Model
for an Integrated Design Evaluation System
using
Knowledge Bases

by
Uk Kim

The University of Michigan
Architecture and Planning Research Laboratory
1000 Bonisteel Boulevard
Ann Arbor, Michigan 48109
Abstract

Computer-aided architectural design (CAAD) systems need to be integrated so that one unified system can generate and do various analysis and evaluation of building models. A database system can not solve this problem because all design concepts can not be stored in the database before the design is completed. As design stage proceeds, design concept and necessary information for analysis and evaluation become complex and detailed. In order to accommodate increasing entities and new relationships between them, knowledge-based systems are integrated into the database of building models. FRAME structure and PRODUCTION RULES are adopted to represent knowledge about the database, and to represent evaluation rules respectively. The system is implemented in Prolog on an Apollo workstation.

Introduction

Current computer-aided design, analysis and evaluation systems store information describing building models in a database so that they can be used for specific design analysis or evaluation. Currently available databases are not appropriate for the integrated analysis and evaluation system mainly because the relationships between entities and the hierarchy of relationships, if any, are predefined according to the primary purpose. Entities for one aspect are not articulate for others in a procedural control structure or program. Therefore, different models for different analyses and evaluation have been developed. Each of these systems solves problems in one domain well, and yet it is difficult to integrate them under one unified model to solve problems in various domains. This study reports on the experimental research for the integration of a database and a knowledge-based system to develop a CAAD system with more analytical and evaluating power. It is one of the projects from the on-going seminar for developing computer-aided design, analysis and evaluation systems in collaboration with Professor Harold Borkin, Eduardo Sobrino and Moon Sang Cho at the University of Michigan.

1) Analysis and evaluation knowledge in a database

As the design stage proceeds, analytical elements for the design concept and the criteria to evaluate alternative design schemes become detailed and complex. The number of entities describing the building model increases. Properties of entities from other sources are necessary for integrated analysis and evaluation. New entities result from a different view of existing relationships, while others are established from a different perspective. This leads to a conclusion that basic knowledge about the design concept, analytical elements and evaluation parameters should be handled as entity, and the FRAME structure[1,2] is appropriate to represent this type of knowledge. Organization of this knowledge facilitates expectation-driven processing, looking for things that are expected based on the context one think one is in. The representational mechanism supplies a
PLACE FOR KNOWLEDGE, AND THUS CREATES THE POSSIBILITY OF MISSING OR INCOMPLETELY SPECIFIED KNOWLEDGE.
IMPLEMENTATION OF KNOWLEDGE REPRESENTED IN FRAME IS SHOWN IN THE NEXT SECTION.

2) DESIGN RULES

IT IS NOT THE GOAL OF THIS STUDY TO COLLECT KNOWLEDGE AS LEGITIMATE RULES IN DESIGN. RATHER, THIS STUDY TRIED TO BUILD A CAAD ENVIRONMENT WHERE THE USER CAN COLLECT AND UTILIZE HIS OWN RULES. BUILDING REGULATIONS AND STANDARDS ARE RULES EVERYONE MUST COMPLY WITH IN DESIGNING BUILDINGS, AND THEY ARE TAKEN HERE FOR THE TEST OF THE PROPOSED MODEL. SINCE BUILDING STANDARDS ARE IMPOSED AS PERFORMANCE CRITERIA, THEY CANNOT BE DESCRIBED IN A STATIC FORMAT. THEY ARE INDEPENDENT FROM PROJECTS. THE BODY OF RULES MUST BE SEPARATED FROM KNOWLEDGE ABOUT BUILDING OBJECTS AND THE DESIGN CONCEPT, AND BE EXPLICITLY WRITTEN FOR EXAMINATION AND MODIFICATION. THE PRODUCTION RULES[3] ARE USED FOR THE REPRESENTATION OF RULES. RULES AND THE CONTROL STRUCTURE ARE EXPLAINED IN A LATER SECTION.

BUILDING MODEL DATABASE

BUILDING OBJECTS AND RELATIONS BETWEEN THEM ARE THE COMMON NOTATION OF BUILDING DESCRIPTION. OBJECTS CAN BE CLASSIFIED INTO TWO TYPES. ONE IS A GENERIC TYPE OF CONSTRUCTION OBJECTS (WALLS, OPENINGS, ETC) AND SPACES (ROOMS), AND IS DEFINED AS OBJECT HERE. THE SECOND IS INSTANCEs OF THIS OBJECT FOR A PARTICULAR PROJECT. OBJECTs DEFINE THE GENERIC CONCEPT FOR OBJECTS, WHILE INSTANCEs HAVE MORE SPECIFIED INFORMATION. THE BUILDING MODEL DATABASE USES A FRAME-BASED KNOWLEDGE REPRESENTATION TO ESTABLISH A PROPERTY INHERITANCE HIERARCHY AMONG OBJECTs AND INSTANCEs. THE FRAMES STRUCTURE IS BASED ON A COMBINATION OF FEATURES FROM KRL (Knowledge Representation Language), SRL (Schema Representation Language), AND FRL (Frame Representation Language). THE PROLOG IMPLEMENTATION OF THE FRAME UTILITIES IS BASED ON THE SIMPLE COMPUTER-VISION SYSTEM[4]. FRAMES HAVE SLOTS, FACETS, AND VALUES. THE IMPLEMENTATION IN PROLOG IS TO LET EACH SLOT-FACET-VALUE TRIPLET BE REPRESENTED AS A PREDICATE Whose HEAD IS THE FRAME NAME. THE STRUCTURE IS SIMILAR TO THAT OF OBJECT-ATTRIBUTE-VALUE. However, THE FACET ALLOWS DEFAULT VALUES FOR OBJECTs. AND IF-NEEDED SLOT CONTAINS AN ATTACHED PROCEDURE (DEMON) THAT CAN BE USED TO DETERMINE THE SLOT'S VALUE IF NECESSARY. DEMONS ADD KNOWLEDGE TO A SYSTEM WITHOUT SPECIFICATION OF WHERE IT WILL BE USED.

1) OBJECT FRAME

FACTS ABOUT OBJECTs AND BASIC KNOWLEDGE ABOUT RELATIONSHIPS BETWEEN OBJECTs ARE DEFINED IN THE OBJECT FRAME. RELATIONSHIPS LIKE PART-OF, ELEMENT-OF ARE BASIC AND COMMON KNOWLEDGE IN THIS FRAME. PARTICULAR EVALUATION PARAMETERS ARE ALSO DEFINED IN THIS FRAME. REQUIREMENTS FOR OBJECT EVALUATION PARAMETERS ARE SIMPLE EXAMPLES OF EVALUATION CRITERIA STORED IN THE SLOT (NAMED REQUIREMENT) AND GIVEN EXPLICIT VALUES. FIGURE 1 SHOWS AN EXAMPLE OF SUCH A FRAME.

205
Figure 1. Example of an object frame

Object frames contain knowledge about the evaluation aspect, mainly as procedural or logical methods to get necessary information for the evaluation. Knowledge is in many cases just a definition of the relationship between the OBJECT and its property. Figure 2 shows an example of how knowledge about a space and the activities accommodated in it can be represented. This is usually not a one to one relationship.

Figure 2. Example of an object frame

<table>
<thead>
<tr>
<th>Space</th>
<th>Area requirement</th>
<th>Value: 3000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Adjacent-to</td>
<td>Value: Auditorium</td>
</tr>
<tr>
<td></td>
<td>Activity List</td>
<td>Value: Restroom, Lobby, ...</td>
</tr>
</tbody>
</table>

2) Instance frame

Instance frames of a project's database have different entries from the building sketching program database according to the subject of evaluation. An instance frame has `isa` (meaning "is a") relationship with its object frame of generic type, and has also `also` (meaning "a kind of") relationship with its object frame of evaluation aspects so that an instance frame can inherit necessary information from its object frame. Any fact asserted about objects can be considered assertions about instances also. The slots of the Room1 frame in the figure 3—inherit from the Room frame in the figure 1—are fully specified.

Figure 3. Example of an instance frame

Room1

<table>
<thead>
<tr>
<th><code>isa</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Activity</code></td>
</tr>
<tr>
<td><code>Value</code>: Office</td>
</tr>
<tr>
<td><code>Part-of</code></td>
</tr>
<tr>
<td><code>Value</code>: Floor(1)</td>
</tr>
</tbody>
</table>

3) Project dependent database

A PLAN database, created through the PLAN sketch program developed by the Architecture and Planning Research Laboratory at the University of Michigan, contains all the graphical information as well as the design description of a project. The PLAN database is a relational database of building components which are defined as geometric models with specific properties. It is converted into an instance frame database. The conversion program is written in FORTRAN so that the program generates the predicates for instance frames into Prolog database as shown in Figure 4.

4) Knowledge representation on frame

Besides design information other knowledge is required to evaluate a suggested design, and it is stored in the object frame. As Figure 2 shows, the designer might
need knowledge about more general space concepts to check the space area requirements requested by customers. All spaces generated in the PLAN sketch program are defined so specifically that it is necessary to define which spaces are considered as a required space. As mentioned earlier, the database generated by the sketching program can only provide data which are predefined as entities in the database.

5) Frame control structure

The control structure of frames is used to obtain an associative value of a certain instance for a certain attribute. One of frame utilities is shown in the List 1. This value may be transferred from the PLAN database. Otherwise, the object (parent) frame will be referred to get either a default value for the attribute, or call procedures to compute the values. As mentioned in the previous section, there are two relationships which allow the control structure to move up the hierarchy to get an undefined value from the PLAN database. One is isa relationship, and the other is a.k.a. They are different in that they have different concepts about paarenthood for its child, but they are same in the way the control structure treats them. The control structure replaces, removes old values, and appends new values to them.

Representation of rules

This system evaluates a project according to the evaluation rules the user provides. Rules are represented as PRODUCTION RULES. In order to manipulate the representation and execution of rules on the system, a data model is necessary. This model allows rules to be checked, corrected, and updated easily. A rule-based inference engine is built on this model.

1) Data model for rules

A rule has the form: If A, B, C then D of a conclusion of D and one or more conditions A, B, C. Rules contain constants as well as variables to stand for arbitrary individuals (instances). The rules of building code specify the OBJECT's value to be matched for compliance. Therefore, the clause (A, B, C, or D) should select instances for a specified object, obtain their values from the database, and match them with those from the code. In a general form, rules look like this:

Rule 100
If Object-1's Attribute-1 value is in Relationship-1 with Value-1, Then Object-2's Attribute-2 value must be in Relationship-2 with Value-2.

Figure 5. Production Rule

Rule representation is manipulated by using predicates[2] as follows:
IF(ID, Oi, Ai, Pi, Vj)
THEN(ID, Oj, Aj, Pj, Vj)

where ID is Identification number
O is Object
A is Attribute
P is numerical/logical operator
V is Predefined value.

Figure 6. Rule model using predicates

Rule representation and its interpretation is shown below:

if(809, floor, greater, occupant_load, 100),
then(809, floor, greater, exit_number, 2).

Rule 809
If occupancy load of a floor is greater than 100,
then the number of exit on the floor must be
greater than 2.

Figure 7. Rule example

2) Inference engine

The inference engine controls the selection of the
requested rule, and checks whether the condition (IF
Clause) is satisfied for the project. If not, it tells the user
that the conditions has not matched. If satisfied, it checks
whether the Requirement (THEN Clause) is satisfied in
the project. Following is the algorithmic procedure, and a
portion of Prolog program is seen in the List 2 to show
how the inference engine is implemented.

1. Get the rule for check request. Find IF and THEN
predicates by the Rule Identification number ID the user
provides. There is a utility program that finds a rule when
provided with attribute and/or object name.
2. Get instances of the object O in the rule by
consulting frame utilities.
3. Get the value of each instance for the attribute A by
consulting frame utilities. If the attribute has not been
defined, consult the user, get the information, and put it in
the knowledge base.
4. Check this value with criteria value V if they satisfy
the relationship of Operator P if met unsatisfied
condition, go to step 6.
5. go back to step 3 while there is any instance of the
object.
6. Inform the result to users.

System Structure

Figure 8 is a diagram of the system structure.

This system was implemented in Prolog on an Apollo
computer. A user interacts with the system by specifying
a project name that the user wishes to have the system
evaluate. The building model has been input through the
PLAN sketch software developed by the Architecture and
Planning Research Laboratory at the University of
Michigan. Both graphical and non-graphical information
about the building model are stored in the project database.

After the building model has been produced
through the PLAN sketch software, the low-level process
builds a project database into the frame structure so that
the system can read them in as predicates. When the
system reads in the project database through the high-level
process, it generates more information into the frame
structure, that is, to produce the object frame database
based on the instance frame database. For example, the
system generates a slot (named INSTANCE-LIST) of an
object frame which has the list of instance names for
quick search while executing rules over the object. From
here on, the system works with frames and rules
exclusively, and evaluates a project based on the design
knowledge and evaluation rules the user provides.

All system interactions are entirely controlled by
the user interface. It has a set of top-level routines that
schedule other processes in the system to be executed.
Whenever a process in the system finishes executing, it
passes control back to these top-level routines in the user
interface. The user interface also provides a number of
generic routines that are available to the rest of the
system.

The core of the frame utilities of the system has
been already described in the previous section. The
routines provided by this component are also generally
available to the rest of the system, and can be considered
as another higher-level language (design concept) on top
of Prolog which is used for system implementation.

Evaluation rules are handled by the inference
database which in turn refers to the frame database through
frame utilities to get the associative values for instances
of the object and its attributes appearing in the rule. As
shown in the system structure, it may need to refer to the
other database for a particular evaluation. These databases
are created by the Groton Database System[5], and
interface is provided in the Prolog interpreter.

Finally, Apollo workstations allow multi-
window processing which makes immediate feedback of
design and evaluation possible. Once the design is done,
the user can evoke the system on one window with the
plan on another window. The design will be evaluated
according to the criteria and rules, and the results will be
reported to the user. It is possible to look at both building
plans and evaluation results simultaneously, and make
changes on the plans if needed. The system then converts
the design information into frame structure, and the new
cycle of evaluation will start.

Analysis and evaluation aspects

This system is designed to allow the designer to
create any design concept, and evaluate it against his
design theoretically, if not practically. However, the
knowledge engineer may have to interpret the concept into
a procedure (demo) using the frame database and utility
facility, and this knowledge about design and evaluation needs to be added into knowledge base. In other words, the user does not need to know how to program the procedure to get the information about design concept, but does need to provide the necessary method for the procedure. The title of knowledge—common terminology among design professionals—will be stored in the value of the frame, while the real procedure to gain information about the knowledge will be stored in the supporting knowledge base.

As mentioned in the last section, different types of data for various evaluation aspects are stored in the database(s), and the system can retrieve necessary information from an associative database. For example, one database contains all the information about the unit costs of construction systems for cost estimation. The user can add up supplementary databases for new analysis and evaluation later on.

1) Area requirement

The system evaluates the space area requirements (Figure 9) according to suggested area requirement by calculating the percentage of actual area to required area. While the system aggregates the activities for the space that are defined in the knowledge base, the appearance of given activity types is checked.

2) Adjacency

Adjacency among activity types is evaluated. If two activities are connected through an opening or open spaces without any construction objects, two activities are claimed as adjacent to each other.

By rule, library should be adjacent to education.
By rule, offices should be adjacent to shipping_receiving.
**Offices is not adjacent to shipping_receiving.

3) Visual Accessibility

Any two rooms are visually accessible if two rooms are adjacent to each other, or two rooms have any common adjacent room (space) without walls blocking visual accessibility from opening to opening. The algorithm is as follows:
1. If room-1 and room-2 are adjacent, report success and exit. Otherwise, continue.
2. Find adjacent spaces to room-1, and store in spaces-1.
3. Find adjacent spaces to room-2, and store in spaces-2.
5. Get the first space from common-spaces, and delete the space from common-spaces.
6. Define a line between openings of two rooms, one connects room-1 and the space, the other connects room-2 and the space respectively.
7. Test if there is any intersection between walls of the space and the line defined in step 6. If there is an intersection, go to step 5. Otherwise, continue.
8. Test if there is any intersection between walls of room-1 and room-2, and the line defined in step 6. If there is no intersection, report success and exit. If there is, go to step 5.

where \( N_{ab} \) = Nearness of room_a and room_b
\( CD_{ab} \) = Distance between centroid of room_a and centroid of room_b
\( CD_i \) = Distance between centroid of room_i and floor centroid
\( n \) = Number of rooms on a floor and represented in percentage.

What is the starting room name?
| grant_wood_gallery |
What is the destination room name?
| marvin_cone_gallery |
Nearness of two rooms is 0.513 %.

Figure 12. Nearness evaluation

5) Walking distance

The walking distance between two rooms on the same floor is calculated.

What is the room where you stand?
| restaurant |
What is the room you go?
| library |
45.73 ft.

Figure 13. Walking distance evaluation

6) Cost estimation

The building cost of a project is calculated using square foot method based on MEANS(6) book, and compared with the desirable cost. The unit prices of construction systems are stored in a database, and by providing the method to calculate the total construction
cost the system can get very precise cost estimation.

<table>
<thead>
<tr>
<th>Project Museum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desirable cost: 4500000</td>
</tr>
<tr>
<td>Actual cost: 6449270</td>
</tr>
<tr>
<td>Percentage: 143.32 %</td>
</tr>
</tbody>
</table>

Figure 14. Cost evaluation

7) Regulation checking

The number of exit on floors for a building is checked by referring to the Means of Egress Sections in the BOCA Basic Building Code Book[7].

<table>
<thead>
<tr>
<th>?- check_rule(809).</th>
</tr>
</thead>
<tbody>
<tr>
<td>... Now checking the IF Clause(floor, greater, occupant_load, 100)</td>
</tr>
<tr>
<td>... Now checking the THEN Clause(floor, greater, exit_number, 2)</td>
</tr>
</tbody>
</table>

Your design is complying with this rule.

Figure 15. Regulation checking

Evaluation of the Model

1) Rules as data

Rules are interpreted into first-order predicates. Predicates are easy to understand and manipulate, and therefore examination and modification of rules can be done without difficulty only if the rule model is known.

2) Design knowledge in a database

As explained in previous sections, evaluation parameters or analysis elements are defined in the OBJECT frame, and are treated as entities. Any design concept that is formulated by manipulating relationships of objects can be also entered as entities.

3) Integration into the design process

The design process is iterative, and evaluation and feedback is essential in the design process framework. This model will meet this need, since it allows the user to evaluate his design immediately, and to change the design according to evaluated results interactively. Besides, the rules can be used as suggestions for the future course of design development.

4) Testing the system within the CAAD studio

The CAAD studio for the second year students of master program was initiated at the University of Michigan for the complete design process. The model was tested by the voluntary students who participated in the attempted CAAD studio, and it proved to be quite advantageous to have a quick evaluation at hand. Space area requirements was the most important aspect of design evaluation because they were explicitly documented as design criteria, and information about area as requested in the criteria could not be acquired during the schematic design stage. Other aspects that I provided did not seem to be of much help for reasons of time limit for the jury, less concern for the evaluation, and lack of enthusiasm.

5) Future expansion and research direction

Rules for synthesis, analysis and evaluation of design information, which we think intuitive and
experimental, must be developed. A prototype for this model would collect primitive rules, test them against various design schemes, and the refined rule body would be established as the base for design rules.

Knowledge about design must be transmitted from one knowledge-based system to another so that no efforts for generating identical knowledge bases are duplicated. The standardization is to make knowledge interchange possible. The idea is to create standard vocabularies and ways of defining things in terms of primitives.

The interface between sketching building models and generating its database must be improved in such a way that changes in design be appeared in the database, and vice versa. At present, only one way of change is available from design to database. Designers should be able to work on the database which will modify their building model in certain domains, even if it is not possible to change the space design and layouts.

References

Bibliography
6. Davis, Randolf, Douglas R. Lucea. Knowledge-Based
List 1.

/* Get the value of Slot in a given frame. */
frame_get(Frame, Slot, Value) :-
  fget(Frame, Frame, Slot, Value).

/* Check for a value Facet. */
ffget(Parameter_Frame, Frame, Slot, Value) :-
  fget(Frame, Slot, value, Value).

/* Check for a default Facet. */
ffget(Parameter_Frame, Frame, Slot, Value) :-
  fget(Frame, Slot, default, Value).

/* Look for a demon. */
ffget(Parameter_Frame, Frame, Slot, Value) :-
  fget(Frame, Slot, if_needed, Rule),
  F =.. [Rule, Parameter_Frame, Value],
  F.

/* Move up the hierarchy. */
ffget(Parameter_Frame, Frame, Slot, Value) :-
  fget(Frame, isa, value, Parent),
  ffget(Parameter_Frame, Parent, Slot, Value).

ffget(Parameter_Frame, Frame, Slot, Value) :-
  fget(Frame, ako, value, Parent),
  ffget(Parameter_Frame, Parent, Slot, Value).

/* Just grab the given Facet or fail. */
ffget(Frame, Slot, Facet, Value) :-
  F =.. [Frame, Slot, Facet, Value],
  F.

List 2.

/* Check ff-clause. If fail, exit. If succeed, return success. */
check_if(Code) :-
  check_factlist(Code),
  check_if(Code).

check_if(Code) :-
  if(Code, Obj, Pred, Attr, Val),
  check_clause(Obj, Pred, Attr, Val),
  add_factlist(Obj, Pred, Attr, Val),
  check_if(Code).

/*Find ff-clause in the fact list. */
check_factlist(Code) :-
  if(Code, Obj, Pred, Attr, Val),
  fact_list(List),
  member((Obj, Pred, Attr, Val), List).

/*Check the rule finding all instances of specified object. */
check_clause(Obj, Pred, Attr, Val) :-
  find_obj(Obj, List),
  c_instance(List, Pred, Attr, Val).

/*Check the rule for each instance. */

c_instance([I, .., I]) :-.

check_instance(Instance, Pred, Attr, Val) :-
  check_instance(Instance, Pred, Attr, Val),
  c_instance(I, Pred, Attr, Val).

/*Check the instance. */

check_instance(Instance, Pred, Attr, Val) :-
  frame_get(Instance, Attr, Value),
  check_pred(Pred, Val, Value).

/* Check two values from Building Codes and building model database. */

check_pred(Pred, Val, Value) :-
  T =.. [Pred, Val, Value],
  T, 1.
Glossary

Database
A computer-based recordkeeping system. Several distinct data files are unified, with any redundancy among those files partially or wholly eliminated. The individual pieces of data in the database may be shared among several different users, in the sense that each of those users may have access to the same piece of data.

Entity
Any distinguishable object that is to be represented in the database.

Field
The smallest named unit of data stored in the database.

Frame
Knowledge representation scheme about the objects and events typical to specific situations.

Inference engine
A program which controls the knowledge-based system's activity to infer conclusions from the input data.

Knowledge representation
A combination of data structures and interpretive procedures that, if used in the right way in a program, will lead to "knowledgeable" behavior.

Predicates
Statements about individuals, both by themselves and in relation to other individuals.

Production rule
A statement cast in the form "If this condition holds, then this action is appropriate."

Prolog
A general purpose language based on first-order predicate logic.

Relational database
A database for which the operators available to the user are ones that operate on relational structures.