CDT: A Computer-Assisted Diagramming Tool

Bharat Dave

Department of Architecture
Swiss Federal Institute of Technology
Zurich CH-8093 Switzerland

This paper describes the development of a computer-based diagramming tool (CDT) that supports incremental structuring of problem information using diagrammatic representations. Diagrams as graphic representations of symbolic propositions allow tentative reasoning and inferencing. The development of CDT has been carried out based on two observations. First, many diagrams are used to represent objects and relations between them. Second, diagrams comprise graphic symbols arranged on a plane using topological and geometric relations to denote problem relevant information. CDT responds to these needs by incorporating a number of computational ideas: graphic interface, direct manipulation, constraint representation by demonstration, and specification and satisfaction of diagram composition rules.

Keywords: tentative reasoning, incremental problem representation and exploration, diagramming.

1 Introduction

In the process of externalizing, reasoning with and exploring information in a given context, problem solvers often do not have all relevant information available simultaneously. During this phase, when a problem is being re-structured, graphic representations like diagrams are often useful. The economy and directness of expression in diagrams seem to be the prime reasons why they are so ubiquitous in many domains. Uses of graphic representations may span from initial phases in problem structuring to the final one in which all information is externalized and may take on a highly stylized and organized format. This paper describes a computer-assisted diagramming tool that is aimed at supporting incremental representation and exploration of information using diagrammatic representations.

The paper is organized as follows: Section 2 outlines the area of study, the motivational background, and a brief review of related studies; Section 3 presents implementation details of a computer-assisted diagramming tool called CDT; Section 4 illustrates the use and various features of CDT; Section 5 provides an evaluation of what is accomplished in CDT; and Section 6 outlines some extensions of the ideas developed in this work.

2 Diagrams

Graphic representations that serve information representation needs appear in many forms: diagrams, sketches, tables, matrix of relationships, and others. In order to distinguish between various graphic representations, we have adopted the distinctions proposed by Ervin (1992). Diagrams are abstract, schematic representations used to explore structural relationships among parts. Unlike maps, diagrams are usually not to scale, nor need be true to shape. Maps are representations of physical or statistical features drawn using a consistent coordinate system of reference, and allow ‘real’ inferences about dimensions and spatial relationships. Graphs are concerned with representation of statistical and quantitative data, often using matrix or network representation. Pictures are representations that are primarily concerned with impression, expression, or realism. The preceding characterizations use the inferential purposes and the degree of resemblance of a graphic with what is depicted as the key criteria by which to distinguish among various graphic representations.

Given the ubiquity of graphic representations in information presentation tasks, it might be that there are indeed benefits in using graphic representations which can only be obtained, if at all, at great costs in textual representations. Based on the comparisons of using sentential representations (which are sequential, textual propositions) with diagrammatic representations (in which each expression is stored at a locus in a plane) in a number of problem contexts, the study by Larkin and Simon (1987) derives the following conclusions:

- Diagrams can group together all information that is used together, thus avoiding large amounts of search for the elements needed to make a problem-solving inference.
- Diagrams typically use location to group information about a single element, avoiding the need to match symbolic labels.
- Diagrams automatically support a large number of perceptual inferences, which are extremely easy for humans.

These three key features of diagrams, i.e., localization, minimum labeling, and perceptual enhancement, help us understand why diagrams are useful. But simply representing a problem using a diagram does not necessarily imply efficient computation on the problem; a problem solver has to know how to take advantage of additional information that is available in a diagram. In view of the benefits of using diagrammatic representations, it is desirable to find ways of supporting such representations using computers. Additional impetus for the development of CDT derives from observing the functional characteristics of many tools and techniques developed in the area of computer aided design. For the sake of brevity, we mention only the modeling and generative systems in the following.

In many interactive modeling systems, a user builds up a digital model of a design in a stepwise fashion using the representations and operators provided in a given system. Although such modeling systems are certainly useful, they force users to commit too many details too early, a constraint which may not be appropriate, especially during early design development. There is no support in such systems for conceptual and explorative representations that we find in the traditional design development stages. In addition, with the availability of better technology, photorealistic rendering of 3D models has become widely available. An unfortunate consequence of this development is a fixation on the part of users with pixels. In other words, users quite often become too engrossed with the con-
creteness and realism of objects; a development which needs to be balanced with tools that support abstractions. Many computer supported design development tools seek to go beyond simple interactive operations and aim to provide automated techniques for non-exhaustive generation of feasible solutions to certain classes of design problems.

Table 1: Example Problem Description
Distance and Adjacency in DPS\(^a\) (Pfefferkorn, 1975)

<table>
<thead>
<tr>
<th>Distance</th>
<th>C1 must be less than 45 in (114 cm) from CT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C2 must be less than 45 in (114 cm) from CT</td>
</tr>
<tr>
<td>Adjacency</td>
<td>The back of T must be against the wall</td>
</tr>
<tr>
<td></td>
<td>The back of C1 must be against the wall</td>
</tr>
<tr>
<td></td>
<td>The back of C2 must be against the wall</td>
</tr>
<tr>
<td></td>
<td>The back of P must be against the wall</td>
</tr>
<tr>
<td></td>
<td>The back of BC must be against the wall</td>
</tr>
<tr>
<td></td>
<td>The back of BCA must be against the wall</td>
</tr>
<tr>
<td></td>
<td>The left side of C1 must be against T</td>
</tr>
<tr>
<td></td>
<td>The right side of C2 must be against T</td>
</tr>
</tbody>
</table>

a. C1, C2, T, P, BC, BCA refer to various furniture items to be placed in a room.

While these approaches are useful, they typically require that a given problem be structured substantially prior to the application of generative techniques, e.g., how many spaces are required and what are the relevant relations between them. Tables 1-2 show typical input information to some automated systems whose development is separated by a number of years. Despite the fact that newer approaches have proposed techniques with better resolution of details, we have not seen system development efforts aimed at graphically facilitating the amount of prior problem structuring expected in such systems.

Table 2: Example Problem Description
Required Spaces in LOOS and WRIGHT (Flemming et al., 1992)

<table>
<thead>
<tr>
<th>Space Name</th>
<th>Min. Dim</th>
<th>Max Dim</th>
<th>Min Area</th>
<th>Max Area(^a)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(m)</td>
<td>(m)</td>
<td>(m(^2))</td>
<td>(m(^2))</td>
</tr>
<tr>
<td>Court</td>
<td>3.60</td>
<td></td>
<td></td>
<td>22.00</td>
</tr>
<tr>
<td>Living Room</td>
<td>3.60</td>
<td>5.40</td>
<td>14.00</td>
<td>10.0</td>
</tr>
<tr>
<td>Master Bedroom</td>
<td>2.40</td>
<td>4.20</td>
<td>7.20</td>
<td>10.0</td>
</tr>
<tr>
<td>Bedroom 1</td>
<td>2.40</td>
<td>4.20</td>
<td>7.20</td>
<td>10.0</td>
</tr>
<tr>
<td>Bedroom 2</td>
<td>2.40</td>
<td>4.20</td>
<td>7.20</td>
<td>10.0</td>
</tr>
<tr>
<td>Hall</td>
<td>1.20</td>
<td>6.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kitchen</td>
<td>2.10</td>
<td>5.40</td>
<td>7.20</td>
<td></td>
</tr>
<tr>
<td>Bathroom</td>
<td>1.80</td>
<td>4.20</td>
<td>4.20</td>
<td></td>
</tr>
</tbody>
</table>

a. Maximum extent overall area from north to south, 18.0 m\(^2\); west to east, 12.0 m\(^2\).
The position advocated in this study is that design problems are not well structured to begin with and that there is clearly a need for providing computing tools that enable incremental problem re-structuring. Such a need may be addressed by supporting diagrammatic representations of design information.

2.1 Response

In order to use diagrams to represent and to communicate information, a consistent convention is needed, including a specification of graphic symbols, what information each graphic symbol is expected to convey, how one or more graphic symbols may be composed together to formulate a diagrammatic expression, and the way in which information represented in a diagrammatic expression may be interpreted. A number of studies originating in graphic communication (e.g., Bowman, 1968; Laseau, 1980; Bertin, 1983) have proposed different frameworks for studying and using diagrammatic and other graphic communication systems.

Based on these ideas, particularly those of Bertin, we have adopted a position in which diagrammatic notations are viewed as expressions defined over an alphabet of graphic marks. A set of marks may be composed using positional relationships to develop a vocabulary of forms. Forms in a vocabulary are used to generate other forms according to a defined syntax, and these compound forms may be treated as well-formed expressions in a diagrammatic notation. A diagrammatic expression becomes meaningful information only when each component of a diagrammatic notation has been associated with domain relevant interpretations.

The decision to use positional relationships between graphic marks as the major construction principle is based on the observation that, in many domains, diagrams typically represent objects and relations between them using positions of graphic symbols. We use the positional relationships to specify two kinds of composition rules: those which involve interconnection of graphic symbols, and those which involve relative positions of graphic symbols. The rules of the first kind enable representation of links or connections between graphic symbols; the rules of the second kind enable specification of relations like above, below, left-of, right-of. Additionally, in the current implementation of CDT, we have restricted ourselves to dealing with only association diagrams (those in which some objects are related to others by some relations). To enable representation of other kinds of diagrams, e.g., containment diagrams, will require addition of some computational machinery to CDT which we have planned as an extension of the work reported here.

Based on these ideas, we have developed a parsimonious diagramming system that is not tied to a particular diagramming convention but provides all the necessary functionalities to define such conventions. A user can define a diagrammatic notation in CDT by

<table>
<thead>
<tr>
<th>Adjacency</th>
<th>Min Dima</th>
<th>Adjacency</th>
<th>Min Dim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Court/S</td>
<td>7.20</td>
<td>Court/W</td>
<td>3.60</td>
</tr>
<tr>
<td>Living/S</td>
<td>3.60</td>
<td>Living/N OR court</td>
<td>3.60</td>
</tr>
<tr>
<td>Living/Hall</td>
<td>0.90</td>
<td>Living/Kitchen</td>
<td>0.90</td>
</tr>
<tr>
<td>Bedroom/Hall</td>
<td>0.90</td>
<td>Bedroom/N OR Court</td>
<td>1.20</td>
</tr>
<tr>
<td>BedroomM/N</td>
<td>3.30</td>
<td>BedroomM/W</td>
<td>3.30</td>
</tr>
<tr>
<td>Kitchen/N OR Court</td>
<td>0.90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a. Minimum length of shared boundary, in meters.

b. N, S, W refer to site boundaries towards north, south, and west.
developing problem relevant graphic motifs, assigning meaningful textual attributes to those motifs, and specifying rules of composition for graphic motifs. A diagrammatic notation thus developed can be subsequently used wherein the user concentrates on the problem relevant information, whereas CDT takes care of satisfying and maintaining notational consistency of information. A diagram constructed in this fashion is, in effect, a computable description and can be used for further processing tasks by other computing tools.

2.2 Related Works
A number of computer systems which make use of diagrammatic representations have been reported in the literature. Many of these systems are interdisciplinary in nature as well as heterogeneous in terms of representations they support. CDT has much in common with automated graph layout systems, graphic knowledge-acquisition tools and pictorial grammar based systems, but CDT also differs in certain regards from many of these systems. In particular, one of the premises for the development of CDT is that, in a given context, problem relevant information is incrementally externalized and need not be manipulated in a textual form to begin with. Thus, in contrast to the automated graph layout systems like COOL-TRIP (Kamada and Kawai, 1991), CDT does not require textual specification of all the information and it is not designed to completely automate the graphic layout. The programming support systems like BLOX (Glinert, 1990) are tied to a selected programming language and predefined graphic templates; this is necessary so the programs developed can be interpreted and executed by a computer. In this respect, CDT is not aimed at providing general programming facilities, although we would like to incorporate a narrow but specialized support of this kind. This decision is based on the recognition that representation of some kinds of information is better handled graphically, but complex algorithms are still better expressed in textual programming languages. Unlike many pictorial grammar based systems, for example Janus (Kahn and Saraswat, 1990), which predefine the syntax of graphic elements, CDT is developed with the aim of exploring ways of defining such syntax graphically at run time, to understand the problems involved and to find possible ways of resolving the problems. Finally, CDT has much in common with the graphic knowledge acquisition tools like Protege (Musen, 1990), but it is not tied to a particular domain or an application.

3 CDT: System Architecture
This section describes the framework adopted for the development of CDT, and describes selected details of how CDT represents and manipulates the information.

3.1 General Framework
Diagrams comprise a number of graphic objects arranged spatially. In order to implement CDT, two assumptions are made. First, diagrams are rendered on a two-dimensional plane in which a location is referenced by a point which is a pair of x and y coordinates. Second, although graphic objects may be described in terms of pixels as the base objects, CDT uses an alphabet of graphic symbols in which each symbol is one of the following: a point, line, curve or an alphanumeric string. From this alphabet, graphic objects are constructed that are relevant to or serve as graphic metaphors for a problem. A set of graphic objects defines the lexicon for a problem in a given domain. For certain tasks in which standardized conventions exist, it is possible to define a finite lexicon. But, similar to the natural languages in which new words may come into being, a lexicon in a diagrammatic language should be thought of as being indefinitely extensible; the limitation is
that only those kinds of new graphic objects constructed using the symbols in the alphabet may be devised.

The syntax of a diagrammatic language specifies how graphic objects may be combined to generate well-formed expressions. For association diagrams, syntactic rules define associations between graphic objects: (a) by requiring that two graphic objects “touch” each other, i.e., two graphic objects have at least one location which they both occupy, and (b) by defining constraints on the relative locations of graphic objects. It should be noted in the case of many natural and formal textual languages based on a linear representation, that concatenation of symbols and a sequential order in representation are implicitly assumed; a letter is followed by just one more letter. In contrast, diagrammatic representations afford room for greater ambiguities due to their spatial nature.

The semantics of a diagrammatic language are domain dependent. The graphic objects in a lexicon and the syntactic rules for one diagrammatic language may be used in more than one domain with different interpretations. Within the scope of this work, attaching domain specific semantics to the diagrammatic language may be accomplished in the following two ways. In a particular problem, it may be necessary to express a certain behavior for a graphic object, e.g., in an entity-relationship diagram for database schema, it may be that a graphic object denoting an attribute cannot exist unless it is associated with a graphic object denoting an entity. Such semantic notions can be expressed in CDT by defining composition rules appropriate for a particular problem. The second way in which semantics of the diagrammatic language are handled in CDT is by using declarative text strings. To summarize, a diagrammatic language may be specified using the following components: a finite alphabet of primitive symbols, a possibly infinite lexicon, syntactic rules to denote associations, and a set of semantic interpretations. These four components enable the development of a diagramming system (like CDT) which can be used to define a variety of diagrammatic languages specific to various domains.

3.2 Data Types

CDT is implemented in an object-oriented programming paradigm. All the data types used in CDT are organized as part of a hierarchy of classes of objects with appropriate methods defined for each of the classes. The characteristics and organization of classes were derived from analysis of how information in diagrams is typically represented. A diagrammatic expression comprises one or more composite objects. Each composite object, in turn, comprises either a diagram object (thus enabling creation of diagram-subdiagram hierarchies), or one of the following two object types: an entity or a relation object. Although the objects which belong to either an entity or a relation type are quite similar, we maintain a distinction between them in order to be able to apply certain operations, e.g., stretching, to the objects that are only of a relation type. Each entity or a relation object is, in turn, defined by composing together any number of defined primitive objects. The composition rules between objects are stored and satisfied by a specialized class of objects called rule objects. The complete hierarchy of classes needed in implementing these ideas in an object oriented programming paradigm is shown in Figure 1. Among these classes, domain specific information is attached with instances of the classes EntityObject, RelationObject, and RuleObject. The PrimitiveObject class serves the needs of defining graphic forms for various objects and relationships. A brief description of each of the classes implemented in CDT follows next PrimitiveObject: This is an abstract class that encapsulates behavior common to all the objects defined as being primitives in the system. Each object in this class has, in addition to its geometric definition, two significant pieces of information attached to it: (a) predefined link points and, (b) designated link points. It is as-
sumed that an object or a relationship in a diagram signifies its interaction with other objects by having at least one of its link points coincident with a link point of another object or a relationship. This assumption takes advantage of the very property of diagrammatic representations that sets them apart from textual representations, namely the spatiality of representation that enables more efficient inferences. In the current version of CDT, link points are predefined and are located on the outer geometric curve bounding an object. The PrimaryObject class comprises four subclasses: LineObject, RectangleObject, CircleObject and TextObject. Figure 2 shows instance variables of two selected classes and their link points. Each point is a variable that is assigned a value at run time, thus neither orientation nor size is predefined in the class definitions.

CompositeObject: This is an abstract class that encapsulates behavior common to its two subclasses, EntityObject and RelationObject. An instance of this class is interactively defined, and it consists of instances of PrimitiveObject subclasses. Once an instance of CompositeObject is defined, it is treated as a closed unit in the main diagram that does not have any direct access to the instances of PrimitiveObject contained in the current CompositeObject. The link points of each component PrimitiveObject are collectively treated as the predefined set of link points for the current CompositeObject. Out of this set, not all may be useful or even meaningful as link points in a given diagramming task, so CDT allows the user to designate all or only a few of the defined link points for subsequent manipulations. Thus, designated link points are analogous to external parameter lists of functions in imperative programming languages.

In keeping with the distinction between objects and relationships, instances of PrimitiveObject that are part of a RelationObject may be specified as stretchable. This is the primary and only distinction which separates EntityObject and RelationObject. This specification is used as follows: if a RelationObject is placed in reference with two EntityObjects, it may be necessary to compute values for stretchable components contained in a RelationObject to satisfy a composition rule. In such cases, only the designated components are stretched while the others remain unchanged. Using the classes described so far, graphic forms for objects and relationships may be developed. Domain specific textual descriptors may be associated with CompositeObjects at three levels. At the first level, a type label signifies that all instances of this class are similar objects. A particular instance of this type may be assigned an instance label in a diagram to distinguish it from other in-

Figure 1. Organization of classes.
stances. At the second level are link points which may be given descriptors that remain the same for all instances of one CompositeObject. This is just another way to assign meaningful named variables to link points. At the third level, CompositeObjects may be assigned a set of domain specific textual attributes for which there may not be graphic representation in the diagramming task but that are useful in a given problem context.
RuleObject: Using the two classes PrimitiveObject and CompositeObject (and their subclasses), it is possible to define a lexicon of graphic objects to be used in a diagram. This is similar to natural languages in which words and phrases may be generated out of an alphabet. In order to construct expressions that are grammatically correct and meaningful, rules of grammar and language semantics are needed. These notions are captured in CDT using the class RuleObject.

The RuleObject class enables representation of domain relevant compositions of objects in terms of (a) link points of objects that need to be coincident, and (b), relative positions of objects. As an example, Figure 4 illustrates a rule specification using the link points of two different kinds of objects for entity-relationship diagrams for a database scheme. The rule specifies that a one-to-many RelationObject has each of its link points coincident with one and only one EntityObject. Graphically this rule restricts the placement of the RelationObject in reference to two EntityObjects. Its domain specific meaning is that a one-to-many relationship is a partial function from one EntityObject to another as denoted by the directional arc.

Figure 4. Specification of a composition rule using link points of objects.
In CDT, CompositeObjects that denote directionality of some kind are disambiguated by the use of named variables during the rule specification stage and subsequently during the diagram construction and interpretation stages (Figure 5). Any number of graphic composition rules that express domain semantics may be defined this way.

On the other hand, a rule specification may involve relative positions of objects as shown in Figure 6. Such rules are defined by declaring an (in)equality relation between the coordinates of bounding boxes of two different EntityObjects. CDT supports an easy and direct manipulation style of interaction for such rule specifications in which a user drags and drops graphic forms to indicate allowable compositions. CDT displays all appropriate coordinates of bounding boxes as buttons on screen, and the selected terms are automatically placed in the equation pane of rule definition window. The composition rules are checked and satisfied during the actual process of diagram generation by the class DiagramObject.

DiagramObject: A specific diagram is an instance of class DiagramObject. Each diagram comprises instances of CompositeObject, and each of these instances complies with relevant rules, if any, encoded in the RuleObject associated with a given diagram. As stated previously, a CompositeObject is comprised of instances of PrimitiveObject or a subdiagram associated with it. Thus it is possible to generate diagrams nested within each other. Using the methods provided in the class DiagramObject, it is possible to navigate through the nested diagram hierarchy.
As implemented in the current system, a diagram may be generated in which all of its components (i.e., instances of CompositeObject and subdiagrams) interact in well-defined ways as specified by a set of rules. The base components of CDT constitute a shell in which a minimal set of methods is predefined; in a specific diagramming task, a user extends the repertoire of objects and relationships in the appropriate ways.

It is necessary to translate information represented in a diagram into formats that may be taken as input for other computing purposes. For example, an entity-relationship diagram for a database scheme may be translated in a textual format which is then taken as input for a database definition language embedded in a database management system. Thus, a diagram is, in effect, a computable description, and the diagramming system may be used as a graphic front-end for a database specification module. In the current implementation, CDT produces a textual representation that is unambiguous and from which other systems can recreate a description consistent with their needs, although it is easy to add methods which generate textual output in exactly the desired formats.

CDT is implemented in Smalltalk/V Mac (Digitalk, 1988), version 1.2, an object-oriented programming language. In the base implementation of CDT, there are no predefined conceptual objects, relations, or composition rules; only the tools which may be used to define them are provided.

4 Examples

In a given diagramming context, the following steps are needed:

Figure 6. Specification of a composition rule using relative positions of objects.
• The first task is to define domain relevant sets of entities and relationships. This step requires the following: definition of graphic forms constructed out of primitive graphic objects; designating components in a graphic object that are stretchable (only for graphic objects denoting relationships), specification of named variables that may be used as part of the graphic form and other textual attributes. CDT creates and inserts appropriate icons of objects in the main diagram interaction window.

• The next step is to define a set of composition rules that specify associations of objects and relationships that are meaningful according to the diagramming context. The rules are specified by demonstrating allowable compositions of various objects and relationships in terms of interconnections of link points and relative positions of various objects.

• Using the specifications generated in the previous two steps, a user can interactively generate a diagram by inserting desired objects and relationships. Every insertion of an object or a relationship is checked by CDT for compliance with the composition rules. At this stage, CDT expects very little effort on the part of the user. Any inserted graphic form may be moved around and given its private, instance name (which may be useful for distinguishing it from the other, similar instances of the same type label). Attributes that are not part of the graphic forms may be assigned values. A selected object or relationship may be elaborated into a more detailed information level by attaching a subdiagram. The hierarchy of diagram levels may be navigated using menu options.

• In addition to the immediate purposes of being able to represent domain relevant information in a facile way, a diagram thus generated may be used in two additional ways. First, if it is expected that similar diagramming tasks will arise in the future, a specification of the diagrammatic language created in steps 1 and 2 above may be stored in an external file, and subsequently read into CDT, thus it is possible to reuse these specifications. Second, a diagram generated interactively by a user describes specific information about a given context. The information represented in a diagram (including all its levels) may also be stored in an external file that may be used as an input for other computing tools applicable in the same problem domain.

The primary interaction with CDT occurs in the diagram interaction window (Figure 7). If a new diagrammatic notation is to be defined then two subordinate windows (shown in Figure 3 and 4) are invoked from the root diagram interaction window.

4.1 Space Layout
This section illustrates the use of CDT in a typical space layout task. This example shows how a tool such as CDT can be used to define simple and useful diagramming notations; how to make use of other problem relevant data, e.g., raster images; and how a diagramming tool may be integrated as part of a range of tools applicable to a given problem.

In this example, an existing residential design is given as the context. The task is to add one more space, i.e., a bedroom, and find possible plan layouts that meet a number of dimensional and other considerations. As a first step: (i) an EntityObject named function is defined together with a number of attributes, and (ii) a RelationObject named adjacency is defined. Once these objects are defined in their definition windows, icons representing these objects are automatically placed in the main diagram interaction win-
One rule is defined in which an adjacency relation associates two function objects. These steps are not shown in the figures included here.

In the next step, a raster image of the existing design is brought in the diagram interaction window (Figure 8) and a diagram representing existing plan is constructed. As shown in Figure 9, the raster image is temporarily removed from the view; a new function labeled bedroom3 is added; and all the relevant textual attributes are given appropriate values in the text panes (at the bottom right in Figure 9).

With the exported data, the external program finds that it is not possible to simultaneously satisfy all the requirements as specified in this problem. At present, CDT is not able to indicate to the user which constraints need to be modified if a solution cannot be found for a given set of constraints. In such cases, any restructuring of the problem is the responsibility of the user. In the particular problem illustrated here, it is evident that the adjacency relationships form a planar graph which implies that the graph can be realized as a two-dimensional layout. Since the solution cannot be found, either the adjacency relationships or the specified dimensional values need to be modified.

As shown in Figure 10, the diagram is modified by moving the function labeled bedroom3 so that it is now connected to the south instead of the north. After exporting the data, the external program is now successful at finding a number of plan layouts which meet the modified set of requirements. The generated plan solutions are illustrated in Figure 11. Due to the space constraints, we have presented here only one example of the use of CDT but it has been applied in diverse tasks such as entity-relationship diagrams for database schema, simulation of Petri nets, minimal spanning trees, and shortest path finding problems. In all these cases, CDT was not required to be modified in terms of the information or interaction needs; the only enhancements that were needed dealt with encoding specific algorithms or methods to manipulate information that CDT already stores internally.
Figure 8. Use of reference data in diagram.

Figure 9. Diagram of new requirements.
Figure 10. Diagram of modified requirements.

Figure 11. Generated plan alternatives.
Table 3: Exported Data from CDT

<table>
<thead>
<tr>
<th>site</th>
<th>1000</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaces</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>FUNCTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>north</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>south</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>east</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>west</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>hall</td>
<td>120</td>
<td>800</td>
</tr>
<tr>
<td>living</td>
<td>350</td>
<td>700</td>
</tr>
<tr>
<td>dining</td>
<td>250</td>
<td>500</td>
</tr>
<tr>
<td>kitchen</td>
<td>150</td>
<td>500</td>
</tr>
<tr>
<td>storage</td>
<td>120</td>
<td>500</td>
</tr>
<tr>
<td>bedroom3</td>
<td>250</td>
<td>550</td>
</tr>
<tr>
<td>bathroom</td>
<td>180</td>
<td>400</td>
</tr>
<tr>
<td>bedroom1</td>
<td>250</td>
<td>550</td>
</tr>
<tr>
<td>bedroom2</td>
<td>250</td>
<td>550</td>
</tr>
</tbody>
</table>

| adjacencies 1 |
| living       | hall | 100 | north | living | 300 |
| living       | dining| 150| dining | south  | 200 |
| hall         | kitchen| 100| dining | kitchen| 100 |
| hall         | storage| 100| north | bedroom3| 200 |
| hall         | bedroom3| 100| hall  | bathroom| 100 |
| bedroom1     | north | 200| bedroom2| hall  | 100 |
| bedroom     | hall  | 100| bedroom2| north | 200 |

5 Evaluation

The current version of CDT incorporates support for the definition of diagrammatic notations in terms of the following: objects and relations of interest, and rules of composition defined over objects and relations. Using a defined notation, CDT can be employed to generate diagrams that conform to the rules of composition in a given diagrammatic notation.

CDT was implemented as a research tool (Dave, 1993) for investigation of issues related to diagram generation. For this purpose, its implementation has been successful. One important lesson to emerge from the development of CDT is that graphic representations are sometimes much easier to develop manually than to express as computational procedures. graphic languages, especially their specifications, can sometimes be quite unwieldy and may be better expressed with the directness and economy found only in sentential representations. For example, rule specifications in CDT could be made very compact if CDT were to allow a sentential specification using named variables attached to
various object and relation types. Thus, there is a need to strike the right balance between
graphic and other forms of specification and interaction in a computing system.

The diagrams generated using CDT may be translated into textual formats so that
information represented in diagrams may be used for further processing tasks. If such
translation is done using the generic format in CDT, external programs need to reformat
the output from CDT to make it consistent with their individual needs. The cost of doing
this is that a translator needs to be coded by external programs; the benefit is that external
programs need not know any details about how data are organized in CDT. On the other
hand, directly adding a custom method makes it possible to produce exactly the informa-
tion in the format desired by external program, and thus the output generated is directly
usable. The cost of doing this is that a method has to be added in CDT with the full under-
standing of how CDT manages its data structure. A better solution may be that CDT pro-
vides modifiable templates so users need not know the internals of CDT in order to specify
the exact format in which information is to be exported.

The facility in CDT to attach constraints on relative locations of objects as part of
rule specifications extends the scope of this system but it needs to be enhanced in three
specific ways. First, it should be possible to use not only the coordinates of the bounding
box of objects but also any link point of such objects as operands in the constraint specifi-
cation. Although this is possible in the current implementation of CDT, the existing con-
straint satisfier is limited and needs to be extended to handle link points of objects as
constraint terms. Second, the only operator permitted in constraint equations is an inequal-
ity operation. This may be extended to accept other kinds of operators. Third, a more ex-
tensive constraint satisfaction machinery may be needed to handle other kinds of
constraints as well as to find efficient ways of satisfying and maintaining the defined con-
straints. Although the major focus in developing CDT was on easy specification of con-
straints, the process of constraint satisfaction in CDT at present can lead to cyclical
violations of constraints. For example, CDT cannot distinguish transitive relations such
as right-of and thereby lacks useful inferencing capabilities.

The decision to implement CDT in the object-oriented environment has been large-
ly beneficial. It has allowed implementation to focus on selected aspects at a given time.
Small pieces of the whole program have undergone major changes at one time or another;
this has had no impact on the other pieces. This may be attributed to the concepts of in-
formation hiding, data abstraction and inheritance that are supported in the object-oriented
programming paradigm. On the negative side, since CDT is implemented on a personal
computer in a language that does not provide 'hooks' into foreign languages and programs,
it has not been possible to test the use of CDT in real time with more complex information.

The design of the interface has not been evaluated so far, and hence no definitive
assessment can be provided in this regard. It is only through careful experimental evalu-
ations that it can be judged whether CDT displays the right amount of information in the
right form with a transparency of operation that is found meaningful to users. On the other
hand, there are no systems next to which CDT may be evaluated to offer useful compar-
isons. The closest kind of computing systems to CDT are those based on the paradigm of
shape grammars, for example, DiscoverForm (Carlson and Woodbury, 1991). In this re-
gard, little work exists or is reported that is aimed at articulating the minimum information
needs, operations, and interaction facilities, using which composition of shapes can be
specified and maintained in an graphic environment. In most reported works, it is usually
the case that rules are coded directly in a selected programming language.
6 Extensions

The following are some extensions that we envisage for extending the scope and power of CDT.

- It should be possible to construct and display simultaneously diagram alternatives for a given information context.
- Ways of integrating appropriate theoretical results from the graph theory need to be investigated. Of particular relevance are the theorems that deal with finding planarity in graphs (in which no edges connecting nodes cross each other). This kind of integration can lead CDT to support inferences (e.g., which concepts may be reached from which other concepts).
- Since CDT internally maintains information represented in the form of a lattice, it should be possible to incorporate ways of regularizing or balancing graph layouts in two dimensions.
- While CDT supports representation of information at various levels of detail, there is no facility for collapsing all the levels into one level. This issue needs further articulation so that no semantic violations will occur as a result of flattening different information levels.
- With the use of named variables in CDT, an opportunity exists to support specification of functions or procedures. This should ideally be achieved by closely adhering to a non-programming mode of expression. Such functionality may lead to a useful classification of relations which recur in a number of information representation tasks.
- Ways of linking information between diagrams, maps, pictures, and graphs require further work (Ervin, 1992) so changes in one representation will be reflected as changes in all the other linked representations.

Acknowledgements

I would like to acknowledge the assistance of Mr. Shen-Guan Shih whose implementation of a space layout program has been used in parts of this study.

References


