can also provide a framework for the execution of the case analysis by the students. The pedagogical merit of the latter originates from the belief that students grasp concepts better and more clearly.

*Figure 2: Main problem page for a precedent design problem.*
ly when they are involved in all stages of the knowledge extraction process.

This object-oriented programming environment reinforces the need to assess the hierarchical nature of problem decomposition and the links between different levels of the hierarchy as well as the links between solutions. One must create abstractions of the problems within the hierarchy by finding commonalities between the precedent problem and the target problem. The system also keeps track of the links within the hierarchy of problems while new sub-problems are created as offsprings of existing ones (Figure 3). In this process, the graphic interface that displays the problem-tree structure plays a pivotal role in navigating through the problem hierarchy and in inputting new sub-problems (Figure 4). During the analysis of a precedent problem and its solution, one must consider how and what aspects of precedent knowledge can be transferred to a current design problem and represent these aspects in addition to the general documentation available for a given precedent.

The ToolBook (Asymetrix Co., 1995) object-oriented development environment also allows the importing of vector and bitmap images. Images from CAD systems can be imported as .dxf files as well as scanned images in a number of formats such as .bmp, .gif, .tif, etc. Therefore, the use of multiple software in representing design knowledge can be explored through the system. Toolbook includes the option to use predefined objects as well as the possibility of creating user-defined objects. Due to its object-oriented nature, this software was found to be suitable for developing the hierarchical design decision representation system of the application called "DesignRep."

Figure 3 View of a sample sub-problem in DesignRep.
the implementation of “designmap”

System objects available in Toolbook originate from a number of concepts such as the “Book” metaphor which provides pages and backgrounds as objects, Windows operating system objects such as viewers/windows, dialog boxes, fields, radio buttons, buttons, etc., the hypercard concept providing hotlinks or hotwords as objects, and the graphic tools that include vector graphic objects such as rectangles, circles, lines, etc. It is an event driven software where events such as enter page, leave page, mouse click, mouse up or down, etc. can be used to control the flow of an application by the user. Toolbook comes with a full programming language called Open Script and also includes features such as auto-scripting (pre-programmed pieces of code that can be attached to events) and record feature that records the actions of the programmer as an Open Script code which can then be attached to any object in the system to define its behavior. Object classes are defined through their attributes and behaviors as it would be in any object oriented software. Through the use of grouping of system objects and by attaching user defined attributes to any single or grouped object, i.e. through the use of the principle of inheritance, new objects can be defined. On the other hand, due to encapsulation unique data sets (attributes) and procedures (methods) can be defined for any system or user defined object class. Toolbook also supports polymorphism, where for example, a user defined attribute called “area” can be calculated through different methods attached to appropriate geometries, i.e. a circle object can have a different method to define its “area” attribute than a triangle object. Furthermore, object linking and embedding is also possible where other Windows applications can be attached to an application developed in Toolbook.
This structure has allowed the development of the following user defined objects to represent the hierarchy of design decisions:

1. Main structure:
   - **CLASS Project** (Parent problem at the top of the hierarchy. User defined object class generated by inheritance from system object "Case")
   - **CLASS Abstraction** (Child of the "Problem" object)
     - Similarities (Property of "Abstraction" object)
     - Differences (Property of "Abstraction" object)
   - **CLASS Sub-problems** (Can be the child of a "Problem" object or be a child of another "Sub-problem" object)
   - **CLASS Solution** (Child of the "Problem" object)
     - Goals (Property of the "Problem" object)
     - Design Criteria (Property of the "Problem" object)
     - Constraints (Property of the "Problem" object)

2. Expansion of the classes defined above:
   - **CLASS Sub-problem**
     - **CLASS Abstraction** (Child of the "Sub-problem" object)
       - Similarities (Property of "Abstraction" object)
       - Differences (Property of "Abstraction" object)
   - **CLASS Solution** (Child of "Sub-problem" object)
     - Goals (Property of "Sub-problem" object)
     - Constraints (Property of "Sub-problem" object)
     - Criteria (Property of "Sub-problem" object)
   - **CLASS Solution**
     - **CLASS Solution** (Property of "Solution" object)
       - System object
       - Goals (Property of "Solution" object)
       - Constraints (Property of "Solution" object)
       - Criteria for selection (Property of "Solution" object)

Every user defined attribute can be viewed through a "Field" system object, or can be constructed through an interface that uses fields and unique dialog boxes. Conceptualizing viewers separate from actual object attributes allows the flexibility to view any branch of the hierarchy through a corresponding viewer without necessarily displaying all of the attributes of a given object.

The application "DesignRep" displays the tree structure of the problem/sub-problem hierarchy through a graphic interface. This tree structure is automatically built as the user inputs the problem and solution schema. The graphic interface plays an important role in navigating through the problem hierarchy. [Figure 4]

"Expand" feature allows the user to view selected objects and their attributes in more detail.

**Summary**

Architectural precedents can provide a valuable source of design knowledge, not only for formal aspects of design problems but also for problems related to architectural technology. Through the use of abstraction and analogy, one can analyze a given design precedent. This is typically accomplished by building a problem-sub-problem hierarchy where goals, constraints and solutions are identified for each level and each branch within the hierarchy. This conceptual model was incorporated into a computer assisted learning environment called "DesignRep" which was created by using Toolbook (Asymetrix Co.) object oriented development system. Representation of such an analysis process through a computer assisted learning environment encourages the student to be systematic in his/her approach to the problem of design analysis while providing a framework for a holistic approach to mastering architectural design concepts. Future uses of the system encompasses as diverse areas as representing design decisions regarding philosophical and historical points of view to decisions that are primarily related to the role of architectonics in
design. An aspect of the system that needs further development is the type of navigational tools that are currently available. Currently, the user can interactively select a sub-problem and view it in more detail, or go to its parent or to its children to view them in more detail. There are plans to supplement this by other search algorithms that allow the matching of different properties of the objects with each other. Furthermore, to make the system more versatile, additional editing features such as global editing also need to be included.

**References**


Graesser, A.C. 1986 “Metaphors and Models”. In the Proceedings of AIAA-8, Menlo Park, CA.


