Abstract

In the last decade we assisted at an evolution of computer aided design systems from drafting, calculation and simulation utilities toward systems able to support the conceptual phase of the design process. Systems supporting conceptual design use knowledge about the design domain and assume a well defined model of the design activity. Their computational framework is usually built by means of a set of representational schemata which lack a formal semantics. This aspect causes a limitation on the applicability of their computational framework to different domains. In this paper we propose a formal knowledge representation language, that has been defined in order to represent the structural relationships of domain knowledge. On the basis of the language structure we propose a number of inferences tailored to case-based conceptual design aiding. Finally we apply the representational framework to the implementation of a computational architecture for conceptual design aiding that integrates multimedia representation of design cases with symbolic information processing. The architecture combines a knowledge representation server and a multimedia server. The knowledge representation server processes both domain knowledge and design experiences according to the Case Based Reasoning paradigm. The multimedia server produces the required case representation.

1- Introduction

In the last decade we assisted at an evolution of computer aided design systems from drafting, calculation and simulation utilities toward systems able to support the design processes conceptual phase (Eastman 91, Gero 93). Systems supporting some aspects of conceptual design have been built according to different paradigms (Chandrasekaran 90, Gero 90). Most of them use knowledge about design domain and assume a well defined model of the design activity. An important distinction can be drawn between autonomous design problem solvers and aiding systems. Systems belonging to the first category (Hammond 89, Kolodner et al. 88) view design activity as a problem solving process and, according to different perspectives, they model the whole design process by means of a computational paradigm. Their distinguishing feature is that they have a rather well defined world model (e.g. functional model of devices). Aiding systems (Barber et. al. 92, Domeshek et al. 92) are usually more suitable when the world model is not completely defined (e.g. when the world complexity is so high that it is quite impossible to define a model). Usually in this case some partial world models exist, but they are not related and a global view of the domain does not exists. For example this is the case of architectural design where a number of partial models exist (e.g. functional, geometric, thermal, dynamic, etc.) but where there is no possibility of defining what is a good project basing the judgement only on those partial descriptions (e.g. because of the aesthetic aspect of the artefact). Nevertheless aiding systems have their own problem solving ability too. It is usually applied to limited well understood domains, but the validation of the solution and the whole problem solving control is deferred to the user.

Given the many different kinds of knowledge usually involved in a design process (e.g. geometric knowledge, functional and/or behavioural model of devices, past design experiences, design process knowledge like histories, failures, etc.), a major issue in the construction of aiding systems have been that of the representation and the integration of
knowledge of different nature. In designing, designers unconsciously use many different kinds of knowledge propagating effects of inferences done in one domain to others. For example, during the definition of a house lay-out that results in a pure geometric disposition of building elements, an architectural designer frequently makes inferences about lighting and/or thermal aspects, and propagates the effects to the geometry (mostly like additional constraints). A different example of knowledge integration is the designer ability to recall pieces of old successful design cases and insert them into a new context according to considerations performed in different domains. Thus knowledge integration is a desired feature for computer aided conceptual design.

Knowledge integration necessary requires a representation schema able to describe uniformly all the processed knowledge. An issue arises when unstructured information has to be merged with structured knowledge. This is specially the case of aiding systems that integrate multimedia representations of design cases with structured domain knowledge. Multimedia representations of design cases are design representations that carry a great quantity of unstructured informations. They are well suited for suggestions and design descriptions but they are scarcely useful for design critics. In fact in a conceptual design aiding system unstructured information is not easily separable into relevant pieces for the problem to be solved, while using the whole information can be misleading.

In this paper we propose a system able to integrate pieces of structured and unstructured knowledge. Knowledge integration is achieved by mapping a conceptual knowledge representation schema, expressive enough to depict every relevant domain aspect, into a set of design case multimedia representations. Multimedia representations inherit part of the structure of the conceptual knowledge representation schema and can be accessed by the inference apparatus offered by the conceptual schema.

The whole system can be view as a case based design aiding system. Case Based Design Aiding (Domeshek et al. 92) is a general CAD paradigm based on the notion that the past design experience plays an important role for the successful generation of new artefacts, especially when the problem domain is not completely understood. Experience provides the user the appropriate guidelines in order to avoid mistakes previously done, to find out solution more efficiently and to exemplify on the field solutions (if they exist in the database). The main processing cycle of a case based design system can be summarized in the following phases: problem assessment, key definition, case retrieval, case adaptation, solution evaluation, solution storing, or, in case of failure, failure explanation, failure storing and solution repair. A complete explanation of the above phases and of the issues involved can be found in (Kolodner 93).

In section two we give an overview of the system architecture. In section three and four we explain the main system modules. Finally in section five we will examine the application of the system to design support.

2 - System architecture

The system we propose is made up of a Knowledge Representation Server, that manages both the conceptual representation schemata and the inferences the system performs, and of a Multimedia Server that manages the multimedia representation of previous design solutions. The system architecture is shown in figure 2.1.

The knowledge base contains general knowledge about the domain. Domain knowledge is represented as a set of descriptive schemata of domain elements and of conceptualizations over them. Schemata have a well defined structure organized according to a generalization-specialization hierarchy. A number of inference procedures produce conclusions according to the contents of the knowledge base. The knowledge base structure is the basic mean for the organization of cases in the Case Base. The case base is the persistent storage memory, it contains multimedia representations of design cases and is maintained by a Multimedia Server. A multimedia representation consists of a video made up of a sequence of frames and of an audio track. A frame sequence contained in a video is multiply indexed by means of the knowledge base schemata: schemata have links pointing to the frame sequence that are exemplifications of the features they describe. In this way descriptive schemata can be used as keys for the retrieval of video sequences. This feature allows the system to retrieve video sequences that have an high probability to be relevant for the problem the system is working on. Furthermore the indexing links follow the semantics of the structured knowledge representation schemata. The interface module organizes the system input-output; it has windows for alpha-numerical data and windows for multimedia presentation of videos.
3 - The Knowledge Representation Server

The symbolic reasoning ability of the system has been built as processing layers based on a knowledge representation kernel that provides some basic representational and processing functionalities. The idea is that of pieces of knowledge organized according to generalization-specialization and difference-analogy relationships. The knowledge representation schema follows the line of structured knowledge representation (i.e. semantic networks and frames), in particular it reflects many aspects of terminological systems (Rich 91). The domain independent representation schema is used to represent both domain object knowledge and design knowledge at different abstraction levels. Knowledge is represented by means of a language with a well defined structure and expressive power. The structure of the knowledge representation is used to derive generalization-specialization relationships syntactically.

3.1 - The knowledge representation schema

One of the main points of Brachman’s seminal paper (Brachman 79) is that it is possible to define an epistemological knowledge representation level which can be suitably used to capture the structure of the knowledge organization. The KL-ONE system (Brachman et. al. 85 ) is the first one embracing this assumption. On the KL-ONE line, the basic element for representing knowledge is called concept. Concept can be viewed as a representation schema with its own internal structure. A concept is defined by means of other concepts by specifying the roles the component concepts play in the definition, and the restrictions they have to satisfy.
A concept example. A typical room can be described by a concept built by at least four walls and at least by one door. The concept language expression is on the right.

Figure 3.1 - A concept example. A typical room is described by a concept built by a sides role of type wall which must contain at least four elements, by a doors role which must contain one element, and by a windows role which could be empty. The notation on the right is the concept language equivalent description (figure 3.1 on the right). Roughly speaking, the roles are defined by the ALL operator, while the restrictions on the role cardinality by the ATLEAST operator. The whole concept expression is then conjoined by the AND operator. The syntax and semantics of the concept language is drawn in appendix A.

Furthermore concepts have instances, that are individuals that belongs to the class denoted by the concept. Instances inherit the structure of the concept they belong to. For example in figure 3.2 the instance k of the kitchen concept is made up of instance a,b,c,d of concept wall and by the door e and the window f. A concept description, like the one in figure 3.1, can express necessary and sufficient conditions or only necessary condition for the inclusion of an individual in the class denoted by the concept. In the former case the concept is said to be derived in the later to be primitive.

3.2 - Subsumption

Subsumption is the main reasoning service offered by a concept based systems. Subsumption captures the generalization-specialization relationship that can be stated between two concepts. By definition, a concept C1 subsumes a concept C2 if the set of individuals denoted by C2 is included in the set denoted by C1. One of the main features of concept languages is that they computes the subsumption relationship (that is defined semantically) using the structure of the concept definition (that is essentially syntactic). The computational complexity of the subsumption algorithm depends on the expressive power of the defined language. An overview is given in (Schaerf 94). An example of concept hierarchy is given in figure 3.2.
4 - Multimedia representation server

The second main module of the system is the multimedia manager. This module maintains a data base of videos representing design cases. A video is represented as a sequence of frames synchronized with an audio track. Every sub-sequence of a video can be accessed and retrieved by the multimedia server. We call a video sub-sequence an aspect.

4.1 - Episodic knowledge: linking structured and unstructured data

As cases are episodes they are instantiated knowledge. In our system, cases are sets of concept instances that have a link with aspects of the same video in the multimedia database. At present, the linkage is defined by the user, which previewing every video that has to be inserted into the case base, generates the opportune concept instances and links them to the related aspects. Once this memory image is generated it can be stored in a long term memory according to the knowledge structure of the conceptual memory. Every concept in the conceptual memory corresponds to a table in the case base which contains data (i.e. instance role values) and logical links to other table rows (i.e. roles). Every table can be recalled by accessing the related concept which the table is an image of. A table row can be linked to an aspect. If the instance stored into the row is recalled by some inference procedure, the related aspect can be processed by the multimedia manager. During processing, cases can be recalled by specifying significant searching keys. The content of a key can be any piece of knowledge episodically related to a case (e.g. personal experience, external unrelated events, etc.). The system does not make any distinction between the knowledge used as a key and the knowledge used to describe the case. In fact, keys are represented like any other knowledge piece (i.e. using the same language and storing it in the long term memory as a part of the case). The implication of this point is that keys have a structure which can be used by the same algorithm (i.e. subsumption) the
system uses to reason with data.

Fig. 4.1 - Long term memory organization - Cases are stored as set of logically linked tables. Every row can be linked to a video sequence.

4.2 - Searching and retrieving aspects

The main task of the multimedia server is the retrieval of aspects that contain multimedia description of a given instance. While the retrieval of a linked aspect of a given instance is straightforward, this is not the case when the instance is not explicitly linked to any aspect. In fact, usually not every instance is linked to an aspect, rather only high level conceptual features are indexed during the preview phase of a video. The reason of this limitation is simply a practical economy of time in the construction of a case base. Moreover the system can use the structure of the knowledge base to form the hypothesis that the aspect of a part is contained into the aspect of the whole. Given the concept...
structure of figure 3.1, for example, it is likely that aspects linked to instances of the concept room contain aspects related to its windows and to its doors. Thus, even if an instance a of a concept has not any aspect directly linked in the case base, the multimedia manager searches for aspects linked to instances that have α as a component in their conceptual structure. Furthermore the probability of finding a good representation of a component into an aspect that is related to a compound object that use the component decreases with the structural distance (i.e. the number of role relationships the manager has to travel to find a compound object with a linked aspect) between the component and the compound object.

5 - Design support

So far we have briefly seen only the basic structure and inference services provided by the system. These services are the fundamental means for the construction of more elaborated inferences for design support. The main service provided by a case based design system is the retrieval of past design cases that are remarkable for the current design problem. A remarkable case is the one that proposes partial solutions, anticipates potential problems or is useful for the interpretation of a current design. In a case based design system the problem of retrieving remarkable cases corresponds to the problem of the definition of a good indexing schema for cases in the case base, and to the problem of the construction of efficient algorithms for case retrieval.

5.1 - Indexing design cases

Indexing design case is a key problem in a case based design aiding system. A good definition of an indexing schema substantially improve the rate of relevant case retrieval. The actual implementation of our system follows a well known approach to design case indexing (Domeshek 94) in which design cases are indexed according to selective presentations of design data called stories. A story is a selection of design data that represents a lesson to teach. Stories can be organized according to guidelines that are very general design rules. More details about the indexing schema can be found in (Domeshek 92). What we want to point out here is that the application of a formal knowledge representation language does not make any commitment to the information represented. The eventual limitations could come only from limitations of the language expressive power. Thus the case retrieval procedure we propose can be applied to every indexing schema that can be represented by means of the formal language.

5.2- Case retrieval

One of the most important operations of a case based design aided system is the retrieval of a useful piece of situated knowledge. Case retrieval is usually performed, following the classical data-base paradigm (i.e. by specifying a searching key and collecting cases that are related in some way to the key). Moreover designers sometime use a different strategy for the query of the knowledge base: rather than specifying from scratch the features the retrieved case should manifest, designers start from an existing case and specify which set of features should be maintained and which should vary. Furthermore variations are usually defined by denoting the strategy more than the final result. Typical query examples are: "Increase lighting", "Get a more comfortable living room", etc. The retrieval schema we are going to introduce supports both kind of case base query.

5.2.1- Feature driven search

The specification of a searching key in our system is accomplished by the same technique used to describe concepts. In fact a searching key is basically a partial description of the desired case, so it can be viewed as a concept. The subsumption procedure is then used to find out the set of subsumed concepts (i.e. more specific descriptions) and in turn the case set they exemplify. As an example the query key of figure 5.1(a) requests a flat with a kitchen and at least two bedrooms. The set of subsumed concepts has more specialized members like that of figure 5.1(b) where there are specifications about dining/living room, bathroom, etc.
The set of retrieved cases contains, among others, all the cases having instances of the concepts of figure 5.1(b). If a retrieved instance has an explicitly linked aspect, the video sub-sequence is shown in an interface window. A noteworthy feature coming from the usage of concept language key description is the possibility of retrieving a piece of a case without any constraint produced from the case storing granularity. The only requirement is that the piece can be addressed by means of the concept language.

5.2.2- Case Base navigation

In designing we sometimes find very natural to express retrieval requirements like desired variations of a given situation. In a case based design system a query like that produces a goal driven navigation through the set of known cases. In order to perform case base navigation, the system needs an internal representation of the differences between two case features. Since cases are represented according to multiple concepts, there are a number of difference relationships: one for every couple of concepts. This high number of relationships rises a computational burden when search is performed. Furthermore, given the target of representing only semantically grounded differences, it looks pointless to compute relationships among concepts that are semantically incomparable. These considerations allow us to introduce a further structure on the knowledge representation schema with the aim of stating the semantical affinity between two concepts. We define a concept collection as the set of concepts that share a good degree of analogy. Figure 5.2 shows a pictorial example of a concept collection stating the accessibility relationships among the rooms of flats. Their are grouped in an "accessibility collection" where differences among elements are related to differences in the graph topologies.
Every element denotes a set of cases in the case base. The idea is that when a reasoning process traverses a relationship link, the set of denoted cases changes in a known way, that is according to the meaning induced by the crossed relationship. If we consider the traversal of a link as a primitive action it is possible to define a planning language to express complex navigation strategies driven by goals. The planning language uses the operator structure to relate goals to strategies. The operator structure is shown in figure 5.3(a). The goal expresses the target of the strategy represented by the plot. The strategy is expressed by means of a language whose syntax and semantics is given in appendix B. The language is interpreted by a planner that builds the related plan and apply it to the initial set of cases.

For example suppose we have the case on the right in the case base of figure 5.2 and that we want to search for a flat with a topology more suited for a dimensional surface reduction. We can activate the goal reduce surface whose strategy can be simply expressed as: (reduce surface)

The planner produces the plan search tree of figure 5.3(b). It first builds the TOP plan node with the input strategy. Then it looks for an operator suited for solving the reduce surface goal, and finds out op#12. The strategy in op#12 says that, in order to reduce the surface, first try to delete a node in the accessibility collection. In case of failure (i.e. there is not any case in the case base related to the new concept) try one more deletion. Recursively the plan is depth first interpreted. When a primitive action is found, the relationship is traversed and the set of cases related to the image concept is returned. If the set is empty, a failure is backward propagated. In the example the execution of the plan produces the left case in the case base of figure 5.2.
**Operator:** op\#12  
**Goal:** reduce_surface  
**Plot:** (IFFail delete_node  
(SEQ delete_node delete_node))

**Operator:** op\#22  
**Goal:** delete_node  
**Plot:** (Action Rt_acc_12)

---

**Plan\_Node Top**  
---  
**Plan\_Node op\#12**  
---  
IFFail  
---  
**Plan\_Node op\#22**  
**SEQ**  
---  
**Plan\_Node op\#22**  
Action Rt_acc_12  
---  
**Plan\_Node op\#22**  
Action Rt_acc_12  
---  
Action Rt_acc_12  
---

(a)  
(b)

Figure 5.3 - Example of the operator structure and of a plan. The plot slot contains the strategy to be applied in order to satisfy the goal. The surface reduction can be accomplished by one or two applications of the primitive action "delete-node"

---

6- Conclusions

The aim of this work has been the investigation of the computational implications of some recent directions on intelligent design support applied to a set of multimedia representations of design solution. The paper has proposed a computational architecture with a structure aimed to obtain a human-computer interaction as much flexible as possible. A great effort has been sustained in the investigation of the suitability of the terminological paradigm applied in the development of a case based aiding design system. The usage of a concept language for the description of domain knowledge naturally drove us to the implementation of a distributed case representation. Cases in the system architecture are distributed as roles induce attributive/parthomic relationships among concepts and multiple conceptual representations naturally coexist in a single case. We also found quite natural to extend the general knowledge memory configuration to long term memory, so that every case-base table contains the structure of the related concept instances. This approach avoids the representational grain size problem that affects systems using highly structured case representation like MEDIATOR (Kolodner et al. 88) and CHEF (Hammond 86). In a distributed system the case is naturally represented at different grain sizes by different concepts. The subsumption procedure applied to a searching key finds out the most appropriate representation size according to what is explicitly asserted in the query. Furthermore the whole case representation can be easily updated when new conceptual descriptions are introduced into the system. Current limitations in the case representation consist on the restriction of the domain knowledge representation. The extension of the ontology to causal relationships, case outcomes and more generally to other aspects of the design process like the usage of the artefacts and the user interaction is a work in progress. In particular, the introduction of design process model will improve the performance of the system in the retrieval of useful cases because the design target descriptions can be used in the construction of the searching keys.
Another aspect investigated by the present work is the effect induced by the usage of structured knowledge for the organization of the unstructured information contained in multimedia representation (i.e. film strips) of design cases. The linkage of sub-sequence of videos (i.e. aspects) to the instance of the conceptual schema offers the possibility to retrieve the same aspect even for component of the explicitly linked concept. This feature is in fact a partial projection of the conceptual schema structure over the unstructured information of the case base.

This work has also some limitations: for simplicity its scope has been circumscribed to domain representations (e.g. architectural domain objects, rooms, flats, etc.). A future work will consist in the investigation of the application of the terminological knowledge representation schema to other aspects like design process representations and physical-causal model representations. Thus, at present, the system has not any strategy for the containment of the number of retrieved cases during searching with quite general keys. Another limitation is in the absence of any adaptation procedure without any case transformation possibility. A future work will investigate the repercussions of the usage of distributed representation in a model based adaptation process (e.g. how multiple models can be used, how to control their interactions, etc.)

References


Kolodner J: 1993 Case Based Reasoning, Morgan Kaufmann.

Appendix A - Concept Language

Syntax.

Given two disjoint alphabets of symbols: concept names, role names.
The set of concept terms and the set of role terms are inductively defined as follows:
Base: Every concept name is a concept term.
Every role name is a role term.
Step: Let C, C_1, ...,C_k be concept terms. Let R, R_1, ...,R_k be role terms and n a nonnegative integer. Then:

*Top*
*Bottom*
(And C_1, C_2, ...,C_k)
(All R C)
(AtLeast R n)
(AtMost R n)
(Exactly R n)
are concept terms.
(Chain R_1, R_2, ...,R_k)
is a role term.

Semantics.

The semantics of ASACL is an interpretation I =<D_1, .I> where the set D_1 ≠ ∅ is the domain of I, and the function .I is the interpretation function of I.
Base
Let A be a concept name. Let R be a role name. Then:
*Top* \( := \Delta^t \)

*Bottom* \( := \emptyset \)

\( A \subseteq \Delta^t \)

\( R \subseteq \Delta^f \times \Delta^f \)

Let \( C, C_1, ..., C_n \) be concept terms. Let \( R, R_1, ..., R_k \) be role terms and \( n \) a nonnegative integer. Then:

\( (\text{And } C_1, C_2, ..., C_n)^f := C_1^f \cap C_2^f \cap ... \cap C_n^f \)

\( (\text{All } R \ C)^f := \{ a \in \Delta^f | \forall b : (a, b) \in R^f \Rightarrow b \in C^f \} \)

\( (\text{AtLeast } R)^f := \{ a \in \Delta^f | \exists b : (a, b) \in R^f | |b| \geq 1 \} \)

\( (\text{AtMost } R)^f := \{ a \in \Delta^f | \exists b : (a, b) \in R^f | |b| \leq 1 \} \)

\( (\text{Exactly } R \ n)^f := \{ a \in \Delta^f | \exists b : (a, b) \in R^f | |b| = n \} \)

\( (\text{Chain } R_1, R_2, ..., R_n)^f := \{ (c_0, c_n) | \exists c_1, ..., c_{n-1} : (c_0, c_1) \in R_1^f \wedge ... \wedge (c_{n-1}, c_n) \in R_n^f \} \).

Appendix B - Planning Language

Syntax.
Given the set of concept term and two disjoint alphabets of symbols: primitive actions and goals. The set of planning expressions is inductively defined as follows:

Base:

*Every primitive action is a planning expression.*

*Every goal is a planning expression.*
Step:

Let \( P, P_1, \ldots, P_k \) be planning expressions. Let \( C \) be a concept term. Then:

\[
(\text{And } P_1 \ P_2 \ldots \ P_k) \\
(\text{Or } P_1 \ P_2 \ldots \ P_k) \\
(\text{Not } P) \\
(\text{Inclass } P \ C) \\
(\text{Seq } P_1 \ P_2 \ldots \ P_k) \\
(\text{Iffail } P_1 \ P_2 \ P_3)
\]

are planning expressions.

And, Or, Not and Inclass are called algebraic operators, Seq and Iffail are called control structures (Giunchiglia 94).

Semantics.
The semantics of ASAPL is an interpretation \( J = \langle X^J, \cdot^J \rangle \) where the set \( X^J \subseteq X^J \) is the domain of \( J \) (\( X^J \) is the set of individuals such that they are linked to elements of the database), and the function \( \cdot^J \) is the interpretation function of \( J \).

Base
Let \( A \) be an action. Let \( G \) be a goal. Then
\[
A \subseteq X^J \times X^J \\
G \subseteq X^J \times X^J
\]

Step
Let \( S_0 \subseteq X^J \) be a starting situation. Let \( P, P_1, \ldots, P_k \) be planning expressions. Let \( C \) be a concept term. Then
\[
(\text{And } P, P_0)^J := \{ (a, b) | a \in S_0 \land (a, b) \in P_1^J \cup P_2^J \land b \in \text{im}(P_1^J) \cap \text{im}(P_2^J) \}
\]
\((\text{Or } P_1 P_2)^j := \{(a, b) \mid a \in S_0 \land (a, b) \in P_1^j \cup P_2^j \land b \in \text{im}(P_1^j) \cup \text{im}(P_2^j)\}\)

\((\text{Not } P)^j := \{(a, b) \mid a \in S_0 \land (a, b) \notin P^j\}\)

\((\text{Inclass } P \ C)^j := \{(a, b) \mid a \in S_0 \land (a, b) \in P^j \land b \in C^j\}\)

\((\text{Seq } P_1 P_2 \ldots P_n)^j := \{(c_0, c_n) \mid c_0 \in S_0 \land \exists c_1, \ldots, c_{n-1}. (c_0, c_1) \in P_1^j \land \ldots \land (c_{n-1}, c_n) \in P_n^j\}\)

\((\text{Iffail } P_1 P_2 P_3)^j := \{(a, b) \mid a \in S_0 \land (a, b) \notin P_1^j \land (a, b) \in P_2^j\} \cup \{(a, b) \mid a \in S_0 \land \exists c. [(a, c) \in P_1^j \land (c, b) \in P_3^j]\}\)