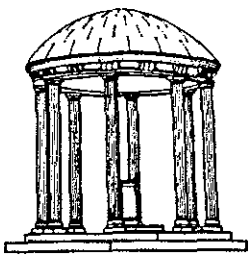
The Display of Temporal Information

TR86-019

July 1986

Karen Shannon



The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

THE DISPLAY OF TEMPORAL INFORMATION

by

Karen Shannon

A Thesis submitted to the faculty of The University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Master of Science in the Department of Computer Science.

Chapel Hill

July 1896

Approved by:

Adviser

Reader

Reader

Acknowledgements

Rick Snodgrass as advisor provided valuable support and encouragement during the entire project. His interest and guidance kept things on track. Also thanks to David Beard and Fred Brooks for valuable comments on the system.

KAREN SHANNON. The Display of Temporal Information

(Under the direction of RICHARD SNODGRASS)

Computer graphics has been used as an effective medium for displaying and accessing static information such as that found in a conventional database. Recently, issues concerning the display of temporal information have emerged. This research has concentrated specifically on the design of a graphical system to display temporal information. The thesis identifies the issues that must be resolved when designing such a system. In doing so, we considered the human perception of time, the properties of the time domain in temporal databases, and how the chosen representation of time can affect the perception of time. Several comprehensive examples show how various representations can portray relevant information. The implementation design shows the feasibility of such a system. Limitations of the system and issues for future research are discussed.

Table Of Contents

The Approach

Comprehensive Examples

Implementation

Evaluation and Conclusion

Appendices

Table Of Figures

The Approach

This part discusses an approach for designing a graphical system to display temporal information. The actual approach is discussed in Chapter 4. Chapters 1-3 provide background material. Chapter 1 gives an introduction and explains the various categories of databases. Chapter 2 discusses the issues in displaying both static and temporal information. Chapter 3 gives an overview of previous research in the areas of static and temporal information display and identifies directions for future research. Finally, Chapter 4 outlines an approach for the design of the graphical system.

CHAPTER 1

Introduction

Computer graphics has been used as an effective medium for displaying and accessing static information such as that found in a conventional database [Herot, et al 1980]. Graphical representation of data is useful in conveying information because pictures are processed by the brain much faster and with more accuracy than their equivalent text [Lodding 1983]. Computer graphics also allows one to view the same data in a variety of ways, thus emphasizing different trends and relationships.

Recently, issues concerning the presentation of temporal information have emerged. Databases can be classified into four types based on their support of temporal information [Snodgrass 1986]. These types are *static*, *rollback*, *historical*, and *temporal*. Static databases are updated by replacing information resulting in data values only at the "current" time. Rollback databases contain all past states of the static database as it is updated over time. All information is associated with a *transaction time*, the time it was stored in the database. Historical databases contain the time when the information being modeled was valid, termed *valid time*. Finally, temporal databases contain both valid and transaction time for information.

This thesis is concerned with the graphical representation of valid time although similar techniques can be used for representing transaction time. Therefore, we use a historical database to model data. Our research has concentrated specifically on the design of a graphical system to display temporal information. We identify the issues that must be resolved when designing such a system. In doing so, we considered the human perception of time, the properties of the time domain in temporal databases, and how the chosen representation of time results in a different perception of time. Our approach in representing both static and temporal information is to provide the flexibility to view data in several ways, thus emphasizing different aspects for different applications. Several comprehensive examples show how various representations can portray relevant information. The implementation design shows the feasibility of such a system.

Included in the design is the specification of a language used to interactively associate graphical aspects with data and with time. Limitations of the system and issues for future research are identified.

Part 1 gives an overview of our approach. This chapter provides a general introduction. Chapter 2 discusses various issues relating to the display of static and temporal data. Chapter 3 gives an overview of previous research in the area. Chapter 4 outlines an approach for designing a graphical system to display temporal data. Part 2, containing Chapters 5, 6, and 7, presents several comprehensive examples. Part 3 discusses the implementation of the prototype system. Chapter 8 describes the underlying data structure. Chapter 9 gives details on the graphical language. Chapter 10 desribes the time displayer and Chapter 11 discusses the Interpreter. The chapters in Part 4 evaluate the prototype system and suggest areas for future research. Appendices A, B, and C give the specification of three important data structures. Appendix D gives the BNF for the Schema Editor commands. Appendix E contains the complete example commands from which portions appear in the text.

Words or phrases denoting important concepts appear in *italics* when first introduced. These words also appear in a glossary at the end of the text. In the examples and running text, user input appears in `fixed-width font`. All keywords in the graphics language and temporal query language appear in **`fixed-width bold`**. Figures contain a combination of fonts. Keywords appear in **`fixed-width bold`**. Values which can be specified by the user appear in `fixed-width font`. Tables within figures are in Roman font. All other text is in Roman font.

CHAPTER 2

Issues in Displaying Information

A graphical system can provide a user-oriented display of the information in a database by providing the flexibility of presenting the same data in various ways depending on the user's needs. This flexibility, however, creates several problems for the designer of the system. The following section examines the issues in designing graphical displays for conventional databases. The next section discusses the additional problems encountered when extending this system to display temporal information. The last section summarizes the study.

2.1. Displaying Information in Conventional Databases

The issues in designing a system for displaying information in a conventional database relate to the representation of the data, the overall layout or presentation of the data, and the efficiency of the display system [Newman & Sproull 1979].

There are several issues to consider when graphically representing data. First, a representation must be chosen so that data is displayed in the most effective way. The image must carry a specific message. This is difficult because often the ideas of the designer of the image are very different than the ideas of the user. Another problem is that some images convey undesirable messages or can have different meanings in different contexts [Lodding 1983]. Secondly, relevant attributes of the data must be displayed to differentiate between distinct data values. A display should also provide several alternate representations of the data. Different tasks may need to emphasize different attributes of the same data. Also, there is sometimes the need to present the same data at different levels of detail depending on what aspects of the data are most relevant to the task. Finally, the display should allow the interactive creation of representations for new data, namely new data derived by querying the database. It may not be reasonable to provide defined

representations for every possible derived entity.

The second area in the design of graphical display systems relates to the overall layout or presentation of the data on the screen. First, there is the consideration of limited screen space. Given a large set of data representations which do not fit on the screen, the system must determine which portions to display. Next, the overall layout must preserve any implicit relations among data values. The data must be presented in a manner that does not obscure these relationships. Finally, the size of each data object must be large enough for it to effectively convey a specific idea. The system must be aware of the size and resolution of the display in order to effectively display each data value.

The final area in the graphical system design is the efficiency of the graphics display system. The graphics system must be efficient enough to use interactively. Hence, the response time between the input command and the reflected changes in the displayed information must be kept to a minimum. In many situations, tradeoffs between complexity of pictures and time to generate pictures must be made. In an interactive system where response time is especially important, the complexity of the pictures must be compromised. However, overly complex images should be avoided anyway when conveying messages [Lodding 1983].

2.2. Displaying Information in Historical Databases

The issues relating to data representation, data presentation, and system efficiency are also apparent when designing systems to display temporal information. In addition, several other problems arise when determining a representation for the time domain of the data. The following section discusses the human perception of time. The next section explains the properties of the time domain in historical databases. Finally, the last section discusses the issues involved when representing the time domain.

2.2.1. Human Perception of Time

An investigation of the display of temporal data would not be complete without some insight into the human perception of time. A major difficulty in determining a representation for time is that time is perceived differently by different people and even differently by the same person when subjected to various environmental conditions. Therefore, like a verbal description of time, a pictorial description can be limiting and difficult to understand.

An analysis of the human perception of time is quite complex. In this section, we greatly simplify it for our purposes. A complete analysis is beyond the scope of this research.

For purposes of this research, we divide the concept of human perception of time into two areas:

- how one *experiences* time
- how one *describes* time

The first area, the *experience of time*, is important because it affects how one interprets temporal information. Since we are conveying information to people, subjective time is more important than actual time. The second area, the *description of time*, is important because, ideally, we would like to represent time in a way which is equivalent to its verbal description.

One does not directly experience time but rather experiences particular sequences and rhythms. A person's awareness of time is based on their attention to the number of changes which occur in a given time interval. The more attention paid to time, the longer it seems [Whitrow 1980]. For example, an hour lecture seems very long when one looks at their watch every 5 minutes. The same amount of time seems much shorter when one's attention is on a specific activity, such as a basketball game.

There are two fundamental ways in which time is experienced [Shallis 1983]. Sometimes it is perceived to flow where the passing intervals of time (seconds, minutes, days, etc.) are endless. This corresponds to a time interval in which perceived changes are gradual, for example, the sun rising and setting over the interval of a day. The second way time is experienced is as a separation of events making past, present, and future distinct. An example is a day at work with meetings scheduled every hour in different conference rooms. Each meeting is distinct from the previous meeting and the next meeting.

A description of time can be in terms of *duration* or *succession*. According to Kummel, neither duration nor succession alone can describe time since duration without succession becomes a static picture and succession without duration contradicts the notion of an event existing in time [Kummel 1966]. However, we can emphasize either duration or succession when describing time.

To emphasize duration, there must be a coexistence of past, present, and future. Time perceived as a line as in a space-time diagram gives this coexistence but transposes time into a spatial image. A spatial description of time has the implicit assumptions that time can be spatialized and that it is linear, continuous and connected. These assumptions have been necessary for scientific convenience but are not necessarily

accurate in a more philosophical description of time. This is because time has the additional property of unidirectionality which space does not. That is, there is a recognition that time only flows in one direction, forward. Objects may be still in space but are always continuously moving in time [Shallis 1983]. A spatial description of time, therefore, is adequate only when expressing static durations. Within this duration, however, is the image of a point in successive positions along a line thus giving a description with both duration and succession [Kummel 1966].

A time description emphasizing a succession of events implies that time is associated with change or motion. A representation of motion assumes that time is not an independent characteristic of an event but rather a way to describe the relations between events, reinforcing the concept that one does not directly experience time but rather experiences changes. Whitrow states, "it is not time itself but what goes on in time that produces effects" [Whitrow 1978]. A description of time as motion also has limitations as does the space-time diagram. Since time seems longer when more attention is paid to it, representing time with motion can possibly distort one's estimation of time.

From this brief study, we concluded that there is no single perception of time. Time is experienced in different ways depending on the events happening in time. Time is also described in different ways depending on the application it is used in. This conclusion had a great impact on how we chose to graphically represent time.

2.2.2. The Time Domain in Historical Databases

Historical databases contain data valid at particular times. Changes to the database are performed by adding information rather than replacing information giving a progression of data values over time [Snodgrass 1986]. This can be contrasted with conventional databases which only contain data valid at the "current" time.

The time domain of data can be one of two types, *event* or *interval*. An event is only valid at an instant of time while an interval is valid between and including two events.

Adding a time domain to data may result in instances of incomplete or incorrect information. This is because an event may not occur at a specific time or this information may not be known. Therefore, data returned from a query can be considered as *true* or *indeterminate*. An indeterminate event may represented

by three consecutive time intervals. The first interval is the time it is possibly valid, the middle interval is the time it is definitely valid, and the last interval is the time it is possibly invalid [Snodgrass 1982].

2.2.3. Representing Time

Determining the one representation of time is analogous to finding the perfect phrase to describe it. Time is different things to different people and has different meanings in various applications.

Representing the time domain in temporal databases involves choosing a representation for events, a representation for intervals, and a representation for indeterminacy. Several questions arise. In the following discussion, the term *event* is used to refer to data with a time domain of type event and the term *interval* is used to refer to data with a time domain of type interval.

(1) *Representing one event*

An event cannot be represented as static data because this contradicts the notion of an event existing in time. An event really has a duration equal to the smallest time unit in the database. Another issue is that time is not really an independent characteristic of an event like other attributes but rather a way to describe the relations between events. How is time shown when there are no other events to relate to?

(2) *Representing one interval*

In an interval, there are two properties that can be shown. The duration or length of time can be shown and the actual starting time or stopping time of the interval can be shown. In some applications, the duration is important. In other applications, the starting or stopping time is important. In still other applications, both are important.

(3) *Representing a sequence of events*

In this situation, we have a succession of ordered events, where the duration between events is measurable. If the succession should be emphasized, the representation of time can be associated with motion making past, present, and future distinct. However, if the duration is also important, this may not be an adequate representation.

(4) *Representing a sequence of intervals*

In this situation, we have a succession of intervals ordered by starting time where the duration between intervals as well as the duration of overlapping intervals is measurable. Again, there are several properties that can be shown. The duration of each interval can be compared. The starting or stopping time of each interval can be compared thus reducing the sequence to a sequence of events. The duration between intervals can be compared. Finally, any combination of the three can be compared. Again, it reduces the problem to one of emphasizing duration, succession, or both.

(5) *Representing a sequence of events and intervals*

If there does not need to be a distinction between events and intervals, the problem is the same as representing a sequence of intervals where each event has a duration equal to the smallest time unit in the database. If there should be a distinction, then a representation must be chosen that differentiates between the two. This involves emphasizing duration.

(6) *Representing indeterminacy*

If an indeterminate event is represented as three time intervals, a representation must be chosen for the "possibly valid" interval, the "definitely valid" interval, and the "possibly invalid" interval. If indeterminate data does not need to be distinguished from true data then all three representations can be the same. If a distinction should be made, the representation for the "possibly valid" interval and the "possibly invalid" interval must capture the inherent nature of indeterminacy. That is, the representation must portray the uncertainty in the data.

The questions above are concerned with representing time in terms of its description in that choosing a representation involves choosing to emphasize duration, succession, or both. Also relevant is representing time in terms of how it is experienced. Should it be perceived to flow where the changes are gradual? Alternatively, should the events be separated by making distinct changes? Choosing particular representations for time can make these changes more or less distinct thus portraying a different experience of time. Also related is how the representation of an object changes over time. For example, if an object changes position over time, time will appear more separated. Alternatively, if an object intensity changes gradually over time, time will be perceived as flowing.

2.3. Summary

This chapter identified the issues that must be resolved when designing a display system for a temporal database. The issues in displaying information in a conventional database are choosing a representation for data and determining the layout of the data while keeping the system efficient enough to use interactively. The issues in displaying information in a temporal database include the issues in displaying static information plus a consideration of the human perception of time, the properties of the time domain, and the problems encountered when representing time domains.

The human perception of time relates to how one experiences time and how one describes time. Time is experienced as flowing or as a separation of events. Time is described in terms of duration and/or succession. Representing time in a temporal database involves choosing a representation for events, intervals, and indeterminacy. The representation chosen can emphasize duration, succession, or both. In addition, a representation can make time perceived as flowing or as a separation of events.

From this study, we concluded that there is no "one" representation of time that could suffice for all situations just as there is no "single" perception of time. Our approach for representing temporal information, encompassing many representations of time, is outlined in Chapter 4.

CHAPTER 3

Previous Work

Much work has been done involving the use of graphics with databases. In addition, some research has been done in the area of displaying time-varying data. The next section describes relevant graphical display systems for static data emphasizing the techniques used for the representation and presentation of the data. The following section discusses the representation of time in display systems for time-varying data. The last section summarizes the current state of the art and presents areas which require further research.

3.1. Display Systems for Static Information

The Picture Building System (PBS) [Weller & William 1976] uses a relational database to store graphical descriptions of data in addition to storing the data values. The graphical descriptions can contain graphic primitives and references to other predefined objects. Storing graphical data in a relational database allows data objects to be displayed directly from the information in the database. This makes the data independent from application programs, permitting different logical views of the data. Unfortunately, the execution efficiency when using a relational database to store the graphical descriptions will always be less than when using tailored data structures. This system has different goals than ours in that it is primarily concerned with the storage and retrieval of graphical information rather than other types of information.

Another area of research is in spatial data management which is the technique of accessing data through its graphical representation. Two overlapping projects were funded by the Cybernetics Technology Office of ARPA, one in 1976 and the other in 1977. In the first Spatial Database Management System (SDMS) [Donelson 1978], the database is presented to the user on a six foot by eight foot rear-projected color television display. Moving about the data surface is accomplished by joysticks and a touch sensitive

screen. Sound and sensation are also used to manage the data. The second SDMS [Herot 1980] also allows browsing through the database without the use of a formal query language. This system uses two screens for the graphical output. One screen contains a world view of the data and the second screen contains a specific view corresponding to a highlighted rectangle on the world view screen. The user can move this rectangle, thus changing the specific view, and can also vary the detail of the specific view by zooming in and out. Both SDMS systems use icons (symbolic pictures) to represent data objects following the principles for effective display of data documented by Morse [Morse 1979].

A disadvantage of the SDMS is that the representation of the data is fixed so the user can only view it in its predefined representation rather than viewing it in different ways depending on the task. Efforts to remove this limitation led to the View System [Friedell, et al 1982] which augments the SDMS with the ability to dynamically generate icons as they are required. The View System provides the ability to choose the attributes of the object to be displayed as well as providing a choice of several predefined views of the object.

Another system which uses the concept of spatial data management is the Automated Desk [Yedwab, et al 1981]. This system is used to access programs and data of a large computer thus simplifying the learning process for new users of the operating system. Programs and data are displayed on a large spatial surface called the desk top. This surface is larger than the screen window so particular portions of the desk top are displayed by scrolling across the surface. There is an additional provision to show an undetailed view of the whole desk top. The system also provides interactive creation and deletion of objects. New objects are created by specifying the shape, border character, label and program to be executed when the object is picked. An object is deleted by invoking the delete command and then pointing to the object.

The spatial data management systems provide easy retrieval of information without a precise specification of the data. However, these systems are more suited for browsing through the database than for locating specific information. Complex queries still require the use of a database query language. The spatial organization of the data with various levels of detail provides a possible solution to the limited screen space problem but is somewhat restrictive in its presentation of data. Certain databases could not be effectively displayed with this organization. Also, the use of multiple screens makes the system device-dependent which is a disadvantage for a general purpose display. A third limitation is that there is no provi-

sion for dynamically changing a graphical view at the request of the user, although the View System does provide several predefined views.

Another relevant area of research, investigated by Friedell, one of the designers of the View System, is the automatic synthesis of graphical object descriptions [Friedell 1984]. Friedell describes a technique for automatically synthesizing graphical object descriptions from a high-level object specification called a quasi-description. A quasi-description gives the identity of the object type and the identity and values of its important descriptive attributes. A graphics knowledge base gives the graphical information necessary for the synthesis of the object given its type and attribute values. This technique was used in the implementation of the View System. While it provides an effective mechanism for associating graphical aspects with existing entities and attributes, an apparent limitation is that there is no mechanism for dynamically generating representations for new entities.

3.2. Display Systems for Time-Varying Information

The display systems for time-varying data incorporate some of the techniques used for displaying static data. In addition, animation is used to display the time domain. The following section describes a graphic system for displaying temporal data and several systems for program visualization.

An existing graphical display system for time-varying data is the Modal Data Management Graphical Interface (MDM/GI) [Ariav & Morgan 1982]. This system provides an animated presentation of time-oriented data. The data objects are represented as icons. The data and the associated icons are stored in the MDM database. The system provides control, such as direction of movement and speed of movement, over the displayed sequence. In addition, a "Time-Icon" is displayed on the screen showing the time each event occurs. This icon can be a calendar page, a month scale, or a clock face and can be chosen by the user or provided as a default based on analysis of the time domain of the retrieved data objects. Although the MDM/GI focuses on the dynamic presentation of temporal data in its prototype implementation, Ariav mentions other techniques which could convey time. One is a three dimensional representation where present events are closer to the user than previous events. Another is a color coding of time where present events have increased intensity and previous events are faded. A third technique uses line or bar charts plotted against a time axis.

Research in the area of program visualization is also somewhat relevant. A graphical system for algorithm animation is a typical example [Brown & Sedgewick 1984]. This system graphically represents fundamental characteristics of algorithms and displays the algorithms in execution. The user has the capability of controlling the execution of the algorithms and changing the presentation of the views. In addition, the system is implemented on a workstation with a bitmap graphic display which provides the capability to create different view windows.

Another system for program visualization was developed to support software development [Kramlich, et al 1983]. This system provides a graphics editor to create, edit, and view static diagrams, a dynamic object controller to bind the diagrams to the program code, and a design database where these diagrams are stored. The user can shift between an overview level and a detailed level and can control animation speed.

Like the MDM/GI system, the two program visualization systems provide only one representation for time. This does not provide the flexibility needed for viewing time differently depending on the task. The graphical representation of the algorithm is also fixed limiting the user to a description which may not highlight the most relevant issues.

3.3. Summary

The graphical display systems described above have presented possible solutions to some of the problems in graphical display. The Picture Building System and Friedell's technique both provide a mechanism for associating graphical aspects with entities and attributes. The spatial database management systems provide several levels of detail for the same view as well as providing one solution to the problem of limited screen space. The time-domain has been represented using animation in both the MDM/GI and program visualization systems.

The limitations of these systems highlight some problems which require further study. First, representations for derived and existing entities need to be generated interactively. Secondly, more techniques for handling limited screen space need to be investigated for their applicability to various tasks. Finally, more research needs to be done for solving the time representation problem. A representation for events and a representation for time intervals must be provided as well as a representation for indeterminate data. Also, these representations should be different for different applications.

CHAPTER 4

Overview of the Approach

Because temporal information is a generalization of static information, a graphical system used to display temporal information should be a generalization of a graphical display system for static information. This graphical system should provide capabilities for graphically representing data in two dimensions on a high resolution display and should manage the layout of the data in an effective way. It also should be efficient enough to use interactively. In addition, it should provide capabilities for displaying time domains and indeterminacy.

Our research has concentrated on those issues relevant to the display of temporal information, though portions also apply to the display of static information. In particular, the scope of the investigation was limited to techniques for representing the time domain. Since these techniques are generalizations of those used to display other attributes of data, they can also be effective when used to display static information.

This chapter gives an overview of the approach taken in designing our graphical display system for temporal information. The techniques described assume a temporal relational database. However, the techniques can be generalized for use with any of the database models. The first section discusses the essential features for our approach. The next section discusses the data model, the graphical model, and the screen model with the third section explaining the relationship between the three models. The fourth section discusses the representation of the temporal portion of the data. Finally, the last section describes the structure and main features of the prototype display system.

4.1. Essential Features

Our research has built on concepts from the systems described in Chapter 3, especially the MDM/GI system [Ariav & Morgan 1982] and object synthesis techniques [Friedell 1984]. This section describes the

essential features of our approach and how each relates to previous work.

(1) *Iconic representation of objects*

Iconic representation of data objects was used in each system described in Chapter 3. This feature has been shown to be effective in these systems.

(2) *Graphical data independence from application programs*

Graphical data independence exists in the PBS [Weller & William 1976], the View System [Friedell, et al 1982], and in the MDM/GI [Ariav & Morgan 1982]. In the PBS and the MDM/GI the graphical data is stored in the same database as the data it describes. In the View System, graphical data is stored in a separate structure. In our system, graphical data is also stored in a separate data structure. We chose not to store graphical data and and other data in the same database for two primary reasons. First, a specific data structure for graphical information allows faster access to information than the operations for a relational database. Since it is known which items are accessed most frequently, the data structure can be tailored so that these are done very efficiently. Also, separate structures allow concurrent accessing of data and graphical descriptions. Secondly, a separate data structure allows different users to create their own description files when accessing the same database. This allows the same database to be used for applications which require different graphical views.

(3) *Multiple views*

In the PBS, the View System, and the MDM/GI the representation of data is not fixed. The user may select both the data and the attributes to be displayed. The View System also gives the user a choice of different representations for the data. Our system extends this feature by allowing the user to interactively create their own description of the data.

(4) *Automatic Synthesis of Graphical Object Descriptions*

The View System uses techniques for the automatic synthesis of graphical object descriptions from a high level specification. The graphical description is built when given the identity of the object type and the identity and values of its attributes. Our system uses similar techniques. The advantages are twofold. First, it allows interactively generating descriptions for new entities. Secondly, it allows a major portion of the system to be device independent.

(5) *Graphical Representation of Time*

The MDM/GI, algorithm animation [Brown & Sedgewick 1984], and the program visualization [Kramlich, et al 1983] systems display time using animation. In each system, the user may control both the direction and the speed of the motion. In the MDM/GI a time icon is displayed to give the user a reference point. Our system augments this feature by providing other representations for time as well. These are described in detail in Section 4.4.

4.2. The Three Models

Much research has been done in the area of information processing. Information processing is concerned with providing both a structure for information and ways to manipulate that structure. For our system, there are three major types of information to process. First, there is the data that we want to display. Secondly, there is the graphic information necessary to display the data. Finally, there is the layout information necessary to effectively present the data on the graphics screen. This section describes the concep-

tual models used for each of these information types in three separate subsections. Section 4.3 discusses their interaction.

4.2.1. The Data Model

We can think of a data model as consisting of two elements. The first element is a mathematical notation for expressing data and relationships between data. The second element is a set of operations on the data. These operations include queries and other manipulation of the data [Tsichritzis & Lochovsky 1982]. For data, we use the relational model [Codd 1970], extended to include time. The first element of the model, the mathematical notation, is the *set-theoretic relation*. A *relation* is a subset of the Cartesian product of a list of domains [Ullmann 1982]. A *domain* is a set of values such as integers or character strings.

In the relational model, information is stored in relations. A relation can be viewed as a table where the rows are called *tuples* and the columns are called *attributes*. The relation in Figure 4.1 contains three explicit attributes (`plane`, `model`, `status`) and four tuples. The temporal attributes (`from` and `to`) will be discussed later in this section.

Figure 4.1 : An Example Interval Relation

airplanestatus (plane, model, status):

plane	model	status	from	to
CessI	414A	inrepair	08:00	10:30
CessII	414A	demoflight	08:00	10:00
PiperI	ArcherII	onground	08:00	09:00
PiperII	ArcherII	trainingflight	08:00	09:00

Each column or attribute has a *domain* which is a set of values valid for that attribute. Each relation is created using a relation schema. A *relation schema* gives the name of the relation and the name and domains of its attributes. The relation schema for `airplanestatus` is shown in Figure 4.2.

Figure 4.2 : An Example Relation Schema

```
relation name: airplanestatus
attributes:
  name: plane        domain: <name>
  name: model        domain: { 414A ArcherII }
  name: status       domain: { inrepair demoflight onground
                               trainingflight freeflight}
```

Attribute domains enclosed in "{}" represent finite domains. Each possible value for the domain is listed in the set. Attribute domains enclosed in "<>" represent infinite domains. For example, <name> is the set of all strings beginning with a letter and containing only letters, digits, and the underscore symbol "_". Attributes with finite domains are termed *finite attributes.* Attributes with infinite domains are termed *infinite attributes.*

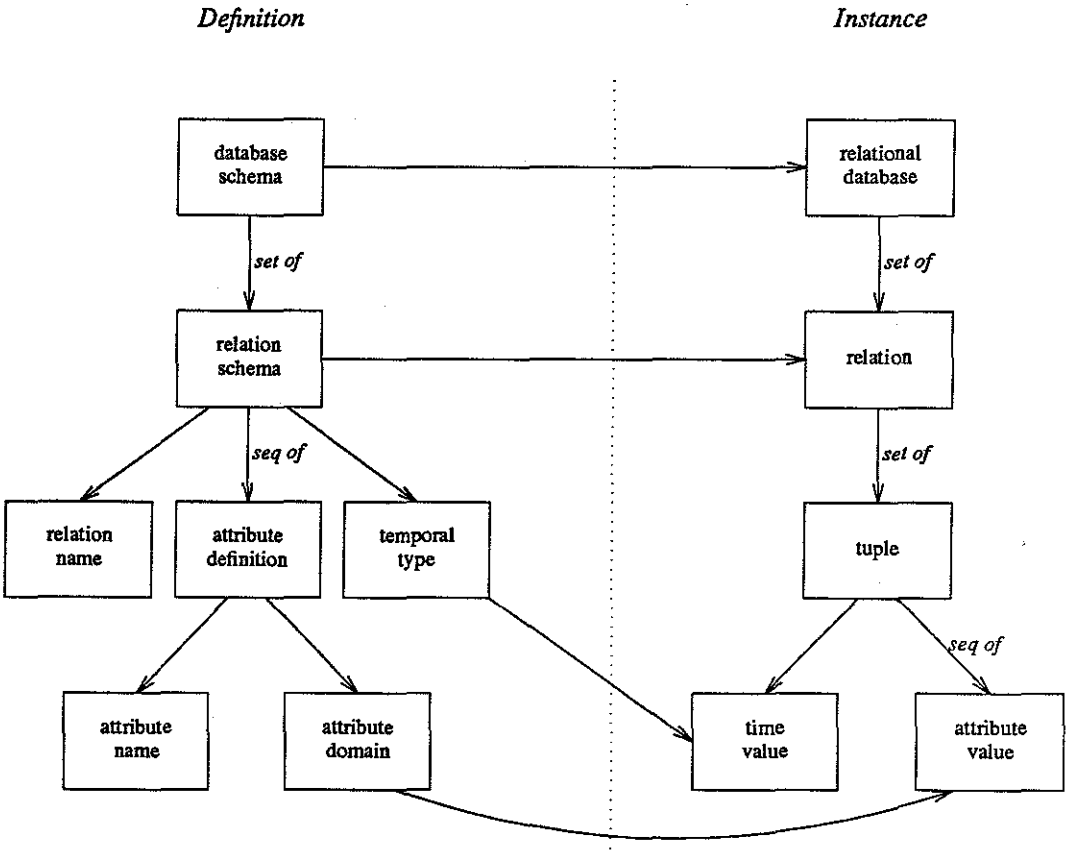
We can classify relations as *entity* relations or *relationship* relations [Chen 1976]. An entity relation contains an entity identifier, or *key*, plus other descriptive attributes. A relationship relation contains two or more entity identifiers plus other descriptive attributes. A relationship relation describes the relationship between the entities names by the identifiers. The group of entity identifiers are used as the key of the relationship relation. Neither classifications of relations may contain any two tuples that agree on all the attributes of the key.

A *relational database* is a set of relations specified by a *database schema.* Each database schema is a collection of relation schemas. The components of a database schema are used to generate a relational database. This interaction is shown in Figure 4.3. The vertical dotted line separates the defining schema and the instance. The horizontal arrows which cross the dotted line represent *defines.* For example, a database schema defines a relational database. Arrows pointing downward within the Definition and Instance columns represent *consists of.* For example, a database schema consists of a set of relation schema.

Historical databases contain the time when the information being modeled was valid [Snodgrass & Ahn 1986]. In *historical relational databases*, one or two temporal attributes are added to each relation schema. Relations with one temporal attribute, named `at`, are called *historical event relations.* The value of this attribute gives the instance of time that the tuple of the relation is valid. Relations with two temporal attributes, named `from` and `to`, are called *historical interval relations.* The values of these attributes

give the time interval that the tuple of the relation is valid. For example, in Figure 4.1, the interval given by the `from` and `to` attributes gives the time when the plane had the particular status. Temporal attributes differ from other attributes in the manner in which they are used.

Figure 4.3 : Interaction Between a Database Schema and the Database It Defines



The second element of the extended relational model is the set of operations for querying and manipulating the data. For this we use the temporal query language *TQuel* [Snodgrass 1986] which is based on relational calculus [Codd 1972]. TQuel extends *Quel* [Held et al. 1975], the query language for Ingres [Stonebraker et al. 1976], to manipulate both event and time intervals.

4.2.2. The Graphical Model

The graphical model holds the information necessary to construct graphic descriptions. In the graphical model, information is stored in *objects*. An example object is shown in Figure 4.4. An object contains a

sequence of *iconic representations* and a set of *graphic characteristics*. An iconic representation is a graphical shape such as an icon, a line, or a polygon stored as a sequence of points. The object in Figure 4.4 contains three icons, each composed of several polygons, and a text icon. Each iconic representation can also contain graphic characteristics. Graphical characteristics are aspects such as color and size. The object in Figure 4.4 has two graphical characteristics, color and coordinates.

Figure 4.4 : An Example Object

object

iconic representation:
```
icon plane
icon wingpropeller
icon wingpropeller
text "CessII"
```

graphical characteristics:
```
color black
coordinates 0,0 to 100,100
```

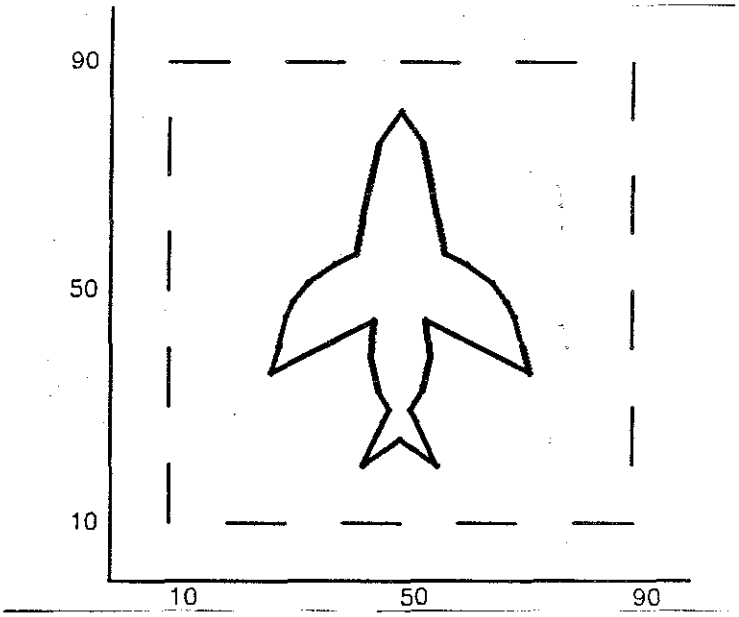
Each object is created using an *object template*. An example object template is shown in Figure 4.5. An object template contains an ordered sequence of *iconic templates*, an ordered sequence of *modifiers*, and a set of *graphic characteristics*. An iconic template names a graphical shape such as a line or polygon. In addition, each iconic template can contain graphic characteristics. An iconic template differs from an iconic representation in that all its points or characteristics may not be defined until after the modifiers are applied. A modifier defines what type of modification should be made under what conditions. A modification can affect graphical characteristics, add iconic representations, or change the intrinsic structure of an object. For example, the second modifier in Figure 4.5 adds the iconic representation of a nose propeller to the existing representation. The object template graphical characteristics are used as the default graphical characteristics for each object. Conditions specify requirements that must be met for the corresponding modification to be made. In this example, the conditions are not specified; they will be described further in Section 4.3.

Figure 4.5 : An Example Object Template

```
object template
  name:
    airplanestatus

  iconic representation:
    icon plane

  modifiers:
    condition1: add icon wing propellers
    condition2: add icon nose propeller
    condition3: add text

  graphical characteristics:
    color black
    coordinates 0,0 to 100,100
```

In the graphical model, each object template has its own coordinate system which can be specified by the user. The default coordinate system of an object template is 100×100. The iconic representations of each object are constructed using this coordinate system. This is best explained with an example. Each icon is constructed within the iconic coordinates 0,0 to 1,1. To achieve the desired size, the icon is positioned and scaled within the coordinate system of the object. For example, scaling the plane icon by 80 in both the x and y directions and positioning it at 10,10 would result in the icon shown in Figure 4.6. This figure as well as all figures representing objects were generated by our prototype display system.

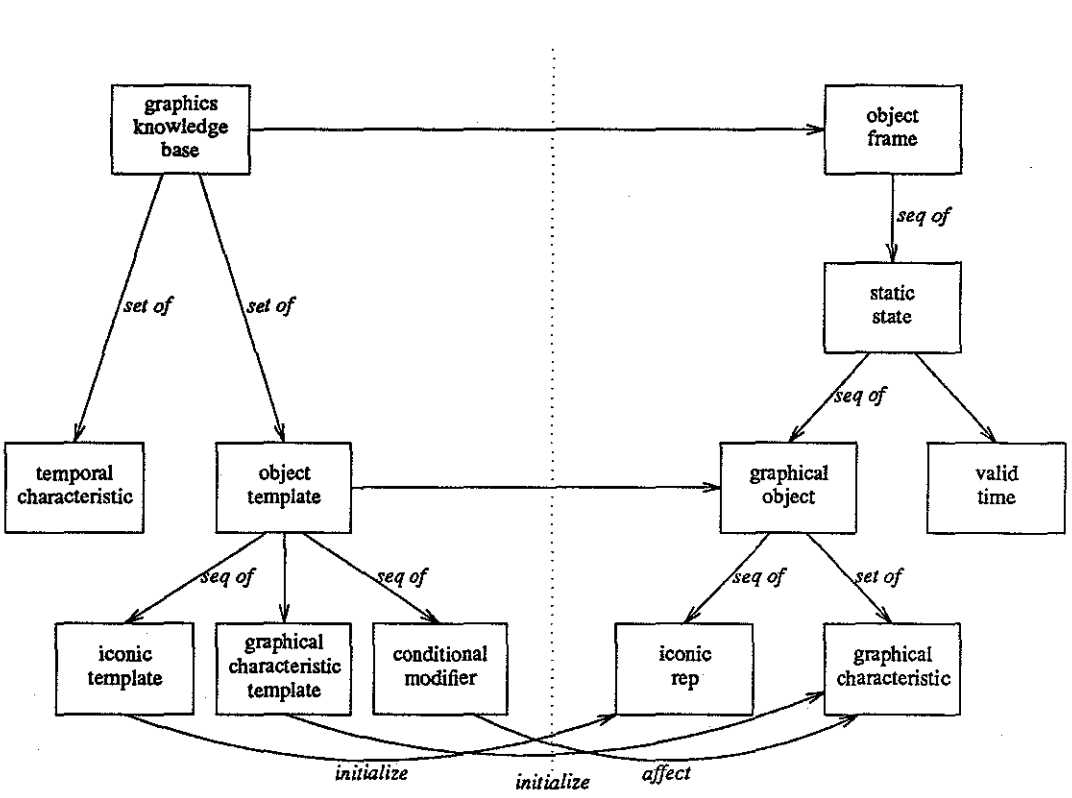
Figure 4.6 : Windowing an Icon within the Object Coordinates



A *graphics knowledge base* is a set of object templates and a set of *temporal characteristics*. Temporal characteristics include the representation of time for all object templates and other properties of time such as the direction of time and the interval step. The set of objects created from the object templates in a graphics knowledge base is stored in a structure called the *object frame*. The object frame consists of a sequence of *static states*. Each static state contains a sequence of objects and the time at which the objects are valid. The static states are ordered by valid time. The valid time can be an instant of time or an interval of time.

The components of the graphics knowledge base are used to define an object frame. This interaction is depicted graphically in Figure 4.7. The vertical dotted line separates the defining graphics knowledge base and the instance. The horizontal arrows which cross the dotted line represent *defines*. Each labelled curved arrow represents the relationship named by its label. Arrows pointing downward within the Definition and Instance columns represent *consists of*.

Figure 4.7 : Interaction Between Graphics Knowledge Base and Object Frame it Defines



A graphics knowledge base defines an object frame. The valid time for each static state of the object frame may be determined by the temporal characteristics of the graphics knowledge base. For example, if the interval step of time was 30 minutes and the direction of time was forward, then each static state's valid time would also be an interval of 30 minutes and the static states would be ordered by increasing time. An object template defines an object. The iconic representation of the object is initialized using the iconic templates of the object template. The graphical characteristics of the object are initialized using the graphical characteristics of the template. The conditional modifiers are then applied, possibly changing object characteristics.

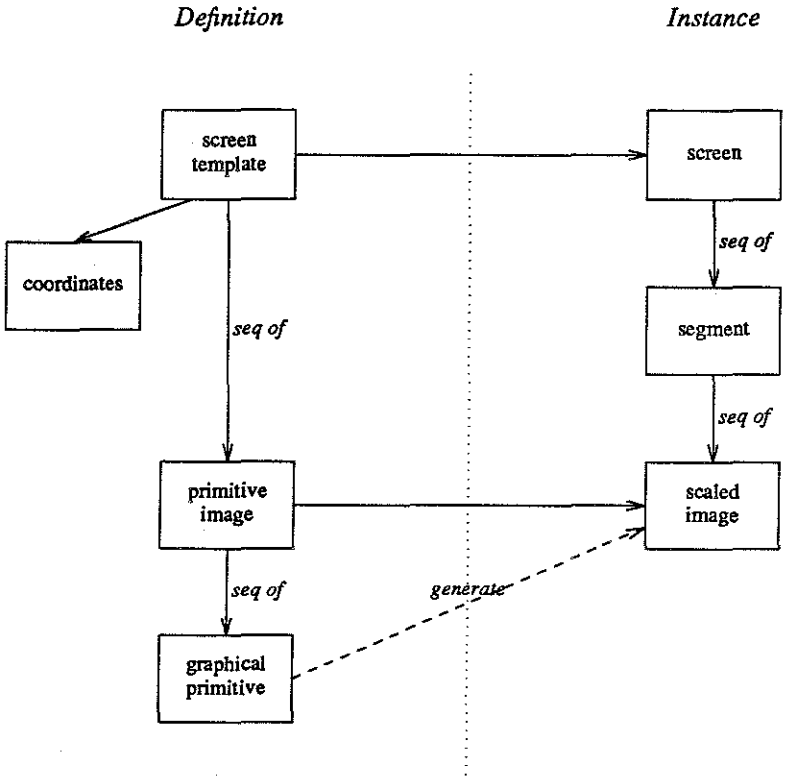
4.2.3. The Screen Model

The screen model holds the information necessary for the layout of information on a screen. A screen consists of a sequence of *segments* each containing a sequence of scaled images. A segment contains a portion of the entire screen. Each image in the segment covers one or more squares of the grid. Images

appearing later in the sequence may cover or overlap previous images.

Each screen is defined using a *screen template*. The interaction between the screen template and the resulting screen is shown in Figure 4.8. The vertical dotted line separates the defining screen template and the instance of the screen. The horizontal arrows which cross the dotted vertical line represent *defines*. The labelled dashed arrow which crosses the vertical line represents *generates*. Arrows pointing downward within the Definition and Instance columns represent *consists of*. For example, the graphical primitives contained in a defining primitive image generate a scaled image.

Figure 4.8 : Interaction Between a Screen Template and the Screen it Defines



A screen template contains a *coordinate pair* and an ordered sequence of *primitive images*. A coordinate pair specifies the lower corner and upper corner of the screen grid. A primitive image is a graphical description defined within a unit square. Each primitive image contains a sequence of graphical primitives such as polygons, color, position, and size. Primitive images generate scaled images. Position primitives specify the grid square the image is mapped to. Size primitives specify how many squares the image should cover in both the x and y directions.

For example, the screen template of Figure 4.9 would generate the screen shown in Figure 4.10. Color is shown with texture.

Figure 4.9 : An Example Screen Template

```
screen template:
    coordinates:
      lowercorner 1,1
      uppercorner 5,5

    primitive images:
      triangle
          graphical primitives:
            position 4,2
            xsize 1
            ysize 3
            red filled polygon 0,0  .5,1,  1,1
      circle
          graphical primitives:
            position 3,1
            xsize 2
            ysize 2
            green filled circle radius .5 center .5,.5
```

Figure 4.10: An Example Screen

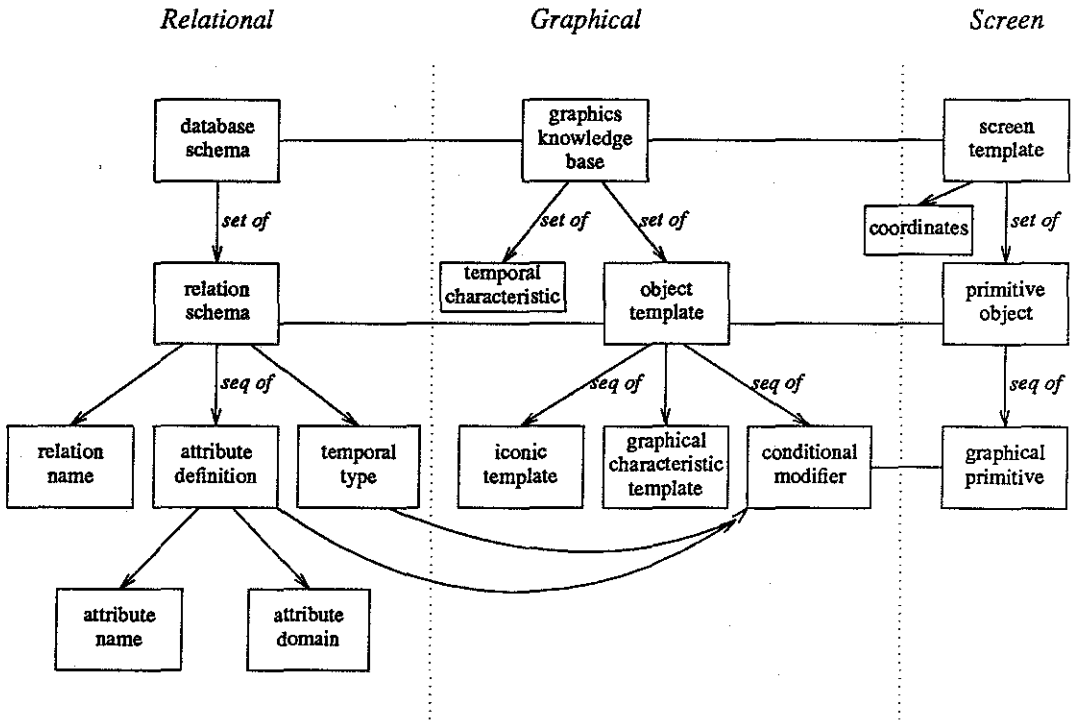
Note that the circle covers part of the triangle since it appears later in the sequence. The lower corner of the circle is positioned at 3,1 on the screen. The circle covers two unit squares in both the x and y directions.

4.3. Relationship Between the Models

The previous section described the three models used in our display system. The data model used is the relational model extended to include time. In this model, relations are defined using relation schema. In the graphical model, objects are defined using object templates. Finally, in the screen model, a screen containing a grid of squares covered by images is defined using a screen template consisting of a coordinate pair and a sequence of primitive images. This section discusses the mapping between the three models.

The interaction between the models consists of two mappings. The first mapping specifies the interaction between the defining portions of each model. The second mapping specifies the interaction between the instance portions of each model. The interaction between the defining portions of each model is shown in Figure 4.11

Figure 4.11: Interaction between Model Definitions

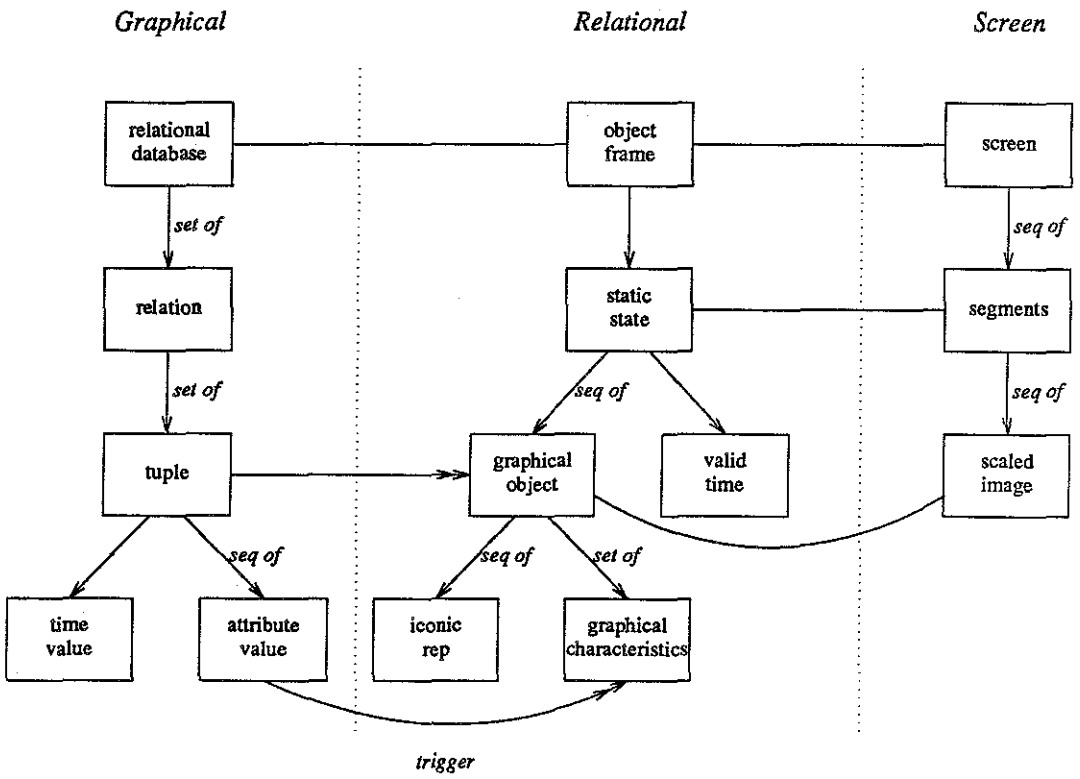


The relationship between the relational model definition and graphical model definition is shown in the first two columns of Figure 4.11. Each database schema is mapped one-to-one to a graphics knowledge base. One-to-one relationships are shown with a solid line in Figure 4.11. Each relation schema is mapped one-to-one to an object template, implying that each relation schema is associated with a sequence of iconic templates, a set of graphic characteristic templates, and, a sequence of modification templates. Each attribute definition is associated with one or more conditional modifiers. This is shown in Figure 4.11 with a two headed arrow. For finite attributes, the condition of the conditional modifier is the equality of an attribute value to some specified value. The modification associated with each possible value of the attribute can be explicitely stated. For infinite attributes, the condition is always met. Each possible value for attributes cannot be listed in a condition; therefore, the modification associated with each attribute is indirectly specified. The temporal type of the relation maps to one or more special modifiers, called *temporal modifiers*. Temporal modifiers use the time value of the tuple mapped to an object to determine if conditions are met. Similarly to infinite attributes, each possible value for time cannot be listed in a condition. Therefore, the temporal modification associated with each time value is indirectly specified.

The relationship between the graphical model definition and the screen model definition is shown in the second and third columns of Figure 4.11. Each graphics knowledge base is mapped one-to-one to a screen template. Each object template is mapped one-to-one to a primitive object. The iconic templates, graphical characterics, and modifications of the object template map to one or more graphical primitives.

The second mapping between the models which describes the interaction between the model instances is illustrated in Figure 4.12.

Figure 4.12: Interaction between Model Instances



The relationship between the data model instance and the graphical model instance is shown in the first two columns of Figure 4.12. Each relational database is mapped one-to-one with an object frame. Each tuple in a relation is mapped to one or more graphical objects. The objects are constructed using the object template associated with the relation schema of the tuple. More than one object is constructed if the representation of the tuple changes over time. The object representation is changed by each modifier which meets a condition. The conditions met are determined by the values of the tuple's attributes and time. Therefore, the attribute value triggers which graphical characteristics are set. The modifications due to the

attribute values differentiate the object from other objects generated from the same object template. All graphical objects contained in a static state are the representations for tuples at the time specified in the valid time of the state.

The relationship between the graphical model instance and the screen model instance is shown in the second and third columns of Figure 4.12. Each object frame is mapped one-to-one with a screen. A segment of the screen has a one-to-one correspondence to a static state. The scaled images within a segment correspond to the graphical objects within a static state. The images are scaled and positioned on the screen using the object's graphical characteristics of `width`, `height`, and `position`.

For example, suppose the object template of Figure 4.5 was associated with the relation schema `airplanestatus` shown in Figure 4.2. The object template name is the same as the relation name. The expanded object template is shown in Figure 4.13.

Figure 4.13: Expanded Object Template

object template

name:
`airplanestatus`

iconic representation:
`icon plane`

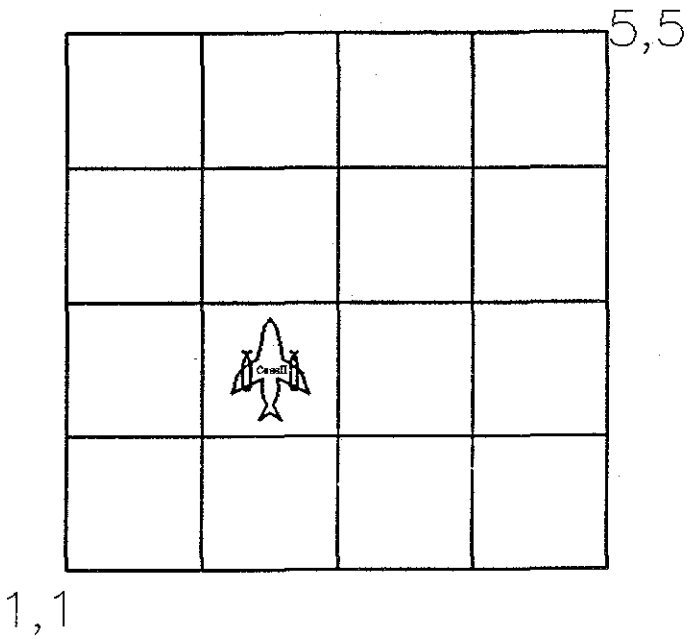
modifiers:
if model = "414A" then `add wing propellers`
if model = "ArcherII" then `add nose propeller`
if status = "inrepair" then `position object at 1 along y axis`
if status = "trainingflight" then `position object at 2 along y axis`
if status = "demoflight" then `position object at 3 along y axis`
if plane in <string> then `add text of the attribute plane value position along x axis`

graphical characteristics:
```
color black
coordinates 0,0 to 100,100
width 1
height 1
```

The graphical object constructed for the second tuple of Figure 4.1 will inherit the iconic representations and characteristics of the object template. Modifications are made to the object depending on what

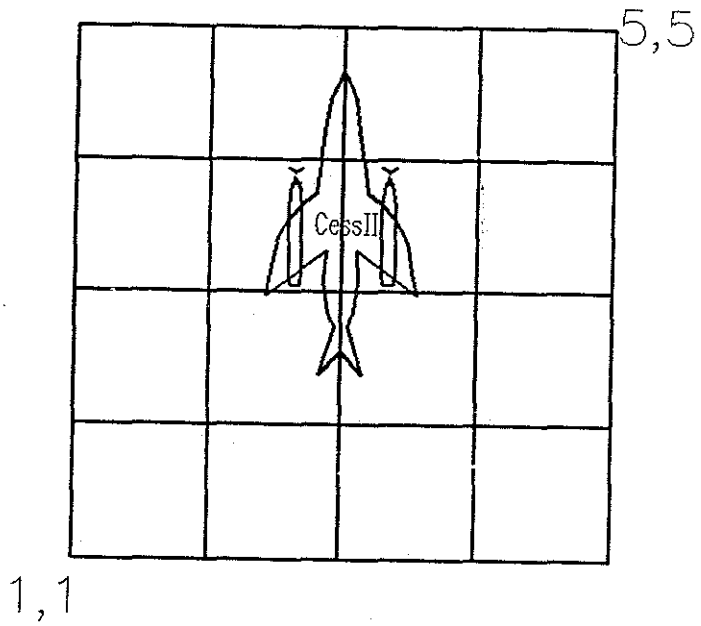
conditions are met. For the second tuple, the conditions met are `model = "414A"`, `status = "trainingflight"`, and `plane in <string>`. Therefore, the object will consist of a `black plane icon`, `wing propellers`, and `text "CessII"`. The object will be placed at position 2,2 on the screen and will be contained within a unit square. The resulting object is shown in Figure 4.14.

Figure 4.14: A Constructed Object Contained in a Unit Square of the Screen.



Increasing the object width to `2` and the object height to `3` results in the scaled object shown in Figure 4.15 (The xposition and yposition each retain their previous values of 2).

Figure 4.15: An Object With a Height of 3 and a Width of 2.



The particular graphical segments presented on the screen are determined in part by the temporal modifier used. The conditions for temporal modifiers use the value or values for the time attribute(s). Temporal modifications are made to objects. The particular modification depends on how time is represented. For example, if time is represented by animation, the modifier will affect the visibility of the object. The next section discusses the various representations associated with the time domain.

4.4. Representation of the Time Domain

In investigating a representation for the time domain, we considered human perception of time and the properties of the time domain in temporal databases discussed in Chapter 2. We concluded that supporting only one representation for time would be too limiting as it would not suffice for all applications. Time means different things to different people and may need to be interpreted differently for different applications.

Representing time in terms of its description involves choosing whether to emphasize succession or duration. In addition, particular representations of time can affect how time is experienced. Gradual

changes in representation make time seem to flow. More distinct changes produce a separation of events.

The time portion of our graphic display system supports the representation of time by motion techniques, geometric transformations, colorscales, intensity, time icons, and various combinations of these representations. Time can also be ignored, reducing the temporal data to static data and allowing the system to be used for the display of static databases. Because of the special nature of indeterminacy, it is represented with blinking, fading, or dashed lines. The remainder of this section discusses how these representations cause time to be perceived in different ways.

The motion techniques supported are `animation`, `animationtrace`, and `blinking`. The system also provides the user with control over the direction of the motion, the speed of the motion, and the setting of the display mode to continuous or event-by-event. Motion techniques are examples of temporal modifiers which can change a property of the objects (i.e the visibility).

Animation emphasizes succession in that it forces past, present, and future to be distinct with only one state appearing at any one time. This representation is effective for a sequence of events if the ordering of the events is important. It could also be effective for a sequence of intervals if the comparison among the starting times of the intervals is important. Animation also produces a separation of events since the changes between events is distinct. Setting the display mode to continuous or event-by-event affects how separated the events appear. Animation is not an effective representation if analysis of data requires coexistence of past, present, and future or if time should be perceived as flowing.

If succession, coexistence of past, present, and future (duration), and flowing time is important, animationtrace is a more effective representation than animation. This technique displays a trail of representations that fade over time. When time is also represented with position, termed *display aiding* [Morse 1979], the various representations are displacements which indicate position in the past. Animationtrace thus provides coexistence of past and present and even a coexistence of past, present, and future if one moves forward in time and then moves backward several states. In addition, animationtrace results in time perceived as more flowing than animation since the changes between states are more gradual.

Blinking provides a coexistence of past, present, and future by displaying all states at once. Succession is shown by blinking the "current" state. Blinking is an effective mechanism for showing valid times although it reduces readability somewhat [Morse 1979]. Time is also perceived more as flowing than with

animation since changes between states are less distinct.

A disadvantage of all three motion techniques is that representing time as motion can possibly distort one's estimation of time. As stated in Chapter 2, a person's awareness of time is based on the number of perceived changes occurring within an interval and the amount of attention paid to these changes. This finding motivated Ariav to use a time icon such as a clock face or calendar page as a reference point for the user [Ariav & Morgan 1982]. Our system also provides time icons for this reason. The various icons provided include a clock face, a digital clock, a month chart, and a text year icon. A time icon can also be used with the representation of a single event to show the valid time for the event. A time icon is an example of a temporal modifier which adds an object representation.

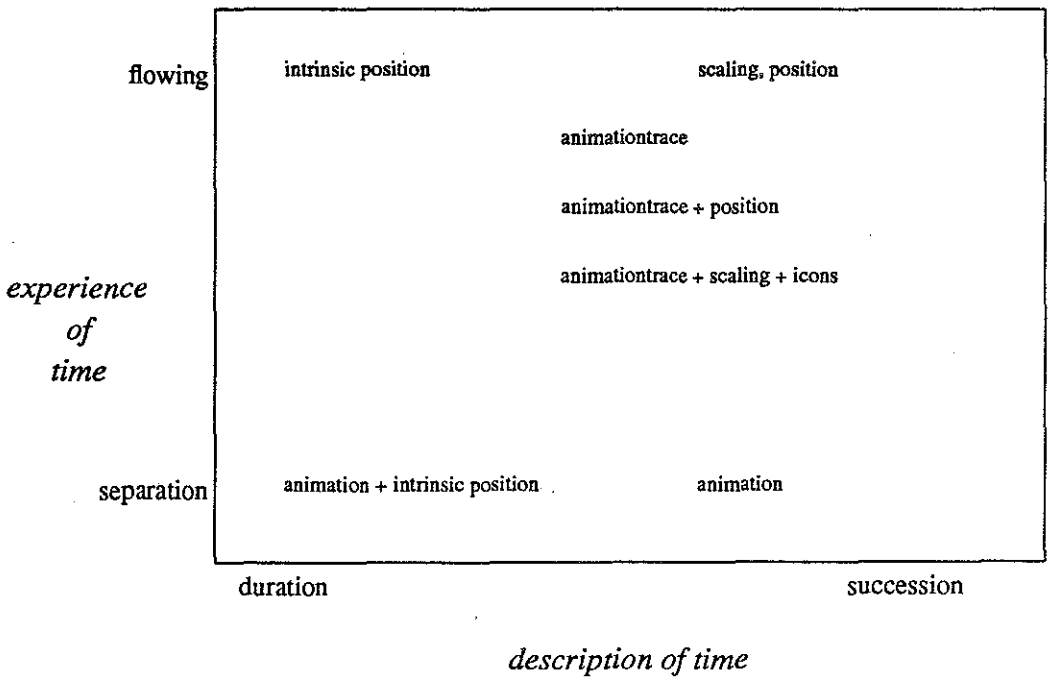
Geometric transformations as temporal modifiers include scaling, rotation, and position. These techniques used without motion techniques provide a coexistence of past, present, and future since all states are displayed at once. Scaling and rotation are effective temporal modifiers if the comparison between objects is more important than the actual time of each object because it is difficult to show how a time value is assigned to a scaling or rotation increment. An example of a scaling representation is where objects in the "present" are larger than objects in the "past". An example of a rotation representation is where objects which are more upright are closer to the present than objects which are more horizontal.

Position as a temporal modifier can be used both to position objects along the *x* and/or *y* axes and to change the intrinsic structure of objects. Another use of position is to position objects along the *z* axis providing a three dimensional representation where present events are closer to the user then previous events [Ariav & Morgan 1982]. Three dimensional representations are discussed as an extension in Chapter 12. Positioning an object involves changing the object's graphical characteristics `xposition` and/or `yposition` using the time value to determine the position value. Using position to change the intrinsic structure of an object, termed *intrinsic position modifier*, involves mapping the time value to an integer value and then using this integer value as the x or y coordinate of of an iconic representation. If the time domain type is an interval, then this representation can be effective for showing the duration of the interval. For example, the starting and stopping time values can be mapped to integer values and used as the x coordinates of a line. The difference in x values shows the duration. This technique is used in the example of Chapter 6.

Colorscales and intensity as temporal modifiers like scaling and rotation are more effective for comparing objects than showing actual times of objects. An example of a colorscale modifier is a gray scale where lighter shades depict early afternoon hours and darker shades depict late afternoon and evening hours. An example of an intensity modifier is where "current" events have increased intensity and previous events are faded. This is suggested as a representation in the MDM/GI [Ariav & Morgan 1982]. Both colorscales and intensity create an illusion of flowing since changes between states are gradual.

Various combinations of the above representations are possible. Already mentioned are the combination of motion techniques and time icons to provide reference points for motion and animationtrace and position to provide a trail of displacements over time. Other combinations affect the emphasis of duration or succession and the perception of time as flowing or as a separation of events. Especially interesting are combinations where one representation is a motion technique. For example, a combination of animation and intrinsic position can emphasize duration while making time be perceived as a separation of events. The chart of Figure 4.16 shows how certain representations and combinations of representations affect the perception of time.

Figure 4.16: Variations in the Perception of Time



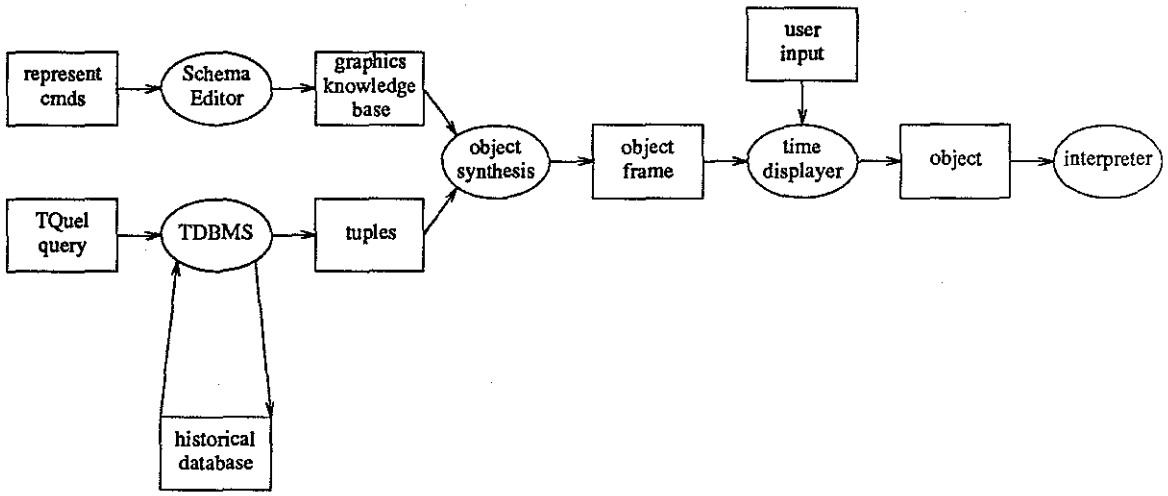
Combinations can also alleviate deficiencies in single representations, for example, time icons provide reference points for motion techniques. Another example is a combination of animationtrace, scaling, and a time icon. A disadvantage of scaling as a temporal modifier is that the actual time of the object is not graphically portrayed. An advantage of scaling is that it is effective for showing comparison between states. A combination of scaling and animationtrace with a time icon gives a reference point for time while providing two representations, increased size and intensity, to indicate "current" objects more effectively.

The final representation decision concerns indeterminacy. Because of the special nature of indeterminacy, it is represented with blinking, fading, or dashed lines. The "possibly-valid" and "possibly-invalid" intervals of the three interval time domain will be represented with blinking, a faded color, or dashed lines. The valid interval will be represented in the normal fashion. Alternatively, indeterminacy can be ignored if indeterminate data need not be distinguished from true data.

4.5. The Structure of the System

The general structure of the prototype display system is shown in Figure 4.17. Rectangles represent data and ellipses represent program modules.

Figure 4.17: Prototype Display System Architecture



The technique used to associate icon representations and graphical characteristics with object templates and modifying representations with attributes is based on that proposed by Friedell [Friedell 1984]. As

mentioned in Chapter 3, Friedell describes a technique for automatically synthesizing graphical object descriptions from a high level specification which gives the identity of the object template and the identity and value of its important descriptive attributes. This technique was chosen because it provides an effective mechanism for associating graphical representations with data. A *Graphics Knowledge Base* holds the information necessary for the synthesis of the object given its type and attribute values. The *Object Synthesis* module builds the object description from the information in the knowledge base and from the tuples returned by the TDBMS. The object descriptions are collected into a structure called the *object frame*. The object frame is passed to the *Time Displayer* which applies temporal modifiers and passes each object to the *Interpreter*. The *Interpreter* interprets the graphical code in the object and makes the appropriate calls to the low level graphic routines to display the object. The Graphics Knowledge Base, the Object Synthesis phase, the Time Displayer, and the Interpreter will be further described in Chapter 8.

A limitation of Friedell's technique is that it doesn't provide a mechanism for dynamically generating representations for new entities. We extended this technique to allow for new representations and also to allow for changes to existing representations. This is done by using a *Schema Editor* which allows the user to interactively specify the graphical representation of the object template and of its attributes. The Schema Editor also allows the user to specify a representation for time and for indeterminacy. The Schema Editor is used to initially construct and to modify the Graphics Knowledge Base structure. A major advantage of this technique is that it allows the user to view the same data in several ways and to specify representations for new relations. The syntax and semantics of the Schema Editor commands are discussed in more detail in Chapter 8.

The TQuel queries on the temporal database are processed by a temporal database management system (TDBMS). One such system exists [Snodgrass & Ahn 1986]. The resulting tuples are collected and sent to the Object Synthesis module.

The next part steps through several examples illustrating the approach defined in this chapter. Several representations of static data and time are shown.

Comprehensive Examples

In this part, we will step through the process of representing and displaying relations from several different temporal databases. Chapter 5 describes a temporal database for an aeronautics school. It then gives examples of two different representations for time in this context which are useful for different applications. It next gives a representation for indeterminacy. Chapter 6 describes a temporal database which holds instances of plays from a football game. In this example, time is shown to be best represented with position. Finally, Chapter 7 describes a temporal database for a simple monitoring system. This section gives examples of relations which use or change the depiction of other relations. Time is represented with both animation and animationtrace. In the examples, sample commands are provided; we discuss the syntax and define the command language in Chapter 9.

CHAPTER 5

Spartan School of Aeronautics

This example steps through the procedure of graphically representing and displaying an example query. Each relation is given an iconic representation and each attribute is given a modifying representation. For this query, time is represented in two ways. First, it is represented with animation. This representation is augmented by also adding a clock icon giving a reference point for the animation. The second representation for time is position. This representation is augmented with a labelled time line. Representing time with position enables two states to appear simultaneously supporting more effective comparison between the two. The final example gives a representation for indeterminacy.

5.1. The Database

Spartan is a fictitious school of aeronautics which provides pilot training to anyone wishing to obtain a pilot's license. The administration of the school maintains a historical database consisting of three interval relations. The first relation describes the school's inventory of planes.

```
airplaneinventory ( plane, make, model, year)
```

The `plane` attribute defines the unique name the school has given the airplane. This serves as the key for the relation. The `make` attribute names the company which built the airplane. Spartan has airplanes made by `Piper`, `aerospatiale`, `Cessna`, and `Beechcraft`. The `model` attribute gives the company model name for each plane. The `year` attribute states the year in which the airplane was built. The `airplaneinventory` relation is a historical interval relation. The implicit `from` attribute of the interval is the time in which the school purchased the plane. The implicit `to` attribute of the interval is the time in which the school sold the plane or "forever" if the plane has not been sold.

The next interval relation keeps a log of activities for the school.

```
log ( plane, status)
```

The `plane` attribute gives the name of the airplane. The `status` attribute gives the status at the time specified in the implicit time interval. The possible values are `demoflight`, `trainingflight`, `freeflight`, `onground`, and `inrepair`. The key of this relation is the `plane` attribute.

The third relation keeps track of the times each pilot or pilot-to-be has flown. It is also an interval relation.

```
flights ( pilot, plane)
```

The `pilot` attribute gives the name of the pilot. The `plane` attribute specifies the particular plane used. The implicit time interval gives the interval of time the pilot was in flight. The key of this relation is the `pilot` attribute.

5.2. Static Representation for an Example Query

Suppose we wanted to know the status of the planes between 8:00 AM and 12:00 PM on Sept. 9, 1985. In addition, we are interested in the model of each of the planes. This information is obtained using the temporal query language TQuel.

```
range of l is log
range of i is airplaneinventory
retrieve into airplanestatus (l.plane, i.model, l.status)
      where l.plane = i.plane
      valid from "9/9/85 0800" to "9/9/85 1200"
```

The tuples returned are shown in Figure 5.1.

Figure 5.1 : An Example Query

airplanestatus (plane, model, status):

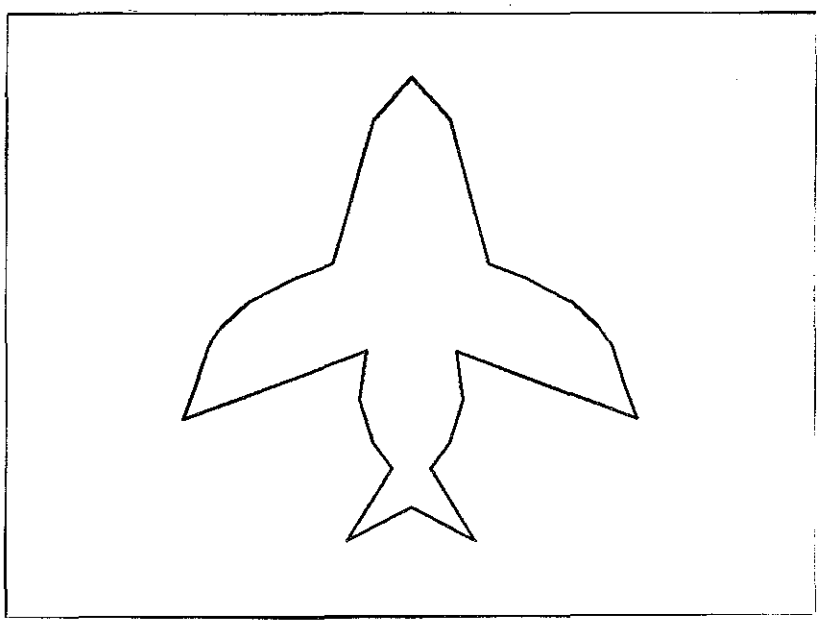
plane	model	status	from	to
CessI	414A	inrepair	0800	1030
CessII	414A	demoflight	0800	1000
PiperI	ArcherII	onground	0800	0900
PiperII	ArcherII	trainingflight	0800	0900
CessI	414A	onground	1030	1200
CessII	414A	onground	1000	1100
CessII	414A	trainingflight	1100	1200
PiperI	ArcherII	freeflight	0900	1200
PiperII	ArcherII	inrepair	0900	1200

Next we determine a representation for the tuples of the relation. This is accomplished using the following *Schema Editor* commands.

```
range of a is airplanestatus
represent a with icon plane position 10,10 size 80,80
```

This icon is a generic plane outline which can be used for all of the different planes. This icon is shown in Figure 5.2.

Figure 5.2 : The Plane Icon



The lower corner of the icon is positioned at `10,10` in the object coordinate system and the icon is scaled by `80` in both the x and y directions.

We differentiate between the two models using additional icons by associating these icons to the particular values for the attribute `model`. For example, the `Cessna 414A` has propellers on its wings (Figure 5.3), while the `Piper Archer II` has a nose propeller (Figure 5.4). These are specified with the following commands.

```
represent a.model=414A with icon wingpropeller position 30,35 size 5,30
                          icon wingpropeller position 65,35 size 5,30

represent a.model=ArcherII with icon nosepropeller
                                position 45,85 size 10,5
```

Figure 5.3 : Additional Icon Representation for Model Cessna 414A

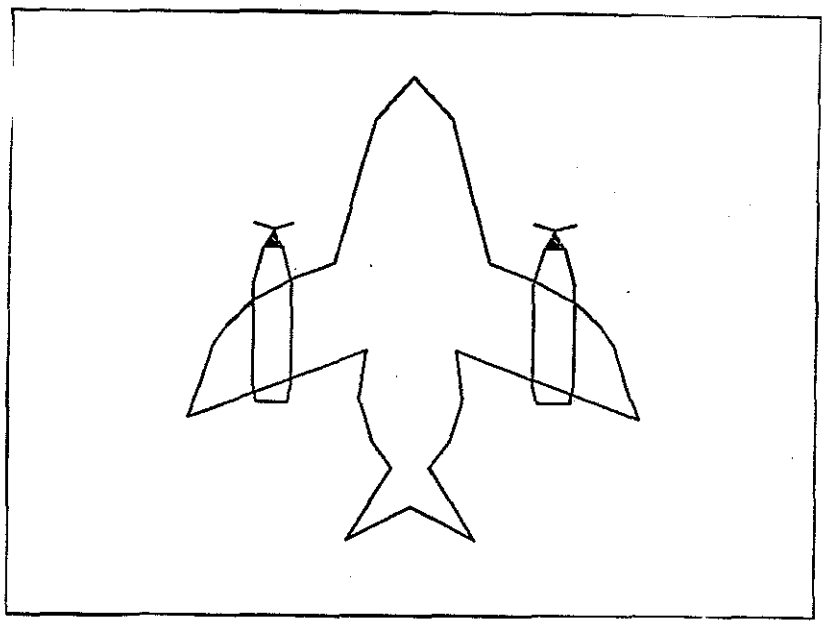
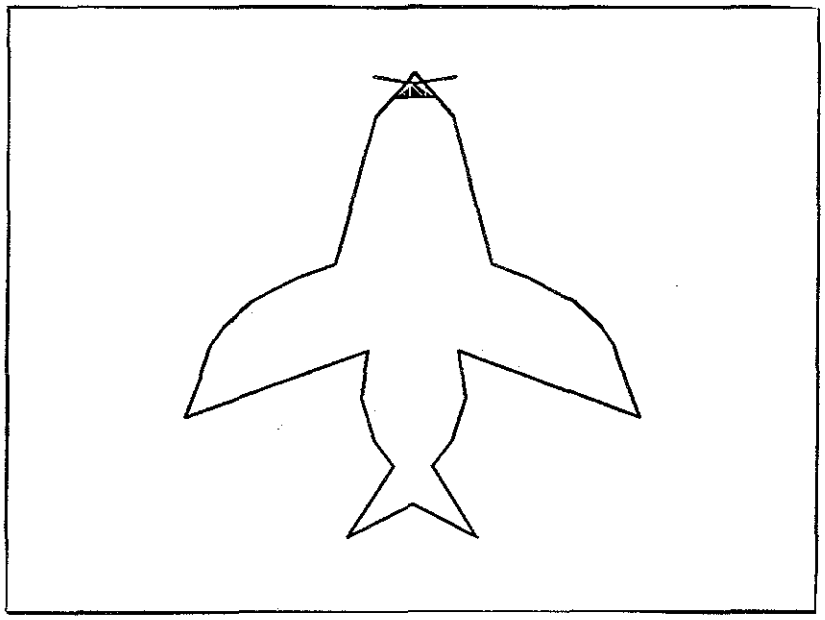


Figure 5.4 : Additional Icon Representation for Model Piper ArcherII



Next, we give a representation for the attribute `status`. To differentiate between the different values, we represent the attribute with *y* position on the screen.

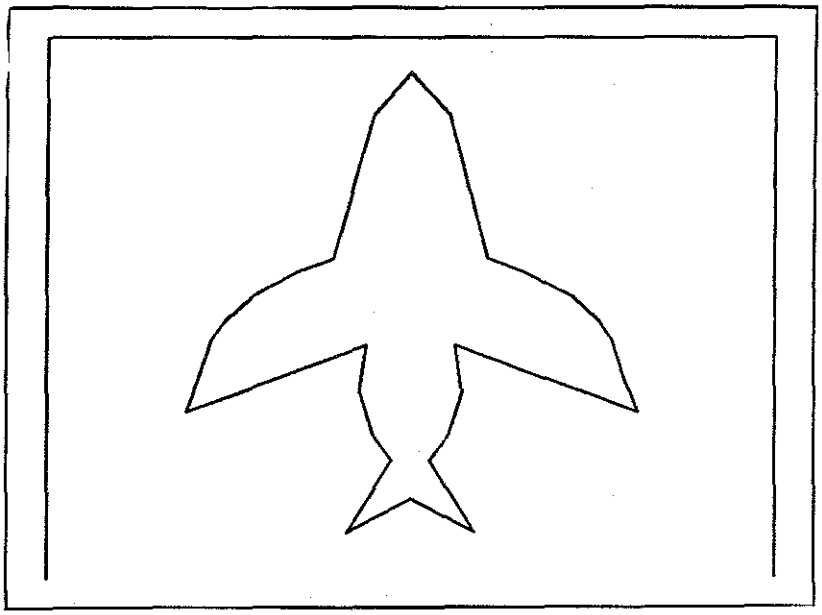
```
represent a.status = inrepair       with yposition 1
represent a.status = onground       with yposition 1
represent a.status = trainingflight with yposition 2
represent a.status = demoflight     with yposition 3
represent a.status = freeflight     with yposition 4
```

This means that an airplane being repaired would be placed at a yposition of `1` in screen coordinates.

Since the attribute-value pairs `status = inrepair` and `status = onground` are both associated with the yposition 1, we also associate the attribute-value pair `status = inrepair` with the icon `garage` (Figure 5.5).

```
represent a.status = inrepair with icon garage
                                position 5,5 size 90,90
```

Figure 5.5 : Additional Icon Representation for `status = inrepair`



The last attribute to represent is `plane`. Since there is an infinite number of possible values for this attribute, we represent it with `text`. This will place a text icon with the value of the `plane` attribute at the object coordinate position 45,50. Each character of the text will have a width and height of 4 in object coordinates.

```
represent a.plane with text at 37,50 size 4,4
```

We also need a way to position the planes along the x-axis. We accomplish this by giving an additional representation to attribute `plane` of `xposition`.

```
represent a.plane with xposition range 1 to 5
```

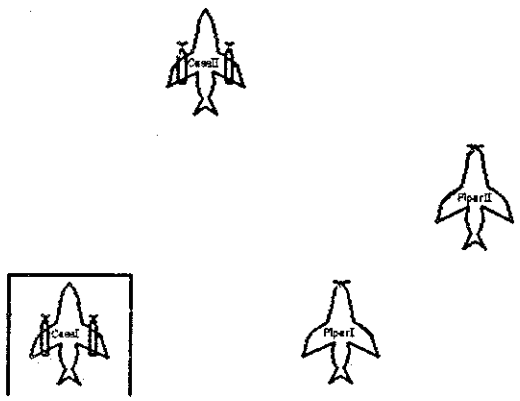
The tuples will be ordered alphabetically by name and positioned between `1` and `5` along the x-axis.

This completes the static representation for the relation. The screen coordinates are now set.

```
set screen.coordinates to 0,0 to 8,6
```

Figure 5.6 shows the resulting display for 8:00 AM.

Figure 5.6 : The Static Representation at 8:00 AM



In this figure, each object is represented with the generic plane outline. The objects are ordered along the x-axis by the value of the `plane` attribute. This value is also displayed as a text icon at position 45,50 within each object's coordinate system. The y position of each object is set by the value of the `status` attribute. The additional icon `garage` is added to the representation for the value of `inrepair`. The wing propeller icon is added for objects with a value of `414A` for the `model` attribute. The nose propeller icon is added for objects with a value of `ArcherII` for the `model` attribute.

5.3. Representing the time domain

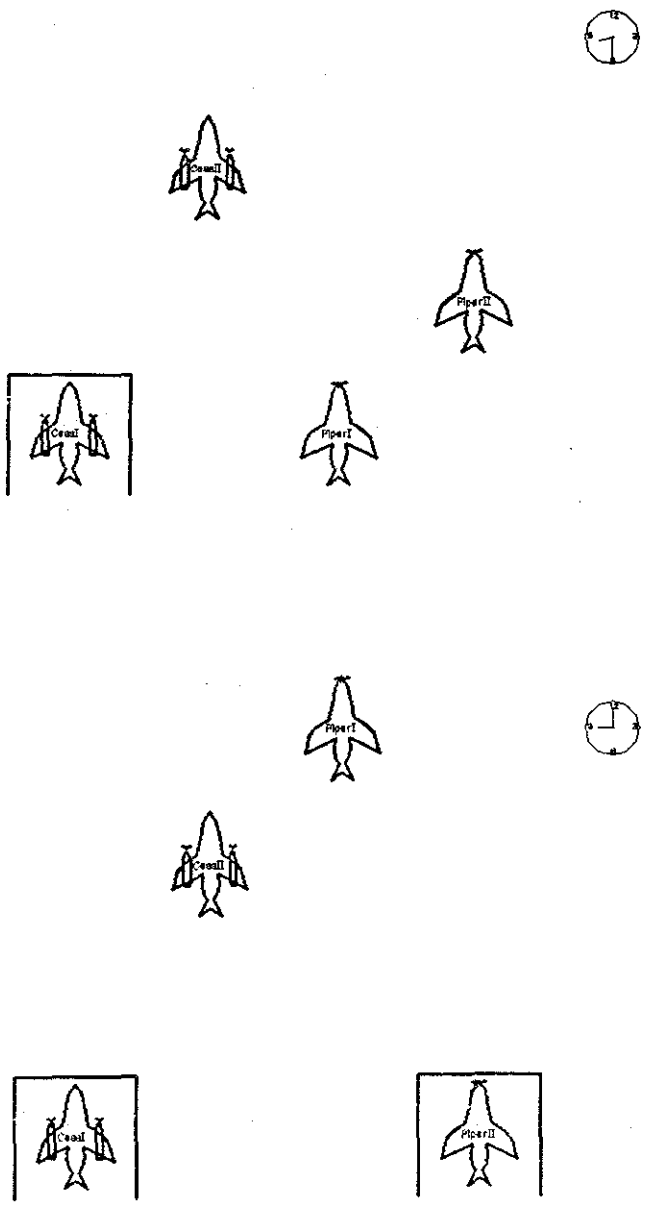
While this representation is adequate for a single static relation, it does not suffice for representing one or more event relations, interval relations, or points in time of an interval relation. We next give two different examples of representations for the time domain of the relation `airplanestatus`.

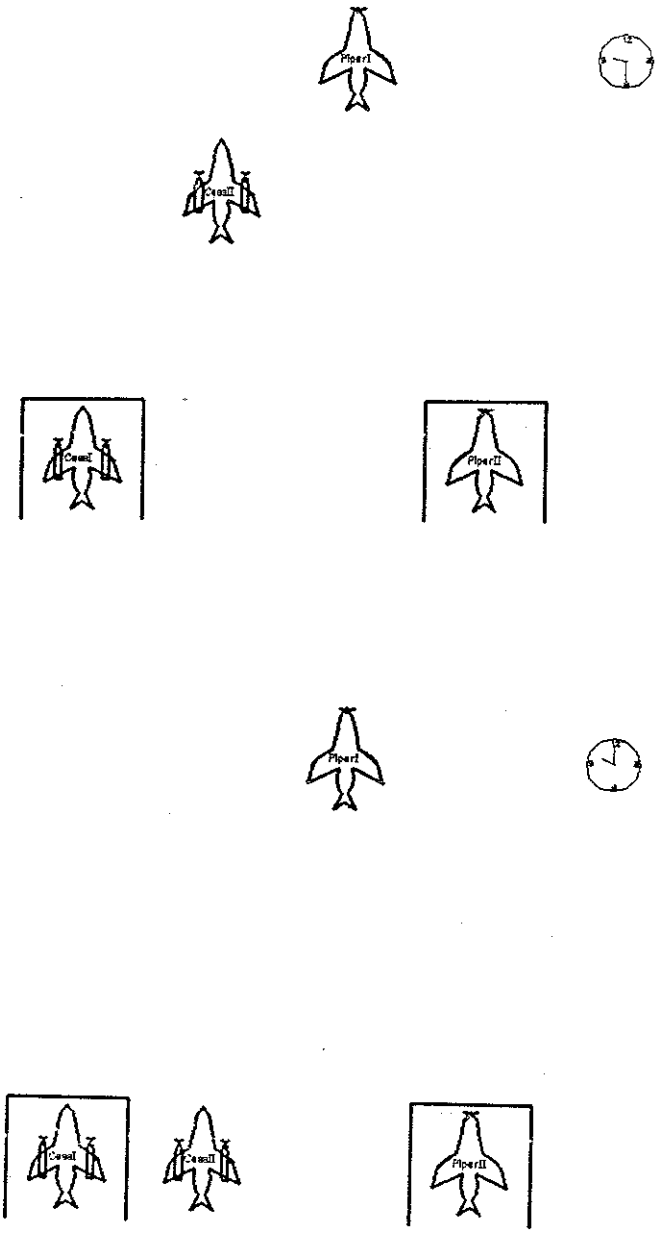
First we represent time with animation.

```
represent time with animation step = 30 minutes mode = stop
                              time_icon clockface position 4,4 size 1,1
```

We set the step size to `30` minutes since this is a good division for our example query. The mode is set to `stop` so that the animation will halt at each step. We augment the time representation with the clockface icon. `4,4` on the screen. Figure 5.7 gives the progression of four states from 8:30 AM to 10:00 AM.

Figure 5.7 : Progression of States Using Animation and Clock Icon to Represent Time





The clock is positioned at 4, 4 on the screen and is the size of a unit square on the screen The hands of the clock point to the corresponding time. If we think of the state at 10:00 AM as a single event relation, the time icon serves to distinguish this event from static data. Thus, the event exists in time. In addition, the icon represents time when there are no other events to depict.

Using animation to represent time for the `airplanestatus` relation emphasizes succession of states. Past, present, and future are distinct because only one state exists at any one time. Animation also produces a separation of events since the changes between events are distinct. Animation is an effective representation for time in the `airplanestatus` relation if the comparison among the starting times of each status is important. However, animation is not effective if an analysis of the data requires a coexistence of past, present, and future. Given one state, it is hard to remember the previous state. Animation is also not effective if we are interested in the duration of each interval. The next representation shows an alternate representation for time which emphasizes duration.

Suppose we are interested in the status of `CessII` between 8:00 AM and 12:00 PM. In particular, we would like to compare how long `CessII` was in each status. We select the appropriate tuples using the TQuel command:

```
range of a is airplanestatus
retrieve (a.plane, a.model, a.status)
    where a.plane = "CessII"
```

The resulting tuples are shown in Figure 5.8.

Figure 5.8 : Status of CessII

airplanestatus (plane, model, status):

plane	model	status	from	to
CessII	414A	demoflight	0800	1000
CessII	414A	onground	1000	1100
CessII	414A	trainingflight	1100	1200

We represent the static portions of the data as before. We change the representation of time to xposition:

```
represent time with xposition step = 30 minutes
```

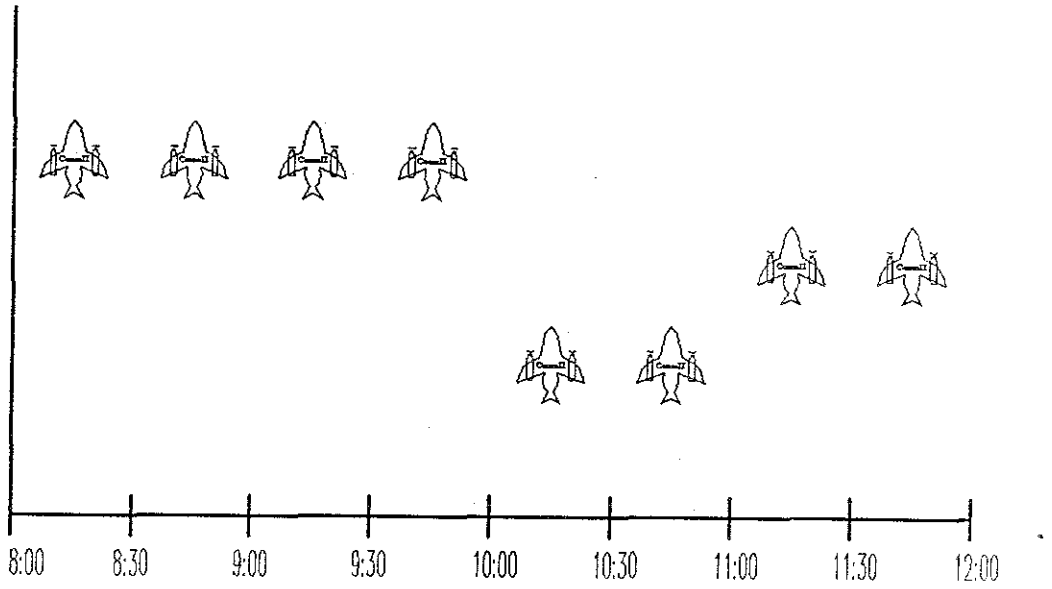
This representation of time changes the x axis of the screen to a time line. The time line has increments of 30 minutes, the value of the step size. The least time value is 480 minutes, corresponding to 8 hours multiplied by 60 minutes/hour. To provide a reference point for time, we represent the background with a labelled time line. The entire description is not given here

```
represent background with line 480,0 to 720,0
                           line 480,0 to 480,6
```

```
text "8:00" at 480,-1 size 2,2
text "8:30" at 510,-1 size 2,2
.
.
.
```

Figure 5.9 shows the resulting representation.

Figure 5.9: Representation for Status of CessII



Each increment on the x axis represents a time interval of 30 minutes. The yposition indicates status: yposition `3 = demoflight`; yposition `2 = trainingflight`; yposition `1 = onground`. For example, `CessII` had a status of `demoflight` for 4 x 30 minutes or two hours.

Xposition as a representation of time as shown in Figure 5.10 emphasizes the duration of each interval by providing a coexistence of past, present, and future. However, the representation also shows succession. Within the duration between 8:00 AM and 12:00 PM is the image of `CessII` in successive statuses along the time line, thus giving a description with both duration and succession. If we think of each state as a slice of width 30 minutes parallel to the *y* axis, time is perceived as a separation of events since the changes between two states is distinct.

5.4. Representing Indeterminacy

The third interval relation, `flights`, sometimes contains incorrect information because the school is not always careful about recording the exact starting and stopping time of a pilot's flight. Therefore, data returned from a query on `flights` is considered as indeterminate.

Suppose we are interested in pilot `Jackie`'s flight on `May 16, 1985`. The TQuel query is:

```
range of f is flights
retrieve into Jflights ( f.plane)
    where f.pilot = "Jackie"
    valid from "5/16/85 00:00" to "5/16/85 24:00"
```

The resulting tuples are shown in Figure 5.10. To represent indeterminacy, the `from` and `to` "timestamps" are stored as intervals.

Figure 5.10: A Query on Relation flights

Jflights (plane):

plane	from	to
CessII	0900, 0930	1100, 1130
PiperI	1300, 1330	1500, 1530

Jackie flew twice on May 16. In the first flight, he took off sometime between 9:00 and 9:30 AM and returned sometime between 11:00 and 11:30 AM. In the second flight, he took off sometime between 1:00 and 1:30 PM and returned sometime between 3:00 and 3:30 PM.

We represent the relation `Jflights` with the icon `plane` and the attribute `plane` with text.

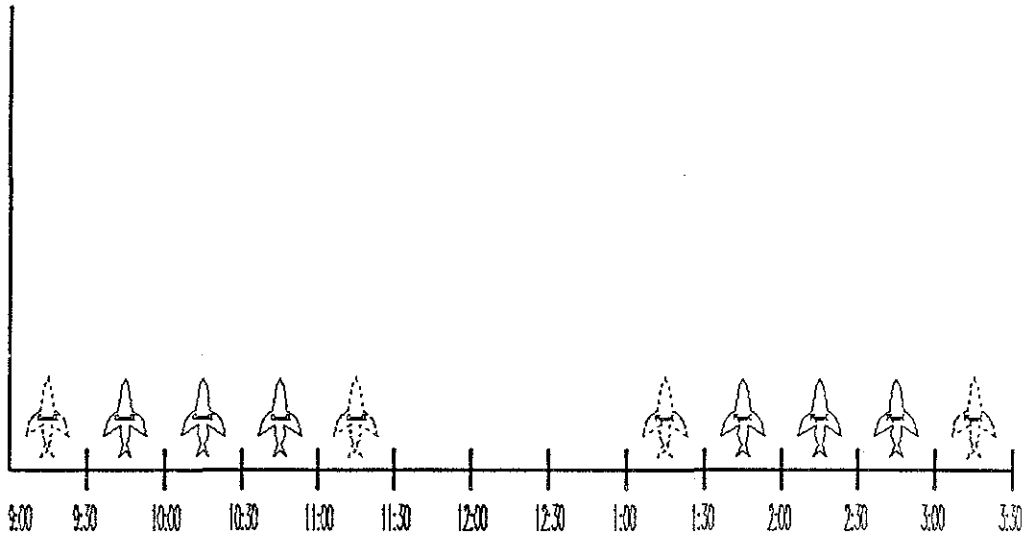
```
range of f is Jflights
represent f with icon plane position 10,10 size 80,80
represent f.plane with text at 38,50 size 4,4
```

We are interested in the minimum flight time, therefore we need to distinguish indeterminate data from true data. In addition, we are interested in the time between flights and the length of each flight. Thus, we represent indeterminacy with dashed lines and time with xposition with a step of 30 minutes.

```
represent indeterminacy with dashedlines
represent time with xposition step = 30 minutes
```

The background representation is a labelled time line where each interval represents 30 minutes. The resulting representation is shown in Figure 5.11.

Figure 5.11: Representation of Indeterminacy



From the representation shown in Figure 5.11, it is easy to determine the minimum and maximum flight time. In addition, the background serves as a reference point for time.

CHAPTER 6

Football Plays

This example shows another technique for representing time with position. This technique uses the values of the starting and stopping time as part of the object description. Three variations are shown. The first plots lines against a time axis and the second plots a bar graph against a time axis. The third positions icons using the time axis. These representations are useful when one is interested in the *duration* of each tuple's time interval rather than *when* each tuple is valid. Once again the relation is given an iconic representation and one attribute is given a modifying representation. The second attribute is used with the time values as part of the object description.

6.1. The Database

A high school coach wants to determine which offensive football plays have been most successful for the season. Information about the plays for each game is kept in two historical interval relations.

The first relation lists the plays for each game.

```
plays ( game, typeplay, startpos, stoppos)
```

The attribute `game` gives the number of the game in the season. The domain for this attribute is integers between 1 and 10. The values for attribute `typeplay` can be `running`, `forwardpass`, and `lateral`. The attribute `startpos` and `stoppos` give the field position in yards just before and just after the play. The implicit time interval gives the starting and stopping time for the play. The key for the relation is a combination of the `game` and the `from` attributes.

The second relation lists the time used for the touchdowns in a game.

```
touchdowns (game, number)
```

The game attribute gives the number of the game in the season. The attribute number uniquely identifies each touchdown. The implicit time interval gives the time used for all the plays leading to the touchdown. The key for the relation is a combination of the game and the number attributes.

6.2. Representations

Suppose the coach wanted to compare the plays in the first two minutes of the first game. This appropriate information is obtained using TQuel.

```
range of p is plays
retrieve into firstgameplays(p.typeplay, p.startpos, p.stoppos)
      where p.game = 1
      valid from "0:00" to "2:00"
```

The resulting tuples are shown in Figure 6.1.

Figure 6.1 : First Game Plays

firstgameplays (typeplay, startpos, stoppos):

typeplay	startpos	stoppos	from	to
running	20	26	00:00	00:30
lateral	26	24	00:30	00:40
forwardpass	24	31	00:40	00:53
forwardpass	31	35	00:53	01:14
forwardpass	35	35	01:14	01:25
lateral	35	45	01:25	01:52
running	45	48	01:52	02:00

The comparison is concerned with the *duration* of each play rather than the starting time of each play. For this reason, time is represented with intrinsic position.

```
represent time with intrinsic xposition step = 1 second
```

Intrinsic position as a modifier involves mapping the time value to an integer and then using this integer value as the *x* or *y* coordinate of an iconic representation. The mapping is specified by the step. In this example, the step is 1 second so all times will be mapped to an integer value equal to the number of seconds. In addition, the x axis of the screen is set to a time line with an interval step of one second.

Each tuple of the relation is represented with a line using the values of starting and stopping time as the x coordinates and the values of the starting and stopping position as the y coordinates.

```
range of f is firstgameplays
```

```
represent f with line (f.from,f.startpos) to (f.to,f.stoppos)
```

The attribute `typeplay` is used to modify the tuple representation with intensity.

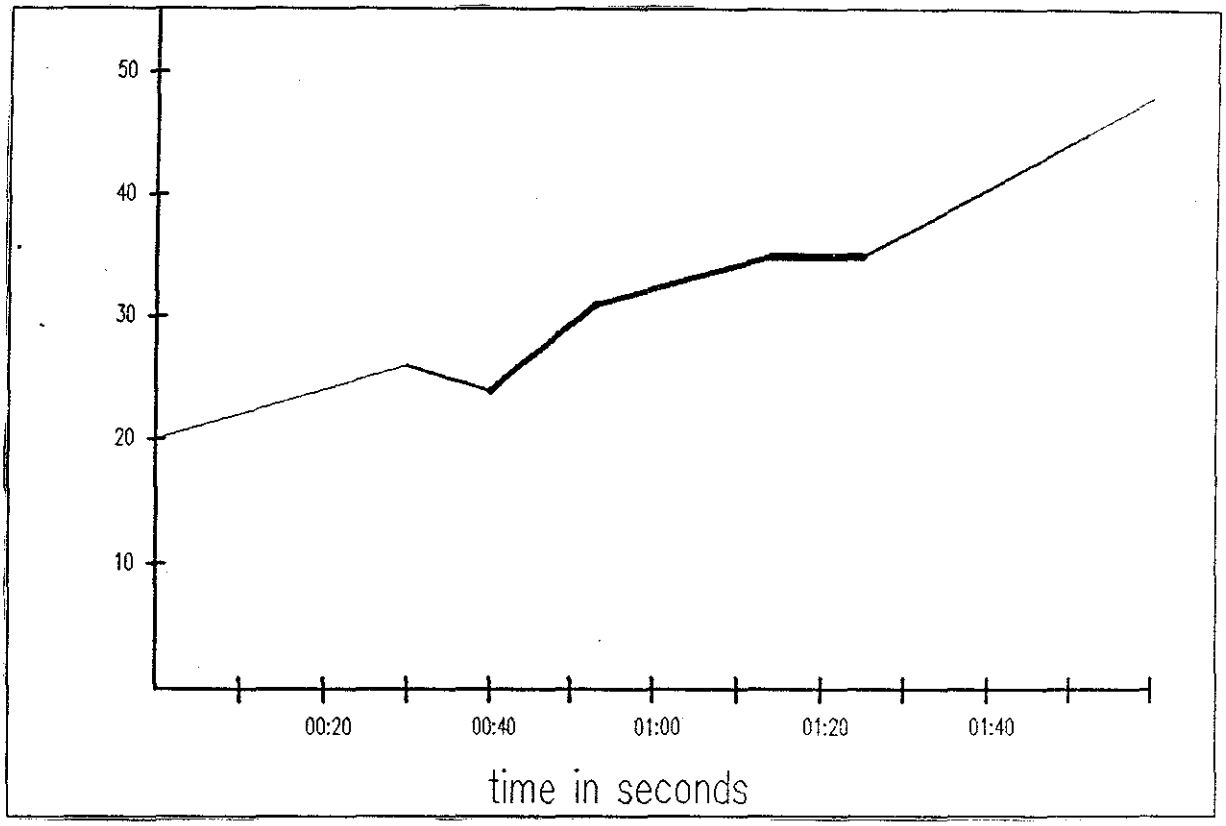
```
represent f.typeplay = forwardpass with intensity 1
represent f.typeplay = lateral     with intensity .6
represent f.typeplay = running     with intensity .2
```

A background representation can be given with the same syntax as object representations. For this example we use a coordinate axis with time as the x-axis and position as the y-axis. The complete description is not given here.

```
represent background with line 0,0 to 100,0
                           line 0,0 to 0,100
                                .
                                .
                                .
```

The resulting representation is shown in Figure 6.2.

Figure 6.2 : Representation Using Lines Plotted Against a Time Axis



Representing time with intrinsic position emphasizes duration rather than *when* each play began. In Figure 6.2, the length is shown by the x distance of each line. Plotting lines against a time axis also gives

the illusion of flowing time since the changes between successive states are gradual. That is, the changes in representation over time are not distinct.

Suppose the coach then wanted to compare the total distance gained from each play with the total time taken and the time of the game. The TQuel query used to obtain the information would be

```
range of p is plays
retrieve into playsdistance(p.typeplay, totaldistance=p.stoppos-p.startpc
      where p.game=1
      valid from 0:00 to 2:00
```

The resulting tuples are shown in Figure 6.3.

Figure 6.3 : Total Distance from Plays

playsdistance (typeplay, totaldistance):

typeplay	totaldistance	from	to
running	6	00:00	00:30
lateral	-2	00:30	00:40
forwardpass	7	00:40	00:53
forwardpass	4	00:53	01:14
forwardpass	0	01:14	01:25
lateral	10	01:25	01:52
running	3	01:52	02:00

Again the x axis of the screen is set to a time axis with an interval step of one second.

```
represent time with intrinsic xposition step = 1 second
```

In this example, the comparison is made using a bar chart. Each tuple of the relation is represented with a polygon using the `start` and `stop` time attributes as the x coordinates, the total distance as the upper y coordinate, and 0 as the lower y coordinate.

```
range of p is playsdistance
represent p with polygon
      (p.from, p.totaldistance) (p.to, p.totaldistance)
      (p.to, 0) (p.from, 0)
```

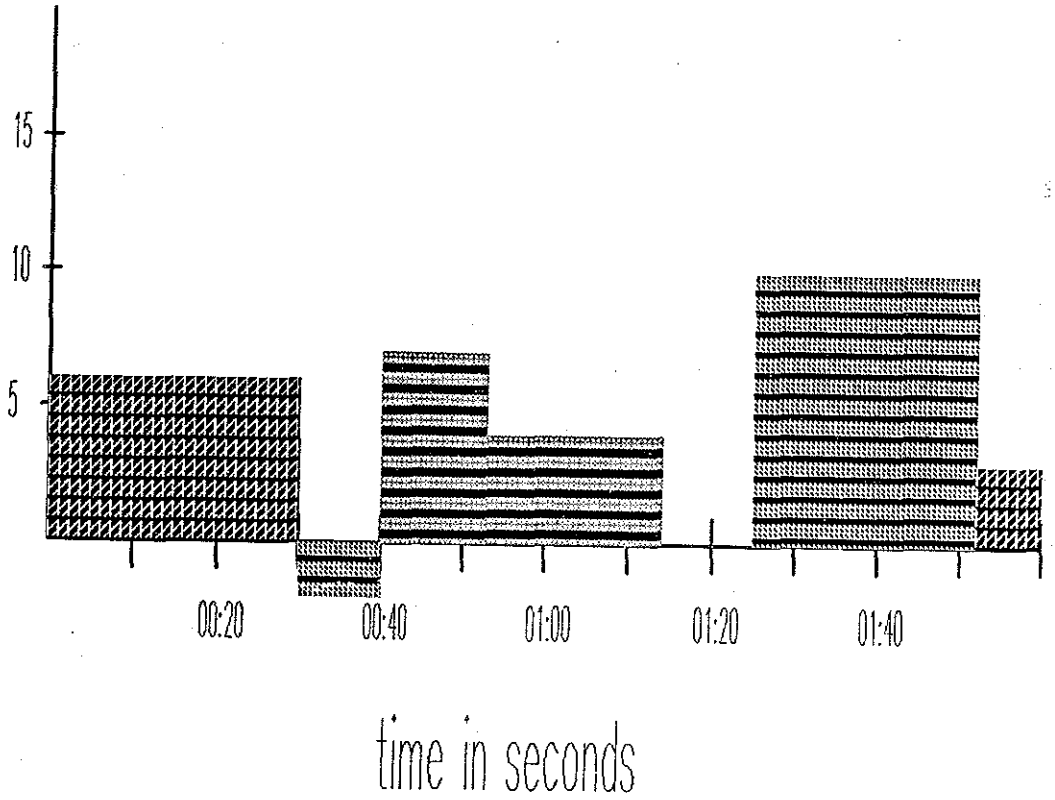
A polygon is described by listing the vertices in either clockwise or counter-clockwise order. The vertices are connected in the order given with the last vertex being connected to the first.

The attribute `typeplay` modifies the object representation with color.

```
represent p.typeplay = forwardpass with color blue
represent p.typeplay = lateral     with color red
represent p.typeplay = running     with color green
```

The background representation is the same as in the last example. The resulting representation is shown in Figure 6.4. Color is shown with texture.

Figure 6.4 : Representation Using A Bar Chart Plotted Against a Time Axis



Representing distance plotted against a time axis stresses the duration of each play. The duration is given by the width of each bar. In addition, the total distance of each play is emphasized by representing this with the height of each bar.

In both Figure 6.2 and Figure 6.4, time is represented similarly to an infinite attribute. Both examples emphasize duration. Succession is shown by successive points along the time line. The next example shows both duration and succession, emphasizing succession rather than duration.

Suppose the assistant coach comes along and wants to look at the plays for the first touchdown of the first game. The TQuel query used is.

```
range of p is plays
range of t is touchdowns
retrieve into firsttouchdown_plays(p.typeplay, p.startpos,
                                   distance = p.stoppos-p.startpos)
      where p.game = 1  and t.game = 1 and t.number = 1
```

```
valid from t.from to t.to
```

The resulting tuples are shown in Figure 6.5.

Figure 6.5 : First Touchdown Plays

firsttouchdown_plays (typeplay, startpos, distance):

typeplay	startpos	distance	from	to
running	55	7	02:45	03:10
forwardpass	62	20	03:10	03:20
lateral	82	8	03:20	03:30
running	90	2	03:30	03:33
lateral	92	3	03:33	03:36
forwardpass	95	5	03:36	03:39

The assistant coach decides to display the information similar to a representation often seen on television. For this representation, each value of `typeplay` is represented with a different icon. The icon is positioned using the value of the `startpos` attribute as the x-coordinate and the value of the `from` attribute as the y-coordinate. The icon is scaled in the x direction by the value of the `distance` attribute.

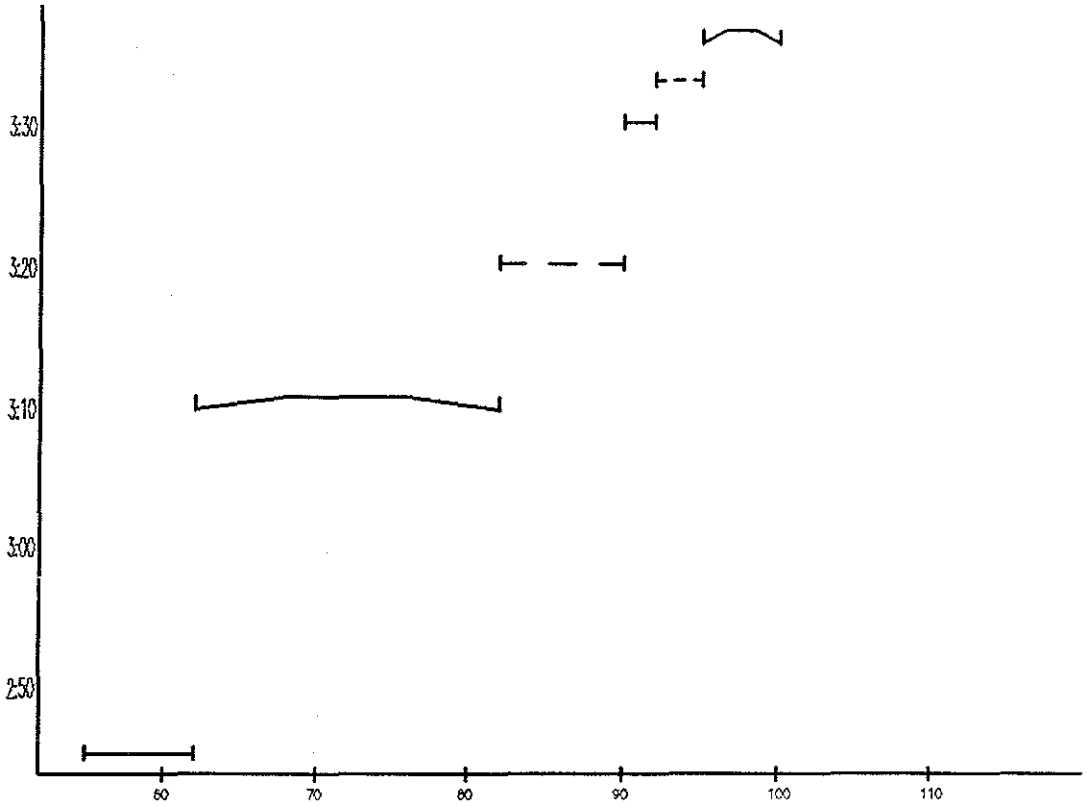
```
range of f is firsttouchdown_plays
represent f.typeplay = running with icon barbell
            position (f.startpos, f.start) size (f.distance, 1)
represent f.typeplay = lateral with icon dotted_barbell
            position (f.startpos, f.start) size (f.distance, 1)
represent f.typeplay = forwardpass with icon curved_barbell
            position (f.startpos, f.start) size (f.distance, 1)
    setting f.width = 70
            f.height = 54
```

In addition, time is represented with intrinsic yposition. The step interval is 1 second.

```
represent time with intrinsic yposition
          step = 1 second
```

The resulting display is shown in Figure 6.6.

Figure 6.6 : Succession of States Using Icons



In this representation, time is used to position each object representation. Therefore, this example emphasizes the starting time of each tuple rather than the duration of each tuple. However, the duration is shown by the vertical distance between icons. As in the other example, time is represented with the same technique as an infinite attribute.

CHAPTER 7

A Simple Monitoring System

This example shows a technique for representing relationship relations. The representation involves using the representation of defined entity relations. Time is represented with both animation and animation-trace. Both time representations use the additional representation of a digital clock icon.

7.1. The Database

This database describes simple monitoring for an operating system. This example is taken from a paper on monitoring complex systems [Snodgrass 1985]. Details on how the data is gathered are not important for representation purposes.

The database consists of three entity relations and three relationship relations. All of the entity relations are historical interval relations.

The first entity relation describes the characteristics of a process.

```
process (id, state)
```

The `id` attribute is the process identifier. The `state` attribute can be `Ready` (the process is scheduled but not currently running), `Running` (the process is currently running on a processor), `Blocked` (the process is waiting on a mailbox), or `Done` (the process has halted or aborted). The implicit time interval gives the time that each process is in the corresponding state.

The next entity relation describes the characteristics of a processor.

```
processor (id)
```

The processor relation contains only one attribute, `id`, which gives the processor identifier. The implicit time interval gives the lifetime of the processor.

The third entity relation describes the characteristics of a mailbox.

```
mailbox (id)
```

This relation also contains only one attribute, `id`, which gives the mailbox identifier. The implicit time interval indicates when a process has access to the mailbox.

The first relationship relation describes which process is running on which processor. This relation is an interval relation.

```
runningon (process, processor)
```

The `process` attribute is the process identifier. The `processor` attribute is the processor identifier. The implicit time interval gives the time that the process is running on the corresponding processor.

The next relationship relation is an event relation which records the instantaneous time that a process sends a message to a mailbox.

```
sendmessage (process, mailbox)
```

The `process` attribute is the process identifier. The `mailbox` attribute is the mailbox identifier. The implicit time event indicates when the process sent a message to the mailbox.

The third relationship relation lists the processes blocked while waiting to receive from a mailbox. This relation is an interval relation.

```
waiting (process, mailbox)
```

Again, the `process` attribute gives the process identifier and the `mailbox` attribute gives the mailbox identifier. The implicit time interval gives the time that the process is blocked.

The tuples for the entity relations are shown in Figure 7.1. The tuples for the relationship relations are shown in Figure 7.2.

Figure 7.1 : Entity Relations of the Monitoring System Database

process (id, state):

id	state	from	to
P1	Ready	1:00:00	2:00:00
P2	Ready	1:23:24	2:05:12
P1	Running	2:00:00	2:15:37
P2	Running	2:05:12	2:45:29
P1	Ready	2:15:37	2:45:30
P2	Blocked	2:45:29	2:54:20
P1	Running	2:45:30	2:52:47
P1	Done	2:52:47	4:00:00
P2	Ready	2:54:20	2:56:10
P2	Running	2:56:10	2:57:05
P2	Done	2:57:05	4:00:00

processor (id):

id	from	to
A	1:00:00	4:00:00
B	1:00:00	4:00:00

mailbox (id):

id	from	to
M1	1:00:00	4:00:00
M2	1:00:00	4:00:00
M3	1:00:00	4:00:00
M4	1:00:00	4:00:00
M5	1:00:00	4:00:00
M6	1:00:00	4:00:00
M7	1:00:00	4:00:00

Figure 7.2 : Relationship Relations for the Monitoring System Database

runningon (process, processor):

process	processor	from	to
P1	A	2:00:01	2:15:37
P2	B	2:05:13	2:45:30
P1	B	2:45:31	2:52:47
P2	A	2:56:11	2:57:05

sendmessage (process, mailbox):

process	mailbox	at
P1	M3	2:00:05
P1	M4	2:00:06
P2	M7	2:30:29
P1	M7	2:51:13

waiting (process, mailbox):

process	mailbox	from	to
P2	M7	2:45:30	2:54:20

7.2. Static Representations for Relations

We first construct the representations for each entity relation. The coordinates of the screen are set to lower left corner 0,0 and upper right corner `12,8`.

```
set screen.coordinates to 0,0 to 12,8
```

The process relation is represented with a circle.

```
range of p is process
represent p with circle center (50,50) radius 25
```

The `id` attribute is represented with text centered in the circle. This attribute is also used to position the processes from `0` to `4` along the y-axis.

```
represent p.id with text at 45,50 size 8,8
          yposition range 0 to 4
```

The intensity of a process is determined by its `state` and the xposition of a process is always `0`.

```
represent p.state = Running with intensity 1.0
represent p.state = Ready with intensity 0.7
represent p.state = Blocked with intensity 0.4
represent p.state = Done with intensity 0.1
    setting p.xposition = 0
```

The processor relation is represented with a rectangle.

```
range of s is processor
represent s with icon rectangle position 10,10 size 70,90
```

The id attribute is represented as text at the top left hand corner of the rectangle. This attribute is also used to position the processors from 4 to 8 along the y-axis. The x position of the processor relation is always 20.

```
represent s.id with text at 60,80 size 8,8
            yposition range 4 to 8
        setting p.xposition = 0
```

The mailbox relation is represented with an oval.

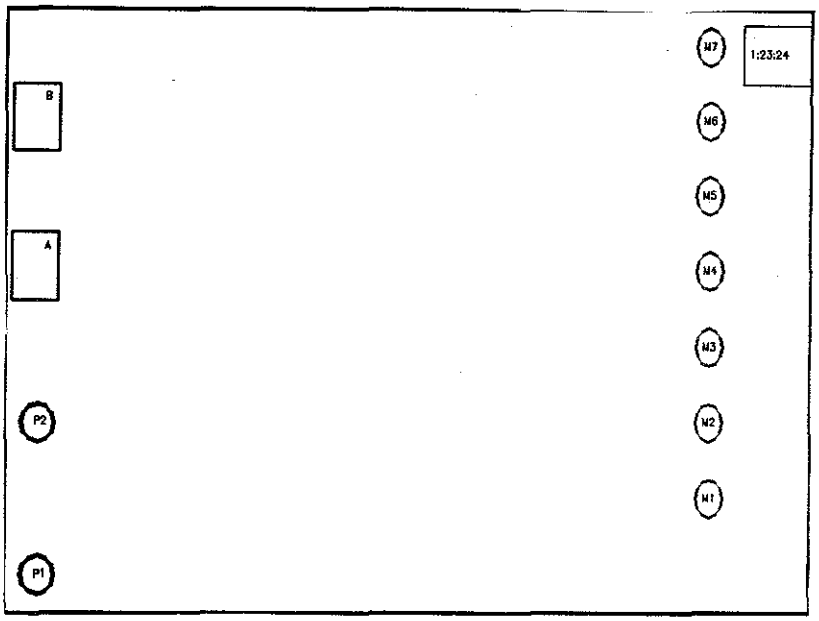
```
range of m is mailbox
represent m with icon oval position 30,25 size 40,50
```

Once again, the id attribute is represented with centered text. The attribute is also used to position the mailboxes from 20 to 80 along the y axis. The x position of the mailbox relation is always 20.

```
represent m.id with text at 14,20 size 8,8
            yposition range 1 to 8
        setting m.xposition = 10
```

The static representations for the entity relations of Figure 7.1 are shown in Figure 7.3.

Figure 7.3 : Static Representation of the Monitoring Database Entity Relations



We next determine the representation of the three relationship relations and of the derived relationship relation. The representations for these relations will differ from previous representations in that they will use the representations of the entity relation named in the attributes which make up their keys.

The representation for the relation `runningon` is the process contained in the processor. The attributes of `runningon` are associated with the representations of the entities they relate.

```
range of r is runningon
range of process is p
range of processor is s
represent r.process with process p
represent r.processor with processor s
    where r.process = p.id and r.processor = s.id
    setting p.yposition to s.yposition
```

It is important to note that these attributes are not associated with representations of the relations that they relate but rather the representation of the particular tuples they relate. Therefore, they will inherit the object characteristics such as xposition, yposition, scale, color, and intensity as well as any iconic representations.

In order to move the process inside the processor, the yposition of the process must be changed. This is accomplished with the *setting* command. This command sets the `yposition` of the `process` object to to the `yposition` of `processor` object (Figure 7.4).

The representation for the relation `sendmessage` is a pointer from the `process` to the `mail-box`. Therefore the attributes of `sendmessage` must be associated with the representations of the tuples they relate.

```
range of s is sendmessage
range of p is process
range of m is mailbox
represent s.process with process p
represent s.mailbox with mailbox m
represent s with pointer (p.xposition,p.yposition)
                         to (m.xposition, m.yposition)
    where s.process = p.id and s.mailbox = m.id
    setting s.width = 12
    setting s.height = 8
```

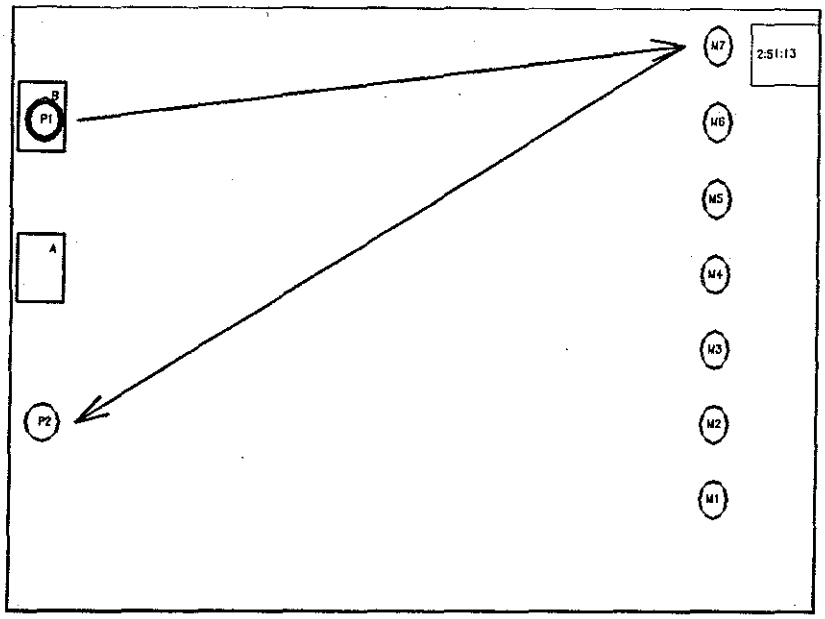
Since the pointer can potentially stretch across the diagonal of the screen, the scale of the object type is set to coordinates of the screen.

The representation for the relation `waiting` is a pointer from the mailbox to the process. The appropriate commands are

```
range of w is waiting
range of p is process
range of m is mailbox
represent w.process with process p
represent w.mailbox with mailbox m
represent w with pointer (m.xposition,m.yposition)
                         to (p.xposition, p.yposition)
    where w.process = p.id and w.processor = s.id
    setting w.width = 12
    setting w.height = 8
```

The resulting representation for all the relations at time 2:51:13 is shown in Figure 7.4.

Figure 7.4 : Static Representation for the Relationship Relations at 2:51:13



7.3. Representing Time

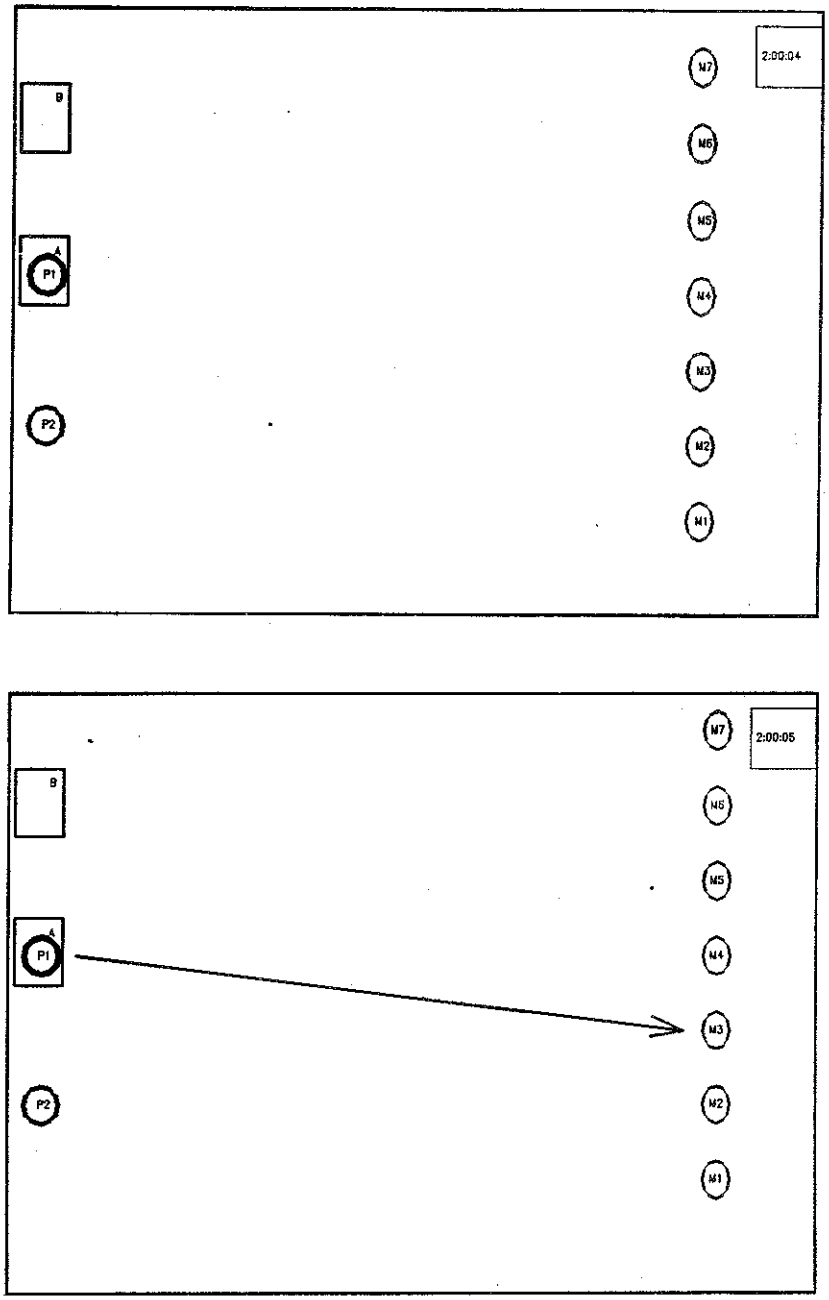
The layout for the static representation of Figure 7.4 contains implicit dependencies among objects. For example, the pointer from `mailbox M7` to `process P2` is dependent both on the position of `M7` and the position of `P2`. The chosen representation of time must preserve these dependencies.

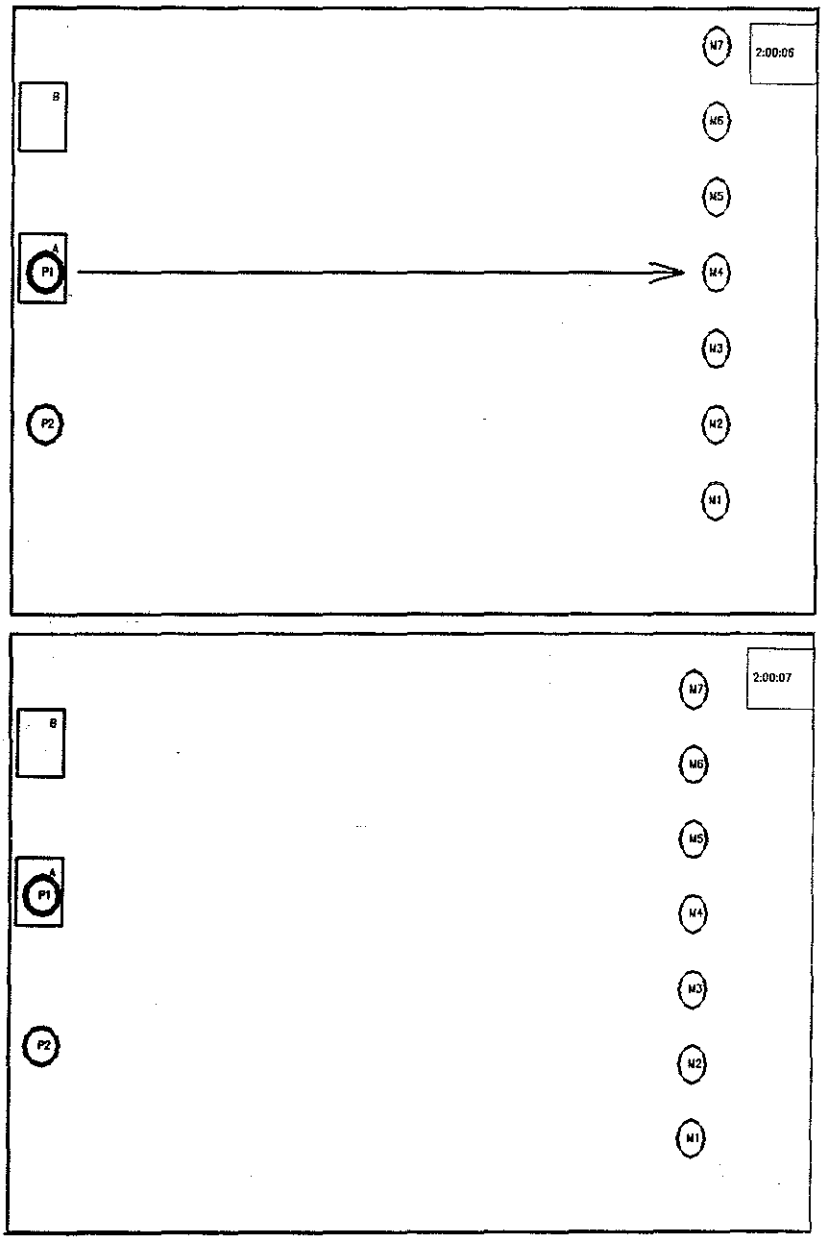
Representing time with animation preserves the dependencies because it does not change the position of objects. However, the position of objects may change over time. Since we are also interested in the actual time, we augment the time representation with a digital clock icon. A digital clock is chosen over a clock face because it shows the number of seconds more explicitly.

```
represent time with animation
                time_icon digitalclock position 11,7 size 1,1
          step = 1 second
```

Figure 7.5 shows a sequence of states using this representation.

Figure 7.5 : Progression of States Using Animation and Digital Clock Icon



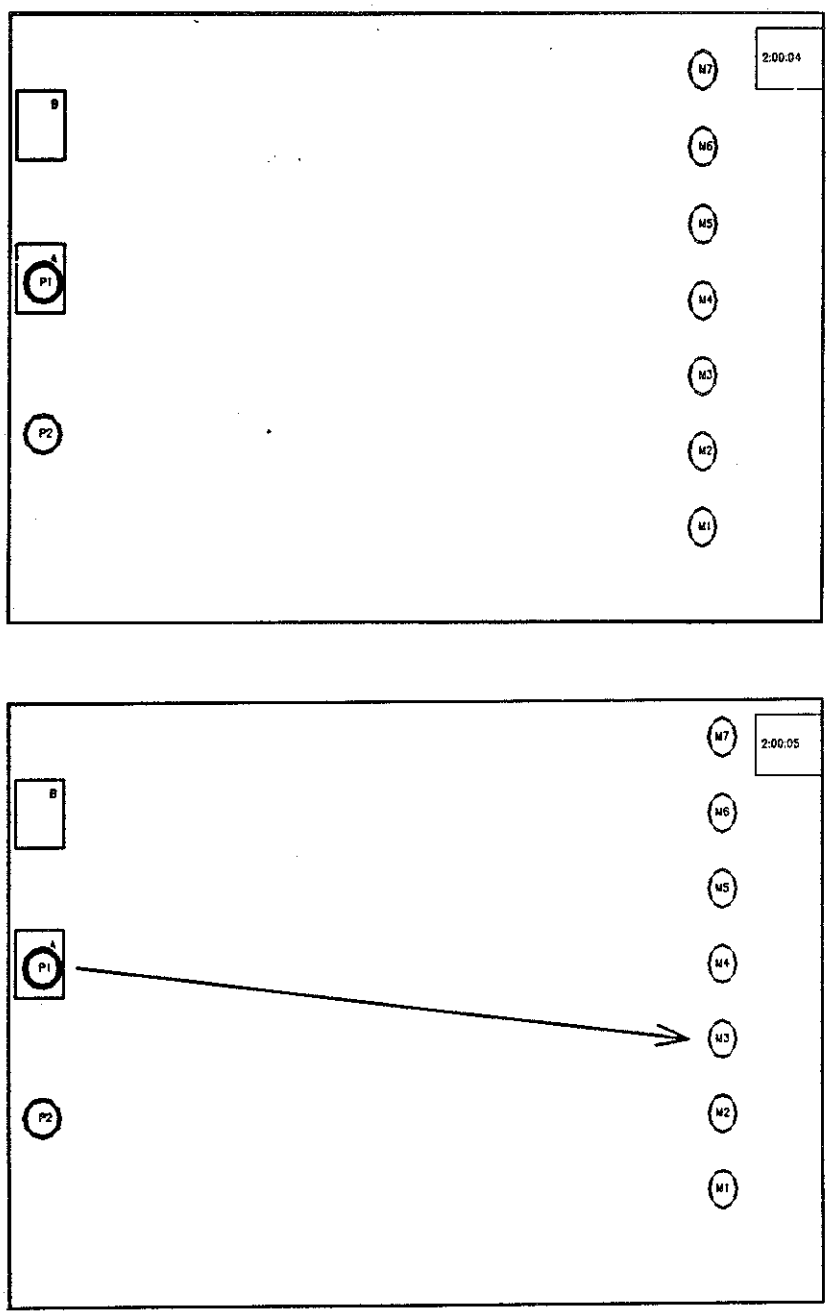


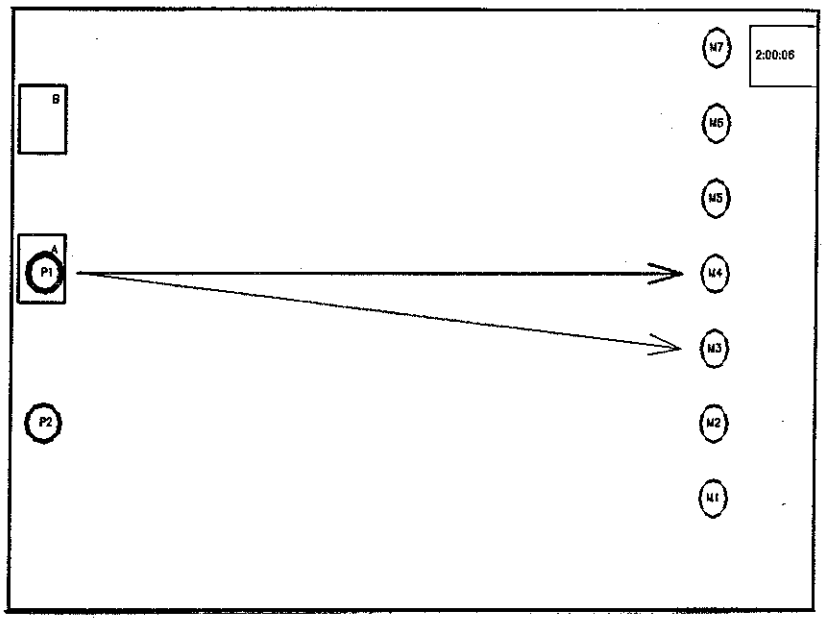
A disadvantage of this representation is that it is hard to remember the previous state. This can be alleviated by replacing animation with animationtrace. With animationtrace, full intensity indicates current events with various decreases in intensity indicating past events.

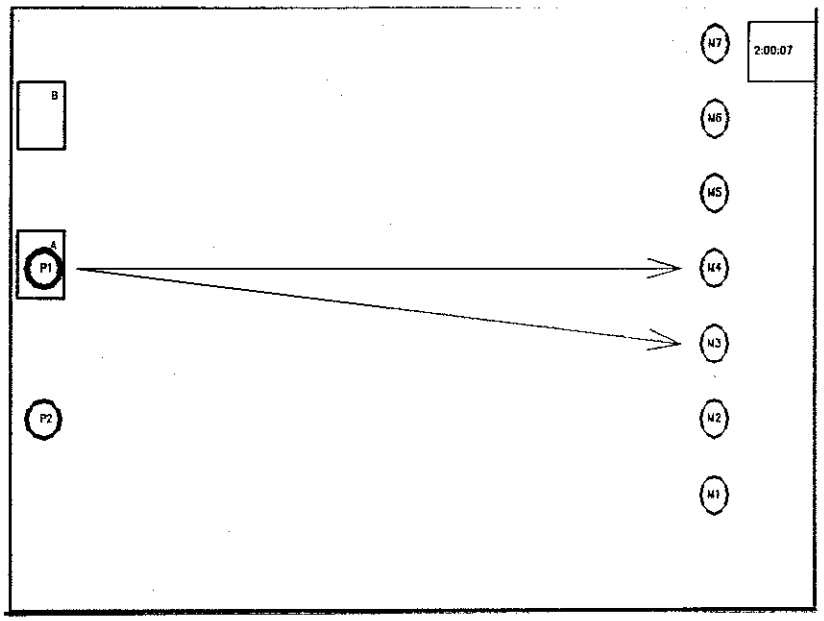
```
represent time with animationtrace
                time_icon digitalclock position 11,7 size 1,1
          step = 1 second
```

Figure 7.6 shows a sequence of states using this representation.

Figure 7.6 : Progression of States Using Animationtrace and Digital Clock Icon




M7
2:00:06
B
M6
M5
A
P1
M4
M3
P2
M2
M1


M7
2:00:07
B
M6
M5
A
P1
M4
M3
P2
M2
M1

Implementation

To show the feasibility of the approach discussed in Chapter 4, we have completed an implementation of a prototype display system. In this part, we describe the main components of the system. Chapter 8 describes the *graphics knowledge base* data structure which is the underlying data structure of the system. Chapter 9 describes the *Schema Editor*, the program used to construct a graphical object description called an object frame using information from the graphics knowledge base. Finally, Chapter 10 describes the displayer. This includes the *Interpreter* which translates the embedded graphical code contained in the object frame to low level graphics subroutine calls and the *Time Displayer*, the module which manages the display of time.

CHAPTER 8

Building the Graphics Knowledge Base

This chapter describes the process of constructing the graphics knowledge base data structure. This process is shown in Figure 8.1.

Figure 8.1 : Building the Graphics Knowledge Base

The first section describes IDL notation which is used for defining the data structures. The second section gives a top down description of the graphics knowledge base data structure using IDL notation. The next section describes the Schema Editor commands to be given by the user. Finally, the last section steps through an example construction of a graphics knowledge base data structure.

8.1. IDL Notation

The data structures are defined using the data specification language *IDL* (Interface Description Language) [Nestor, et al. 1982]. This specification language was chosen for several reasons. First, it provides a high level mechanism for describing complex data structures. Secondly, it supports an object oriented view of the data which is necessary for our approach. And finally, it can be automatically mapped into C language constructs by the IDL translator. This assures consistency between specification and realization.

The following paragraphs give a brief description of the concepts used in the notation. These should be sufficient for understanding the usage in the remainder of the chapter.

The fundamental data structure building blocks of IDL are nodes and classes. These are organized into named collections called *structures*. A *node* is a named collection of zero or more named values called *attributes* that the user wishes to treat as a unit. Attributes actually hold the data values; nodes are a grouping device. Node definitions use the symbol "=>". An example of a node is:

```
action => applyto: objecttemplate,
          code: primitive;
```

The node `action` has two attributes `applyto` and `code`.

The domain of values that an attribute can hold is specified by its *type*. An *attribute type* can be a basic type, a structured type, or a node or class type. For example, the attribute `applyto` has a type which is the node `objecttemplate`. IDL provides four *basic types* and two kinds of *structured types*. The IDL basic types are named by the IDL keywords **`Boolean`**, **`Integer`**, **`Rational`**, and **`String`**.

The structured types that IDL provides are specified by **`Set Of`** <type> and **`Seq Of`** <type>. Here, <type> stands for any valid attribute type, other than sets or sequences. A **`Set Of`** <type> is an unordered collection (set) of objects of <type>. Duplication of objects is not permitted within a set. A **`Seq Of`** <type> is an ordered collection (sequence) of objects of <type>. Duplication of objects *is* allowed in a sequence.

Attributes having a node or class type allow directed graphs to be specified. Nodes can be referenced by several other nodes, permitting arbitrary sharing.

A *class* is a collection of nodes sharing common aspects. The elements of the class are called its *members*. Attributes associated with a class are propagated to all members of the class. Such attributes comprise the common aspects shared by members of the class. Class definitions use the symbols "::=" and "|" to define members of the class and the symbol "=>" to define attributes of the class. An example of a class is:

```
unit ::= second | minute | hour | day | month | year;
unit => val: Integer;
```

The class `unit` has six members; `second`, `minute`, `hour`, `day`, `month`, and `year`. The class attribute `val` is propagated to all six members and is a common aspect shared by all members of the class.

8.2. Description of Graphics Knowledge Base

The *graphics knowledge base* holds all the information necessary to build a graphical description of a data object given the identity and values of its attributes. The data structure consists of a sequence of object template descriptions, a background description, a sequence of time representations, a set of time characteristics, and a representation for indeterminacy. The time characteristics include the interval step, the speed, and the ordering for time, and the mode for displaying frames.

```
graphicsknowledgebase   => objecttemplates: Seq of objecttemplate,
                           background: Seq Of primitive,
                           timerepresentation: Seq Of timerep,
                           timestep: steptype,
                           timespeed: speedtype,
                           timeordering: orderingtype,
                           timemode: modetype,
                           indeterminacyrep: indeterminacyrep;
```

Each object template contains a name, an iconic description, a sequence of modifier attributes, and a set of graphical characteristics for the object template. The graphical characteristics for an object template include the coordinates, the xposition and yposition on the screen, the height and width of the object template, the color, and the intensity level.

```
objecttemplate  => name: String,
                   iconicrep: Seq Of primitive,
                   attributes: Seq Of attribute,
                   coordinates: coordinatepair,
                   xposition: Integer,
                   yposition: Integer,
                   height: Integer,
                   width: Integer,
                   color: color,
                   intensity: Rational;
```

The `iconicrep` sequence contains the iconic representations for the object template, and may include iconic primitives such as lines, circles, points, and polygons and transforming primitives such as color, scale, and translations. These primitives will be discussed shortly.

Attributes of an object template are classified as *finite* or *infinite*. Finite attributes are those that have a finite domain of possible values. Each of the possible values exists in the graphics knowledge base along with modifying actions associated with it.

```
attribute         ::= finite_attribute | infinite_attribute;
attribute         =>  name: String;
finite_attribute  =>  values: Seq Of value_actions;
value_actions     =>  value: String,
```

```
                    actions: Seq of action;
```

Actions of the `value_actions` pair specify what modifications should be made to the representation of the object template when the attribute has the corresponding `value` of the `value_actions` pair. An action is applied to the object the attribute is contained in or to another object. An `action` also consists of a `code` field containing the graphical primitive which is applied to an object.

```
action => applyto: objecttemplate,
          code: primitive;
```

The attribute can be used to modify a characteristic of the object or to add an additional iconic representation to the existing iconic representation of the object. For example, the `code` could contain a `color` primitive which would change the `color` characteristic of the object. Alternatively, the `code` could contain a `polygon` primitive which would be added to the iconic representation of the object.

An infinite attribute has an infinite domain of possible values. Therefore, an explicit modification action cannot exist in the graphics knowledge base. Instead, the modification action is indirectly specified. The possible modification actions associated with infinite attributes include adding text to the object template description, applying a function modifier to the object template description, changing the intrinsic structure of the object template description, and adding another object template description to the existing iconic representation of the object.

```
infinite_attribute     =>  rep: infinite_attributerep;
infinite_attributerep ::= text | functionrep | intrinsicrep |
                          objecttemplate;

functionrep    =>  applyto: objecttemplate,
                   code: primitive,
                   ordering: orderingtype,
                   range: rangetype;
orderingtype  ::= forward | reverse;
intrinsicrep  =>;
```

A `functionrep` uses an ordering of the attribute values to determine the value for a transforming primitive. Again, this modification can be applied to the object the attribute is contained in or to another object. An `intrinsicrep` uses the value of the attribute as a coordinate in one or more components of the object's iconic representation. Different values change the shape of the object representation. Finally, an object template representation adds the iconic representation of another object to the representation of the object.

The graphic primitives that make up the graphical code for each object template are classified in three groups.

```
primitive ::= iconicprimitive | transformativeprimitive | stackprimitive;
primitive =>  dependent: Boolean;
```

All primitives have a field `dependent` which is TRUE if the evaluation of the primitive depends on some attribute value. For example, if the x coordinate of a `point` primitive was equal to an attribute value, the `dependent` field of the `point` primitive would have the value TRUE.

Iconic primitives define graphical shapes.

```
iconicprimitive ::= point | line | pointer | curve |
                    polygon | circle | icon | text;
```

A `point` contains an x and a y field of type `numbertype`. A `numbertype` can be a `Rational` value or the value of an attribute.

```
point => x: numbertype,
         y: numbertype;

numbertype ::= number | objectattributepr;
number     =>  num: Rational;
objectattributepr => objname: String,
                     attname: String;
```

A `line` and a `pointer` both consist of two points. A `pointer` also has a small triangle at its second vertex;

```
line    => vertex1: point,
           vertex2: point;
pointer => vertex1: point,
           vertex2: point;
```

A `curve` is made up of a sequence of points. A spline function maps a smooth curve to these points.

```
curve => points: Seq Of point;
```

A `polygon` also consists of a sequence of points but a connection is assumed between the last point and the first point. A `polygon` also has a `filled` field which indicates if the polygon should be filled in or if only the outline should be drawn.

```
polygon => vertices: Seq Of point,
           filled: Boolean;
```

A `circle` consists of a center point and a radius. It also has a `filled` field indicating if it should be filled in.

```
circle => center: point,
          radius: numbertype,
          filled: Boolean;
```

An `icon` consists of a name, a position, and a size. The `position` attribute gives the position of the lower left corner of the icon in object coordinates. The `size` attribute gives a scaling factor in both the *x* and *y* directions. The graphical code for the icon is contained in another structure. During the object synthesis phase, the icon primitive is replaced by the graphical code and appropriate scaling and translation primitives.

```
icon => name: String,
        position: point,
        size: point;
```

`Text` contains a string value for the text, a position to place the text, and the size of each character in the string in object coordinates.

```
text => value: String,
        position: point,
        size: point;
```

Transformative primitives are further classified into two groups: color primitives and geometric primitives.

```
transformativeprimitive ::= colorprimitive | geometricprimitive;
```

Color primitives affect the color of the object. If the color primitive is a `colorscale`, one value of the scale will be used depending on the attribute value. If the color primitive is an `intensity` value, color will be changed accordingly.

```
colorprimitive ::= color | colorscale | intensity;
color      => name: String;
colorscale => name: String;
intensity  => ival: numbertype,
              range: rangetype;
```

Geometric transformation primitives affect the geometric characteristics of the object template.

```
geometricprimitive ::= rotate | scale | translate | position;
position           ::= xposition | yposition;
```

Rotate, scale, and translate primitives affect the orientation, the size, and the position of the object, respectively.

```
rotate     => angle: numbertype;
scale      => xs: numbertype,
              ys: numbertype;
```

```
translate  => xt: numbertype,
              yt: numbertype;
```

Position primitives define the object position on the screen. They contain a value field or nothing (represented as type `void`).

```
position => value: numbertypeOrvoid;
numbertype ::= numbertype | void;
```

Stack primitives are used in the interpreter to push and pop the color and transformation stacks. This insures that the graphical characteristics and transformations for icons will only be associated with one icon.

```
stackprimitive ::= PUSHCPT | POPCPT | PUSHCOLOR | POPCOLOR;
```

There is one representation of time for all object templates in the graphics knowledge base. This is not as restrictive as it may seem however, because the display system allows the user to change this representation interactively.

The time domain can be represented with geometric primitives, a `colorscale`, `intensity`, `animation`, `animationtrace`, `blinking`, a time icon, or intrinsic position.

```
timerep ::= geometricprimitive | colorscale | intensity |
            motiontype | time_icon | intrinsicxpos | intrinsicypos;
motiontype ::= animation | animationtrace | blinking;
time_icon  ::= clock_icon | month_icon | year_icon;
clock_icon ::= clockface_icon | digitalclock_icon;
```

The step, mode, speed and the direction of time can be set by the user. The mode is controlled by setting the `modetype` to `continuous` or `stop`. The `steptype` is set to the desired interval. This can be the starting or stopping time of each tuple or a specified unit of time. The speed of time is a value from 1 to 10 with 1 being the least time between frames. The direction of time is specified by setting the `orderingtype` to `forward` or `reverse`.

```
steptype ::= starttime | stoptime | unit;
unit     ::= second | minute | hour | day | month | year;
unit     =>  val: Integer;
modetype ::= continuous | fixed;
speedtype => val: Integer;
orderingtype ::= forward | reverse;
```

There is also only one representation for indeterminacy for all object templates. Indeterminacy can be represented by blinking, fading, or dashed lines. Alternatively, it can be ignored.

```
indeterminacyrep ::= blinking | fading | dashedlines | void;
fading =>;
dashedlines =>;
blinking =>;
```

If a blinking representation is used, the step, mode, speed, and ordering time charactertics can be set by the user.

8.3. Schema Editor

The Schema Editor allows the user to interactively specify what iconic representations to associate with an object and what modifying representations to associate with the attributes of the object. In addition, it provides a mechanism to graphically represent the time domain. These associations are accomplished through the use of the `represent` command. The complete BNF for the valid represent commands is given in Appendix D. Portions of these commands in BNF are used in this section.

An object is represented with one or more iconic primitives. An iconic primitive is a line, a pointer, a circle, a point, a polygon, a defined icon, a curve, or text.

```
<iconic primitive> ::= line <point> to <point> |
                       pointer <point> to <point> |
                       {filled}? circle center <point> radius <point> |
                       point <point> |
                       {filled}? polygon <point> { <point> }* |
                       icon <name> position <point> size <point> |
                       curve <point> { <point> }*
                       text at <point> size <point>
```

A point is an x,y coordinate where the values of x and y can each be specified with a number or with the value of an object type attribute. A point can optionally be surrounded by "`()`".

```
<point> ::= ( <point> ) |
            <numtype>, <numtype>
<numtype> ::= <num> |
              <object type> . <attname>
```

If the value is specified with an attribute value, the attribute value type must be numeric rather than a character string.

An object is represented with one or more iconic primitives by the command

```
range of <control> is <name>
represent <control> with <iconic primitive> {<iconic primitive>}*
    setting <set cmd>
    where <where cmd>
```

Each iconic primitive is added to the iconicrep sequence of the object template. The complete description of each primitive must be given in the command.

The default coordinate system for the object template is a square of dimension 100×100. This means that the points used to define the lines, polygons, etc. which make up the object type must be contained in the square with lower left corner 0,0 and upper right corner 100,100. The coordinates of the object template can be changed using the *setting clause* of the represent command.

```
range of <control> is <name>
represent <control> ...
    setting <control>. coordinates to <point> to <point>
```

The values of the points used to define the lines, polygons, etc. should now be contained within the new dimensions of the object type scale.

The setting clause is also used to change other graphical characteristics of the object template. The syntax is

```
<set cmd> ::= set{ting}? <control> . <characteristic set>,
                    {<control> . <characteristic set>}*
<characteristic set> ::= xposition to <int> |
                         yposition to <int> |
                         width to <int>       |
                         height to <int> |
                         color to <name> |
                         background color to <name> |
                         intensity to <rational> |
                         coordinates to <point> to <point>
```

The where clause gives additional conditions for the representation.

```
<where cmd> ::= <where condition> {<where condition>}*
<where condition> ::= <control>.<attribute> = <control>.<attribute>
```

The conditions are a test for the equality of tuple attributes. The representation is only applied if these conditions are met.

Finite attributes of an object template are represented with one or more iconic or transformative primitives. The complete description of the primitive must be given. A finite attribute is represented with a finite primitive by the command

```
represent <control>.<attname> = <attval>
                with <finite_attribute rep> {<finite_attribute rep>}*
<finite_attribute rep> ::= <iconic primitive> |
                           color <name> |
                           background color <name> |
```

```
                                    intensity <rational> |
                                    rotate <int> |
                                    scale <point> |
                                    translate <numbertype> |
                                    xposition <numbertype> |
                                    yposition <numbertype>
```

A represent command must be given for each possible value of the attribute. Each primitive is added to the action sequence of the appropriate `value_actions` node in the finite attribute `rep` sequence. Alternatively, a finite attribute can be treated as an infinite attribute as explained below.

Infinite attributes are represented with geometric primitives, colorscale, intensity, text, or with the representation of another object template. For infinite attributes, the complete description of the primitive need not be given. The syntax is

```
range of <control> is <name>
represent <control>.<attname> with <infinite_attribute rep>
                {<infinite_attribute rep>}*
                {ordering <ordertype>}? {range <rangetype>}?

<infinite_attribute rep> ::= rotate |
                             scale |
                             translate |
                             xposition |
                             yposition |
                             colorscale <name> |
                             text at <point> size <point> |
                             objecttemplate
<ordering type> ::= forward | reverse
<range type> ::= <numbertype> to <numbertype>
```

The ordering type can be forward or reverse. The default is forward. The range gives a range of values to use. The values of the attribute for all valid tuples will be sorted and then assigned a value for the infinite primitive. This allows for maximum differentiation between values.

Time is treated similarly to infinite attributes in terms of representation in that a complete description of the primitive need not be given. Time can be represented with combinations of infinite attribute representations, blinking, animation, animationtrace, a time icon, intrinsic position, or nothing. Representing time with nothing implies that the data will be treated as static data rather than temporal data.

```
represent time with <timerep> {<timerep>}*
                {ordering <orderingtype>}? {step = <steptype>}?
                {mode = <modetype>}? {speed = <speedtype>}?
<timerep> ::=   <infinite_attribute rep> |
                blinking |
                animation |
                animationtrace |
```

```
                    <time icon> at <point> |
                    intrinsicxpos |
                    intrinsicypos |
                    void
<time icon> ::= digitalclock |
                    clockface |
                    month |
                    year
```

For time representations other than void, the ordering, step, mode, and speed can be specified. The ordering is forward or reverse. The step is a time increment. This can be the starting or stopping time for the object or a number of seconds, minutes, hours, days, months, or years.

```
<steptype> ::=  starttime
                stoptime
                <int> second(s)
                <int> minute(s)
                <int> hour(s)
                <int> day(s)
                <int> month(s)
                <int> year(s)
```

The mode is continuous or stop. The speed is a value from 1 to 10 with 1 being the least time between frames.

```
<modetype> ::= continuous |
               stop
<speedtype> ::= [1-10]
```

Indeterminacy is represented with blinking, fading or dashed lines.

```
represent indeterminacy with <indet_rep>
                {ordering <orderingtype>}? {step = <steptype>}?
                {mode = <modetype>}? {speed = <speedtype>}?
<indet_rep> ::= blinking | fading | dashedlines;
```

If a representation is not specified, it is ignored. The ordering, step, mode, and speed can be specified if they are not already specified in a time representation command.

8.4. Example Transformation

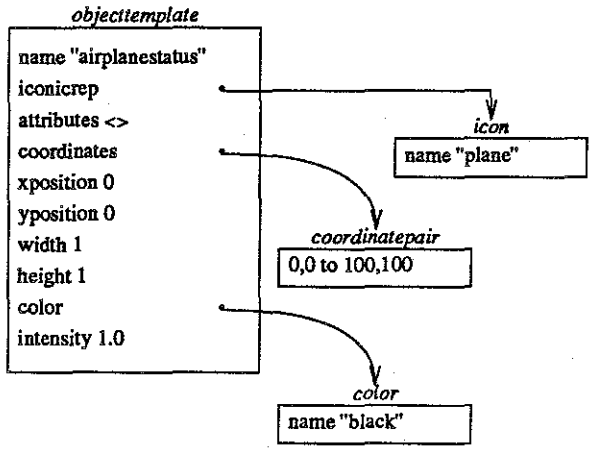
This section will step through an example construction of a graphics knowledge base using the Schema Editor. The commands used are those used to represent the tuples of the `airplanestatus` relation in of Figure 5.1.

The first command represents each object with a plane icon.

```
range of a is airplanestatus
represent a with icon plane position 10,10 size 80,80
```

This creates an object template named `airplanestatus` with an iconic representation of a plane icon. The characteristics of the template are set to the defaults. This object template is added to the graphics knowledgebase. The object template for `airplanestatus` is shown graphically in Figure 8.2.

Figure 8.2 : Object Template for `airplanestatus`



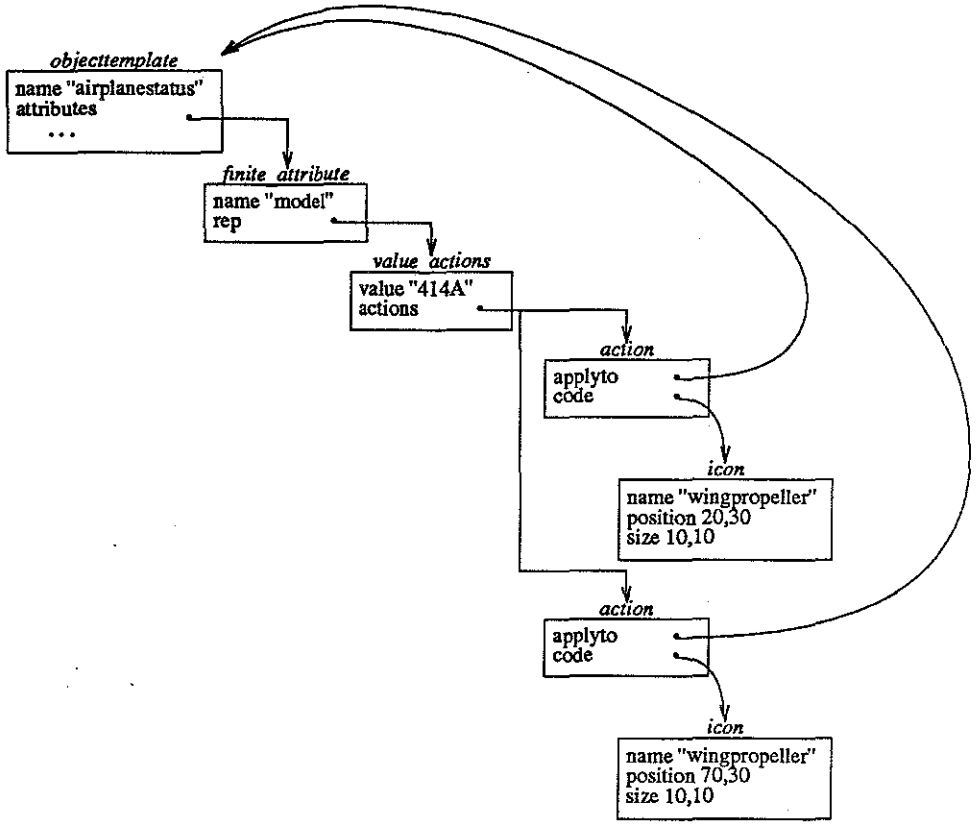
The next command represents the finite attribute-value pair `model = 414A`.

```
represent a.model = 414A with icon wingpropeller position 20,30
                                                    size 10,10
                               icon wingpropeller position 70,30
                                                    size 10,10
```

A finite attribute is created and added to the attributes sequence of the object template. The `rep` of the attribute contains two `actions` for the value "`414A`" of "`model`". These actions are applied to the representation of the object template.

Each `action` corresponds to a representation clause for the attribute-value pair. The `code` of the `action` gives the graphical primitive for the representation. The actions for the attribute pair "`model = 414A`" are shown in Figure 8.3.

Figure 8.3 : Actions for Finite Attribute Pair `model = 414A`

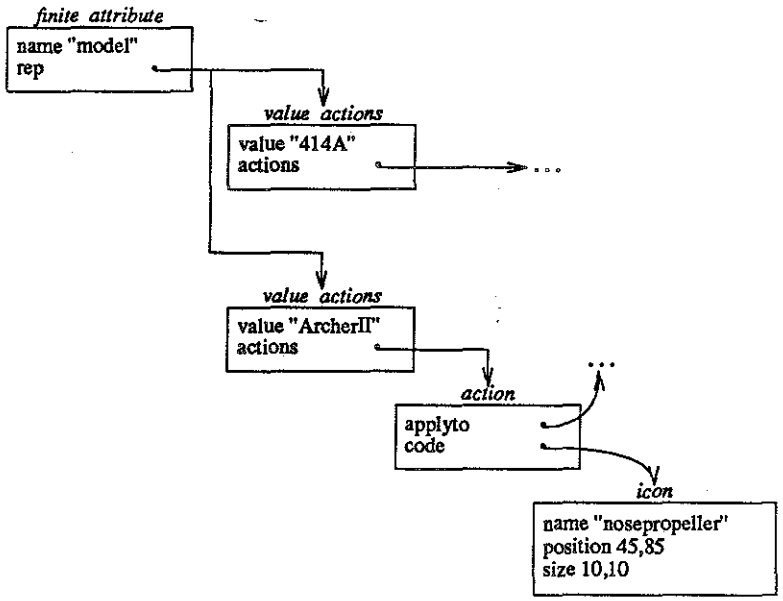


The next command represents the finite attribute-value pair "`model = ArcherII`".

```
represent a.model = ArcherII with icon nosepropeller
                                        position 45,85 size 10,10
```

Since the finite attribute "model" has already been created, a new `value_actions` node is added to the `rep` of the existing attribute. There is one action for this pair corresponding to the one representation clause. The action for the finite attribute pair "`model = ArcherII`" is shown in Figure 8.4.

Figure 8.4 : Action for Finite Attribute Pair `model = ArcherII`

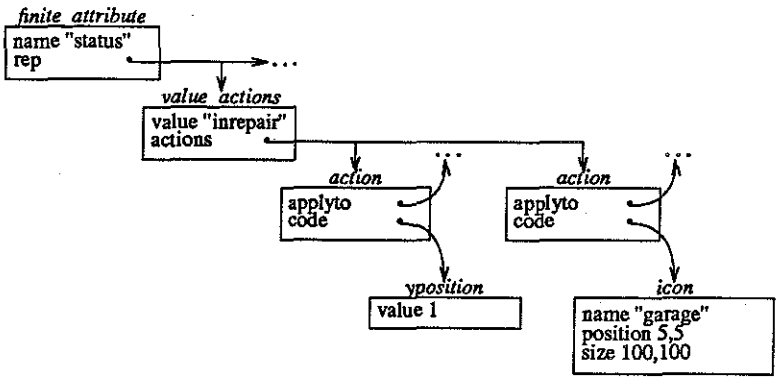


The next commands represent the finite attribute "`status`".

```
represent a.status = inrepair with yposition 1
                                   icon garage
                                      position 5,5 size 90,90
represent a.status = onground       with yposition 1
represent a.status = trainingflight with yposition 2
represent a.status = demoflight     with yposition 3
represent a.status = freeflight     with yposition 4
```

First a finite attribute for "`status`" is created and added to the attributes sequence of the object template. Next a `value_actions` node is created for each attribute-value pair and added to the `rep` of the finite attribute "`status`". Again the actions are added for each representation clause. The actions are shown for attribute-value pair "`status = inrepair`" in Figure 8.5.

Figure 8.5 : Actions for Finite Attribute Pair `status = inrepair`

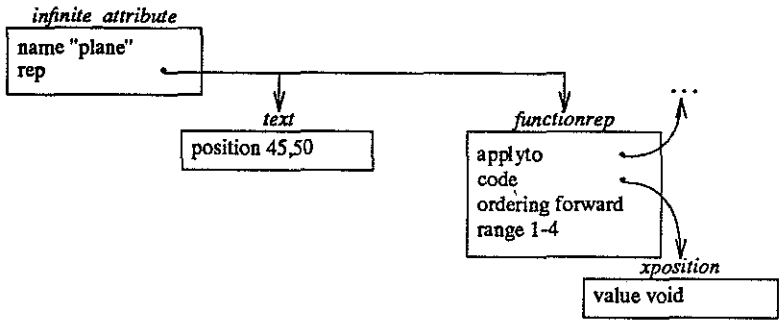


The next command represents the infinite attribute "`plane`".

```
represent a.plane with text at 45,50
                        xposition range 1 to 4
```

First an infinite attribute for "`plane`" is created and added to the attributes sequence of the object template (Figure 8.6). Next, a representation is added to the `rep` sequence of the attribute for each representation clause. The resulting infinite attribute is shown in Figure 8.6.

Figure 8.6 : Representation for Infinite Attribute `plane`

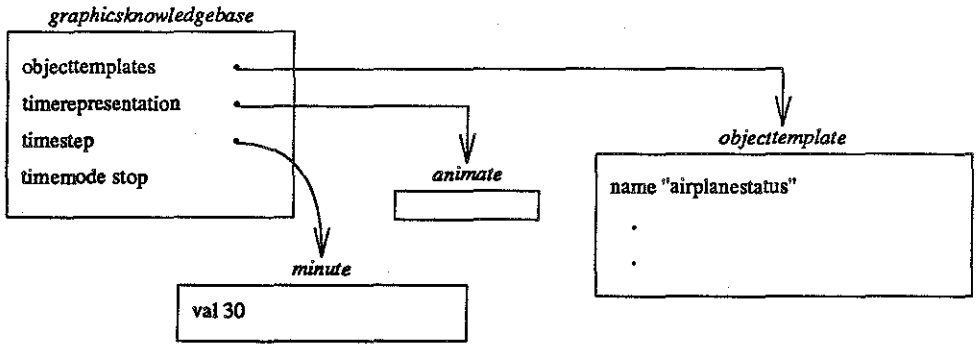


The last command represents `time`:

```
represent time with animation step = 30 minutes mode = stop
```

A new `timerep` node "`animate`" is created and added to the `timerepresentation` sequence of the graphics knowledgebase. In addition, the `step` and `mode` are set. The speed and ordering are set to defaults. The time representation is shown in Figure 8.7.

Figure 8.7 : Representation for Time



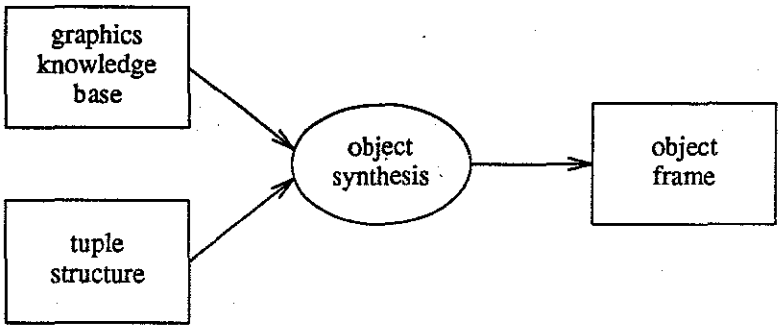
The graphics knowledge base is now constructed. The next step is to construct object descriptions for each tuple returned from the TDBMS using the object templates in the graphics knowledge base. Objects are constructed during the object synthesis phase described in the following chapter.

CHAPTER 9

Building the Object Frame

The object synthesis process uses information in the graphics knowledge base to construct an object description for each active tuple. This process is illustrated in Figure 9.1.

Figure 9.1 : Building the Object Frame



The object synthesis process builds a data structure called the *object frame* which contains embedded graphical primitives. After the synthesis is complete, the object frame is sent to the Interpreter which calls the appropriate low level graphics routines for each graphical primitive in the object frame.

The first section describes the tuple data structure returned from the TDBMS. The next section discusses the object frame data structure. Finally, the last section steps through an example object synthesis process using the constructed graphics knowledge base of Section 8.4.

9.1. The Tuple Structure

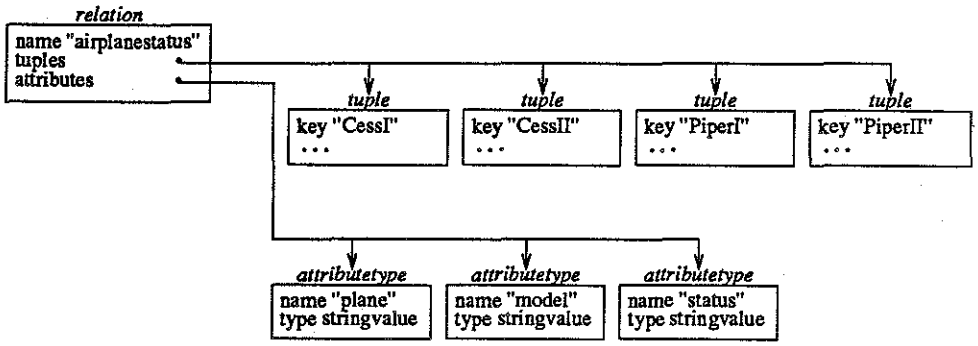
The tuple structure contains the list of active tuples returned from the user query. The list is organized so that all tuples from the same relation are listed inside the relation structure.

```
relation => name: String,
            tuples: Seq Of tuple,
```

```
attributes: Seq Of attributetype;
```

The relation structure also contains the name of the relation and the sequence of attributes contained in the relation. The example relation of Figure 4.1 is illustrated in Figure 9.2.

Figure 9.2 : An Example Relation



Each attribute has a value type of numeric or string.

```
attributetype => name: String,
                 type: valuetype;
valuetype     ::= numericvalue | stringvalue;
numericvalue =>;
stringvalue  =>;
```

Each tuple returned from the user query contains a unique key, a sequence of attribute name-value pairs, and a time domain.

```
tuple => key: String,
         attributes: Seq Of attributepair,
         time: timetype;
attributepair => name: String,
                 value: String;
```

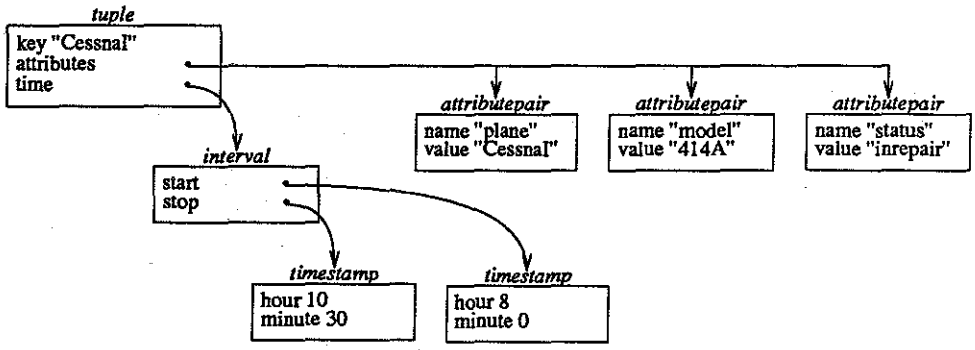
Each key is a single string-valued attribute. The `time`-domain can be of type `event`, `interval`, or `static`. A `static` time-domain implies that the data is static rather than temporal. An `event` time-domain contains one time stamp and an `interval` time-domain contains two timestamps.

```
timetype  ::= event | interval | static;
event      =>  start: timestamp;
interval   =>  start: timestamp,
               stop: timestamp;
timestamp =>   year: Integer,
               month: Integer,
               day: Integer,
               hour: Integer,
               minute: Integer,
               second: Integer;
```

```
static     =>;
```

The tuple for `CessI` is illustrated in Figure 9.3.

Figure 9.3 : Tuple for `CessI`



9.2. The Object Frame

The object synthesis process builds an object for each tuple in the tuple structure. This object contains the information necessary to graphically display the corresponding tuple.

The object frame contains the graphics knowledgebase, the defined icons, the active tuples, and the constructed objects.

```
objectframe => knowledgebase: graphicsknowledgebase,
               iconlist: icons,
               activetuples: relations,
               objects: Seq Of object;
```

Each object contains a reference to the object template and tuple from which it was constructed plus other graphical attributes.

```
object => template: objecttemplate,
          tuple: tuple,
          rep: Seq Of primitive,
          coordinates: coordinatepair,
          xposition: Integer,
          yposition: Integer,
          width: Integer,
          height: Integer,
          color: color,
          intensity: Rational;
```

The `rep` sequence is constructed from the appropriate primitives of the object template given the tuple's attribute values. The graphical characteristic values are copied directly from the graphical

characteristics of the object template and then modified by the representation for the object template attributes.

9.3. Object Synthesis

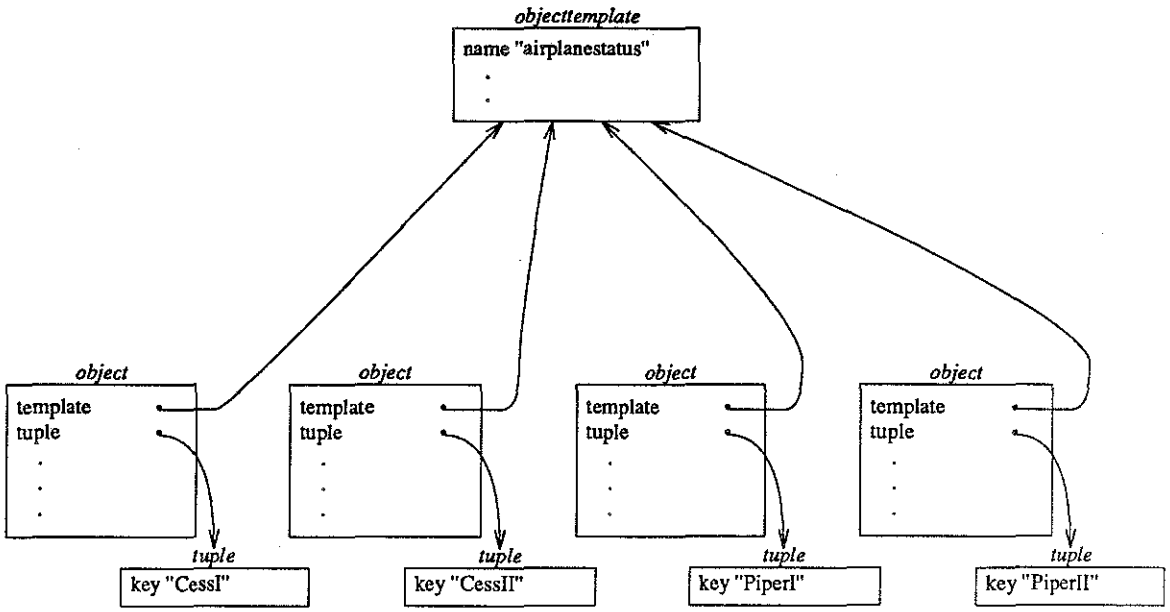
This section describes the object synthesis phase using the constructed graphics knowledge base of Section 8.3 and the following tuples.

`airplanestatus ( plane, model, status ):`

plane	model	status	from	to
CessI	414A	inrepair	0800	1030
CessII	414A	demoflight	0800	1000
PiperI	ArcherII	onground	0800	0900
PiperII	ArcherII	trainingflight	0800	0900

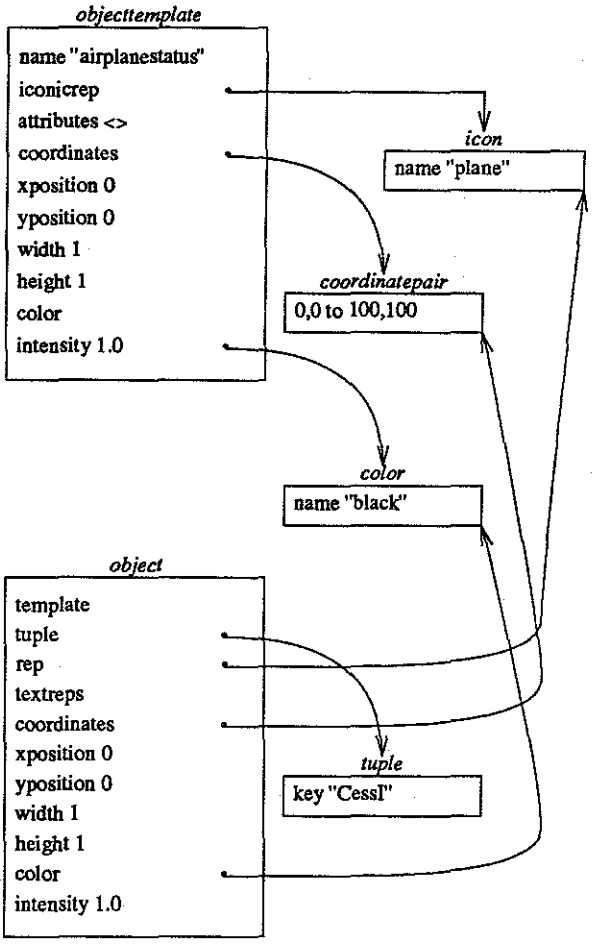
The tuple structure contains one relation node `airplanestatus`. The first step is to create an object description for each tuple in the relation. The object templates in the graphics knowledgebase are searched for a name corresponding to the relation name. If one is found, an object will be created from this template for each tuple. If one is not found a default object template is used. Each object will have a pointer back to the template and a pointer to the tuple. This relationship is illustrated in Figure 9.4.

Figure 9.4 : Creating an Object for each Tuple



Several attributes of the object template are shared by the objects created from the template. Other attributes are copied in to each object. Integer, rational, and boolean attributes are copied. String and node attributes are shared unless an object characteristic is changed by a conditional modifier. For example, in Figure 9.5, the `iconicrep`, `coordinates`, and `color` attributes are shared while the `xposition`, `yposition`, `width`, `height` and `intensity` attributes are copied. The sharing of attributes increases the efficiency of the object synthesis module since it decreases the number of dynamic memory allocation calls needed.

Figure 9.5 : Sharing Object Template Attributes



The next step is to add the attribute representations to each tuple. The attribute representations modify the object representation. For each attribute in the tuple, an attribute with the same name is searched for in the graphics knowledge base. If no attribute is found a default representation is used.

If the attribute found is a finite attribute, the actions associated with the value of the attribute corresponding to the attribute value in the tuple are applied to the object representation. If the graphical code of the action is an iconic primitive, the primitive is added to the `rep` sequence of the object. If the code modifies the object, the appropriate graphical characteristic of the object is changed.

For example, the finite attribute `model` has the actions shown in Figure 8.3 and Figure 8.4. For the object created for `CessI`, the actions applied will be the actions corresponding to the attribute-value pair "`model = 414A`" shown in Figure 8.3. Therefore, two wingpropeller icons are added to the `rep` field of the object (Figure 9.6).

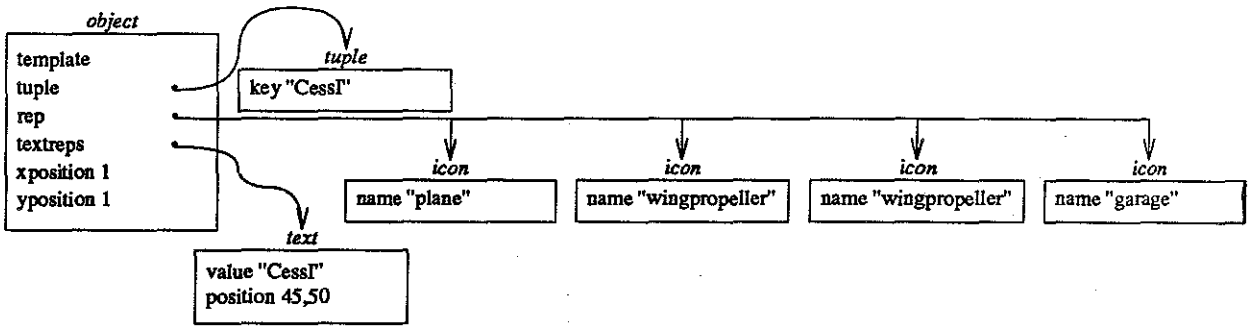
If the attribute found is an infinite attribute, the appropriate value of the graphical primitive given by a `functionrep` is determined using the attribute value. This primitive then modifies the object. The primitive is assigned the correct value depending on the highest and lowest value of that attribute for all tuples. This allows the maximum differentiation between different values. Any text representations are added to the `textreps` sequence of the object. These representations are separated from the other iconic representations so that the text can be displayed last.

For example, the infinite attribute `plane` has a `text` representation and a `functionrep`. These representations are illustrated in Figure 8.6. The `text` is added to the `textreps` sequence of the object. The code of the `functionrep` contains the primitive `xposition`. The value for this primitive is determined by the ordering of the tuples using the `plane` attribute and by the range. The ordering of the tuples using the `plane` attribute is

```
CessI   CessII   PiperI   PiperII
```

The range is 1 to 4. Therefore, the tuple with attribute-value pair "`plane = CessI`" is given the xposition 1. This representation changes the `xposition` graphical characteristic of the object. The resulting object for `CessI` after applying the modifications of the `model`, `status`, and `plane` attributes is shown in Figure 9.6.

Figure 9.6 : The Object for `CessI`



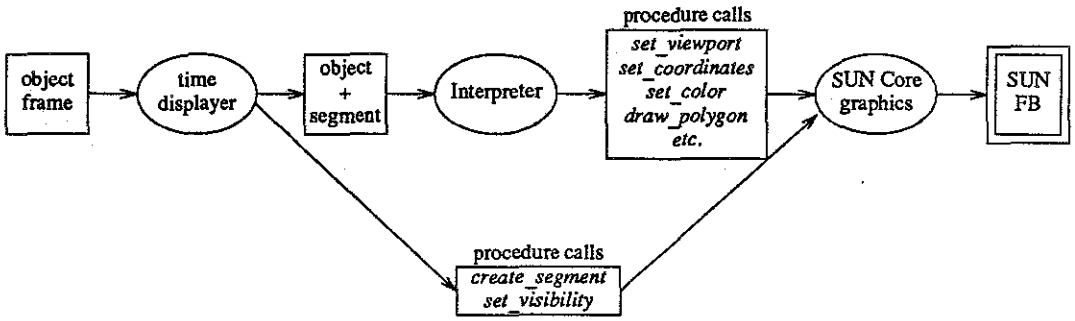
After attribute representations have been added for each tuple, the object frame will contain high level graphical descriptions for each tuple. The object frame is then sent to the time displayer module.

CHAPTER 10

Displaying the Object Frame

This chapter explains the modules which display the object frame data structure. This process is shown graphically in Figure 10.1.

Figure 10.1: Displaying the Object Frame



The first section describes the module which manages the display of time. The next section describes the Interpreter which translates the embedded graphical description language, or primitives, into procedural calls to SUN Core graphics. These procedural calls generate the images on the SUN frame buffer (FB).

10.1. Time Display Controller

The order in which the objects are displayed depends on the representation for time. If time is represented with the same techniques as other infinite attributes, then all objects are displayed at the same time. If time is represented with animation or animationtrace, then the objects are displayed in the order in which they are valid if the ordering is forward and the opposite order if the ordering is reverse. If time is represented with blinking, then all objects are displayed at once and each object representation will blink when that object becomes valid. For animation, animationtrace, and blinking representations, a time icon such as a clock face or calendar page can appear as a reference point.

Real time animation is simulated through the use of graphical segments. A *graphical segment* is the part of the entire graphical display list which represents a portion of the complete picture on the screen. Each segment contains calls to the SUN core graphics package to draw lines, polygons, points, etc. on the display. By turning the visibility of the segment off, a portion of the total picture can be deleted. Turning the visibility back on will redisplay that portion.

One or more segments are created for each object. The number of segments created for an object is determined by the duration of the object's time interval, the time step, and the representation of time. If time is represented with intrinsic position, then one segment is created for each object and all objects are displayed at the same time by setting the visibility of all segments on simultaneously. If time is represented with position, scaling, rotation, intensity, colorscale, or any combination of these, then the number of segments created for each object is equal to the duration of the object divided by the timestep. For example, if the duration of the interval for an object is 2 minutes and the time step is 30 seconds, then four segments will be created for the object; one for each time step in the interval. Each segment will contain a description of the object with the appropriate transformation. If the time representation include animationtrace, then the number of segments will be five. The last four segments will contain a representation of the object with decreasing intensity. Finally, it time is represented with only animation or blinking then one segment is created for each object.

If the time representation includes blinking, then all segments are displayed simultaneously and each segment is blinked once when it becomes valid. If the time representation includes animation or animationtrace, then each segment is set to visible when the segment becomes valid and set to nonvisible when is is no longer valid.

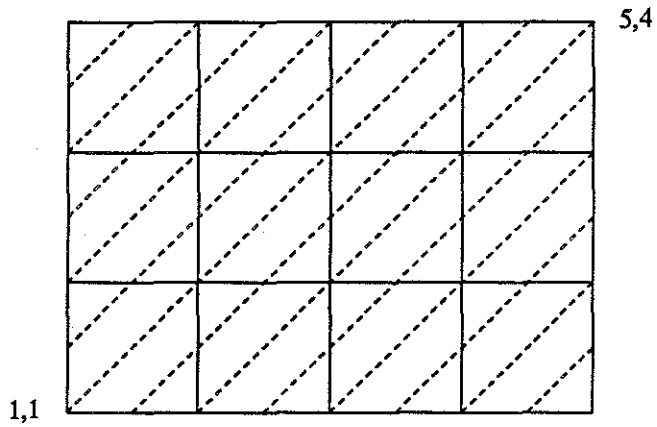
The SUN core graphics procedure calls contained within each segment are determined by interpreting the embedded graphical description language for each object. The Interpreter is described in the following section.

10.2. Interpreter

The graphical representations of each object are drawn on the frame buffer (FB) of a SUN Workstation. A *frame buffer* is a rectangle of pixels. Graphical shapes are drawn by assigning a color to the

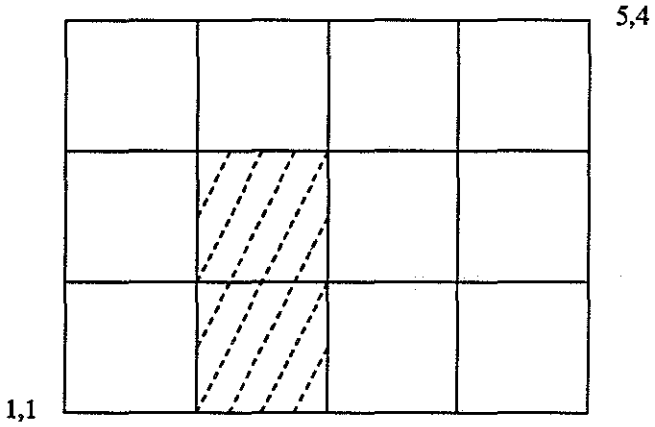
appropriate pixels. Initially, all pixels are assigned the color "white". The *viewport* is the portion of the frame buffer which can be written into. The viewport for the entire screen is a rectangle with lower corner 0,0 and upper corner 1.0,.75 which covers the entire frame buffer. Within the viewport is the grid of squares for the screen. For example, if the screen coordinates are lower corner 1,1 and upper corner 5,4, the screen will contain a grid of 12 squares. The shaded portion shown in Figure 10.2 is the viewport for the entire screen.

Figure 10.2: Viewport for Entire Screen



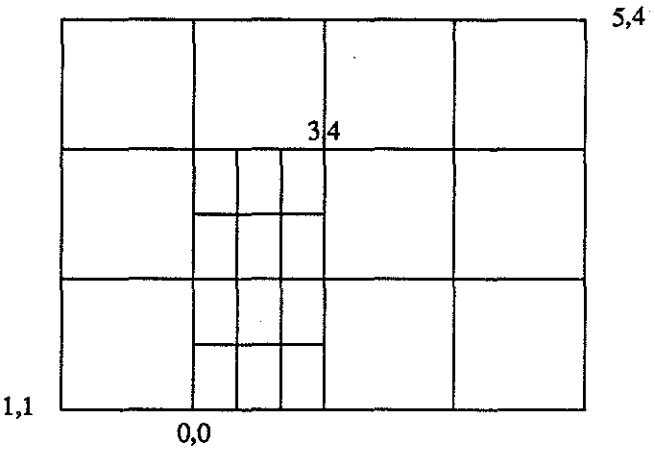
The viewport for each object is determined by the `xposition`, `yposition`, `height`, and `width` characteristics of the object and from the coordinates of the screen. For an object with `xposition = 2`, `yposition = 1`, `width = 1`, and `height = 2`, the viewport is shown as the shaded portion of Figure 10.3. The viewport now has lower corner .25,0 and upper corner .5,.5.

Figure 10.3: Viewport for Object



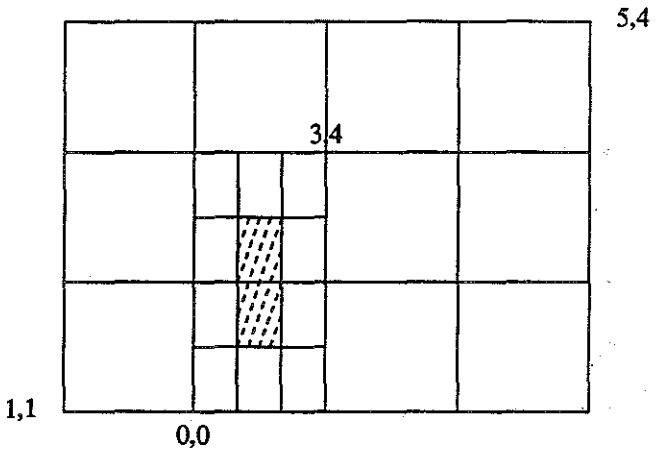
After the viewport is calculated, a *set_viewport* call to SUN core graphics sets the specified region on the screen. The coordinates for the object are set within this region. For example, if the object coordinates are lower corner 0,0 and upper corner 3,4, the region would have the coordinates shown in Figure 10.4.

Figure 10.4: Object Coordinates Within Viewport



All iconic primitives contained in the object will be drawn in the object coordinate system with the set viewport. For example, the polygon defined by the sequence of points `1,1 2,1 2,3 1,3` would be drawn as shown in Figure 10.5.

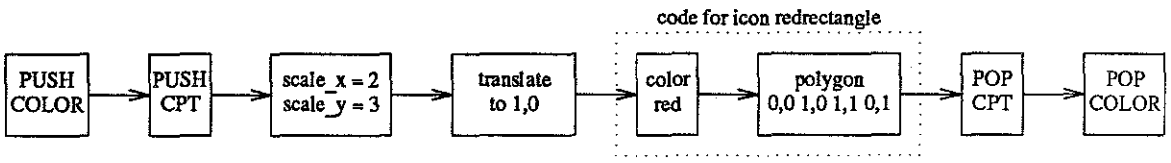
Figure 10.5: Polygon Drawn Within Object Coordinates



The interpreter maintains a *current position transformation* 3×3 matrix (CPT), a current color, and a stack for each. Initially, the CPT is the identity matrix. As each geometric transformation is encountered, it is combined with the CPT. Two operators exist for saving and restoring the CPT: PUSHCPT and POPCPT. These operators surround the graphical code for each icon so that the transformations applied to the icon will not affect any other icon. The current color is initially the object color. As each intensity command or color command is encountered, the current color is changed accordingly. Two operators also exist for saving and restoring the current color: PUSHCOLOR and POPCOLOR. These operators also surround the code for each icon.

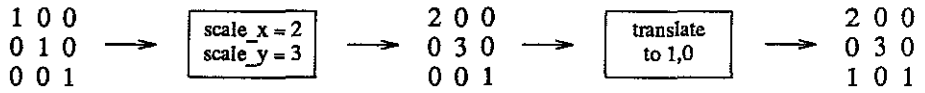
For example, the sequence of primitives generated for the icon `"redrectangle"` with position `1, 0` and size `2, 3` is shown graphically in Figure 10.6.

Figure 10.6: Sequence of Primitives for Icon



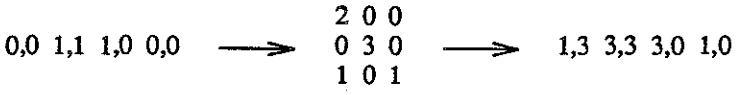
The current CPT is first saved on the stack. Next, a new CPT is constructed by applying the scaling and translating primitives to the current CPT. For example, if the initial CPT was the identity matrix, then the scale primitive and translate primitive would be applied to the identity matrix to generate a new CPT. This is shown graphically in Figure 10.7.

Figure 10.7: Generating a New CPT



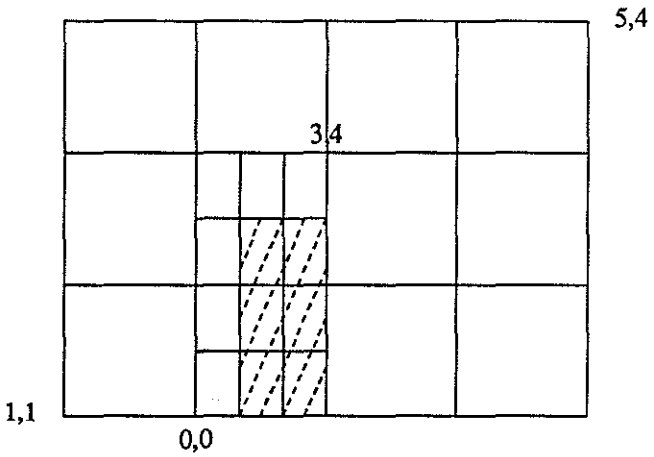
As each point in the polygon is encountered, it is transformed by the CPT as shown in Figure 10.8.

Figure 10.8: Transformation of Polygon Points



The current color is then set to "`red`" and the polygon is drawn in the object coordinate system within the set viewport. Both the setting of the color and the drawing of the polygon are accomplished by calling the appropriate routines in the SUN core graphics package. The resulting representation is shown in Figure 10.9. Color is shown with texture.

Figure 10.9: The Icon Representation



After the icon is drawn, the previous CPT and color are restored by popping each off their respective stacks. The remainder of the iconic representations for the object are then transformed by the current CPT (the identity matrix) and assigned the current color until another icon is encountered.

After the interpretation of the object is completed, the segments associated with each object will contain a complete low level description of the object. The properties of the segments such as visibility are

then changed by the time displayer depending on the specified representation for time.

Evaluation and Conclusion

This part contains the evaluation and conclusion chapters of the thesis. Chapter 11 gives an evaluation of our display system including a discussion of how display issues were resolved and how the essential features of the approach were met. In addition, efficiency considerations are discussed and possible extensions to the system are proposed. Chapter 11 concludes the research.

CHAPTER 11

Evaluation

This chapter gives an evaluation of our display system. We first give the status of the implementation. We next discuss how the issues described in Chapter 2 were resolved. We next describe how the essential features of our approach were met. We next discuss the efficiency of the system and suggest possible improvements. Finally, we discuss the possible extensions to our system.

11.1. Status of Implementation

The prototype display system was implemented as described in Chapters 8, 9, and 10. Currently the Schema Editor, the Object Synthesis module, the Time Displayer, and the Interpreter are all implemented. The figures in Chapters 5, 6, and 7 were generated using this display system. In addition, a pretty printer for graphics knowledge bases was implemented. Appendix E was generated by the pretty printer.

The entire system was written in C on a SUN-2 running UNIX 4.2BSD. The system uses IDL, lex, yacc, and SUN Core. The system consists of 15,663 lines of C source code, 3811 lines of C declarations, 874 lines of lex and yacc, and 349 lines of IDL. Lex and yacc generated 3348 lines of the source code and 111 lines of C declarations. The IDL translator generated 6593 lines of the source code and 3627 lines of C declarations.

11.2. Static Display Issues

The issues in displaying information in static databases include effective representation of data, layout of data, and efficiency of the system.

For effective representation, there are five goals.

(1) The image must convey a specific message. Conveying a particular message or meaning is difficult because often the ideas of the designer of the image are very different than the ideas of the user. In addition, some images convey undesirable messages or have different meanings in different contexts.

(2) Relevant attributes of the data need to be displayed in a way that differentiates between distinct data values. The choice of representation affects the amount of differentiation.

(3) Alternate representations of the same data should be provided. Different tasks may need to emphasize different attributes of the same data.

(4) Different levels of detail are useful. Certain applications need more or less detail.

(5) Interactive creation of representations for new data, namely data derived from querying the database, should be provided. It may not be reasonable to provide defined representations for every possible derivation.

Our system met goals 1-3 and goal 5. These goals are met by allowing the user to interactively specify the representation of the data. TQuel, a temporal query language, allows the user to choose the appropriate attributes of interest. The Schema Editor allows the user to represent the object and the attributes graphically. The representation commands are processed interactively allowing the user to experiment with different object and different attribute representations. Interactive specification of representations allows different tasks to emphasize different attributes of the same data. Another advantage of interactive specification is that a designer is not required to determine one representation for all applications. Supporting only one representation might compromise the effectiveness of each application display. Finally, the Schema Editor commands allow the representation of newly defined queries on the database. This again allows interactive experimentation to determine the effectiveness of various displays. The fourth goal is listed as an extention in Section 11.4.

Determining an effective layout of data is a research topic in itself. There are three main issues.

(1) The graphical system must handle limited screen space. Given a large set of objects which do not all fit on the screen, the system must determine which portions to display.

(2) The graphical system must preserve any implicit relations among data or representations of data when determining a layout. The chosen layout must not obscure or change these relationships.

(3) The graphical system must ensure that the size of each representation is large enough to convey a specific idea. The system must be aware of the size and resolution of the display in order to effectively display each object.

Since layout of data was beyond the scope of our research, we provided one limited solution. The approach is similar to the representation approach in that it allows the user to choose between different screen layouts. User specification of screen layout is accomplished by representing various attributes with `xposition` or `yposition` on the screen. This positioning representation of attributes determines the layout of the objects on the screen. The implicit relations between the objects are preserved because the layout is based on the differences of attribute values. The coordinates for the screen are also specified by the user thus allowing different sizes for the objects. Scaling of objects can also be controlled by the user.

An interesting improvement to our approach would be to provide automatic resolution for objects which appear at the same position on the screen thus obscuring each other. This improvement is discussed as an extension in Section 11.4.

The final issue in displaying static information is attaining adequate efficiency of the display system. The graphics system must be efficient enough to use interactively. In many situations, the complexity of generated images must be restricted to attain this efficiency. Adequate efficiency is also relevent for temporal display especially when animation is used to represent time. Efficiency in our prototype system is discussed in Section 11.4.

11.3. Temporal Display Issues

Additional issues that arise when displaying temporal information include a consideration of the human perception of time, the properties of the time domain, and the problems encountered when representing time domains.

The human perception of time relates to how one experiences time and how one describes time. Time is experienced as flowing or as a separation of events. Time is described in terms of duration and/or succession. Representing the time domain involves choosing a representation for events, intervals, and indeterminacy. The representation chosen can emphasize duration, succession, or both. In addition, a representation can make time be perceived as flowing or as a separation of events by making the changes more or less distinct.

A major feature of our approach is that we allow more than one representation for time. In addition, time representations can be combined for different effects. Again, the representation can be chosen by the user to allow experimentation. The remainder of this section discusses how the choice of particular representations resolves the issues listed in Section 2.2.3.

(1) *Representing one event*

An event cannot be represented as static data because this contradicts the notion of the event existing in time. In addition, time is not an independent characteristic of an event but rather a way to describe relations between events. Given these restrictions, time can be represented with a time icon or other reference point such as a labeled time line in a background. A time representation of a time icon or other reference point allows the event to exist in time. In addition, the representation differentiates an event from other events which may otherwise be graphically equivalent.

(2) *Representing one interval*

When representing an interval, the duration of the interval can be shown or the starting or stopping time of the interval can be shown. In addition, both the duration and the starting or stopping time can be shown. A time representation of position will emphasize the duration of the interval. Showing the starting or stopping point of the interval reduces the interval to an event. Showing duration and the starting and stopping point can be accomplished by representing time with position and using a background representation such as a labeled time line as a reference point.

(3) *Representing a sequence of events*

When representing a sequence of events, we can show the succession of events or the duration between events. To emphasize succession, a motion technique can be used. An additional representation of a time icon provides a reference point for each state. Additional representations such as scaling or color provide further contrast among different events. Animationtrace and blinking provide a coexistence of past, present, and future and are useful if comparison between states is necessary. To emphasize duration between events, a time representation of position can be used. Again, a background representation of a labeled time line provides a reference point.

(4) *Representing a sequence of intervals*

In this situation, we can show duration of each interval, duration between intervals, starting or stopping times of each interval, or any combination of the three. Duration can be shown with a representation of position. Starting or stopping times can be shown with the same representations as used for a sequence of events. Combinations can be shown using position and a background labeled time line or with a combination of a motion technique and position.

(5) *Representing a sequence of events and intervals*

If there does not need to be a distinction between events and intervals the representation is the same as that for representing a sequence of intervals. If there does need to be a distinction, then the time representation should show duration. Possible representations include position, intrinsic position, and a combination of a motion technique and position.

(6) *Representing indeterminacy*

If indeterminate data should be distinguished from true data, the possible representations include blinking, fading, or dashed lines. These representations show the uncertainty in the data. Alternatively, indeterminacy can be ignored if indeterminate data need not be distinguished from true data.

The resolutions to the questions above involve choosing a representation for time which emphasizes duration, succession, or both. To represent time in terms of how it is experienced involves making changes between states more or less distinct. Choosing particular representations of time makes time be perceived more as flowing or more as a separation of events. For example, plotting data against a time axis makes time be perceived as flowing since the changes between states are less distinct. Also related is how the representation of objects change over time. For example, if an icon changes position over time, then the changes will seem distinct and time will be experienced as a separation of events. Alternatively, if an object changes intensity over time then the changes are less distinct and time appears as flowing.

11.4. Essential Features

This section discusses how the essential features of our approach listed in Chapter 4 relate to the goals for our display system.

(1) *Iconic Representations of Objects*

Providing iconic representation of objects allows the user to visualize data in different ways. Pictorial data is useful in conveying information because the brain processes pictures much faster than the equivalent text. Allowing the user to choose his own iconic representation assures that the actual information conveyed is what the user intends.

(2) *Graphical Data Independence*

Graphical data independence from application programs allows interactive specification of representations. The storing of information in a tailored data structure rather than a relational database allows faster access to information that is used often. This also provides the possibility of concurrent accessing of data and graphical descriptions thus creating a more efficient system. In addition it allows different users to have different description files for the same database.

(3) *Automatic Synthesis of Objects*

This feature allows the interactive generation of description for new data as well as allowing the user to experiment with several representations for existing data. This provides the opportunity for representing data differently for different applications.

(4) *Graphical Representation of Time*

This is by far the most important feature of our system. However, an effective representation for time depends on the existence of the other features listed above. Allowing multiple time representations and interactive specification of time representation allows the system to be used for various applications of temporal databases.

The Schema Editor allows the user to choose his own representation for static data and for time. The representation information, stored in the graphics knowledge base data structure, is independent from application programs. The data structure is tailored to support fast access to information. In addition, each user can have his own graphical description for the same database.

The Object Synthesis module supports the automatic synthesis of objects. Objects are created using information in the graphics knowledge base. The Interpreter interprets the graphical primitives contained in each object and makes the appropriate calls to display the object.

The Time Displayer applies the temporal modifiers and passes each object to the Interpreter. An object may be sent to the Interpreter multiple times if its representation changes over time.

The architecture of our prototype display system was designed to provide the user with the flexibility to interactively choose representations. This flexibility resulted in a system less efficient than systems

where the representations are hard-coded into the application programs. The next section discusses the efficiency of our system and possible improvements.

11.5. Efficiency

Efficiency in software systems often involves space-time tradeoffs. In our interactive display system, time efficiency had higher priority than space efficiency. However, space efficiency considerations are also important, especially when displaying large databases.

The implementation of the system was facilitated by the use of IDL notation and the IDL translator. The IDL specifications of the data structures were automatically translated into C language declarations by the IDL translator providing fast prototyping.

We first discuss efficiency of the system using the most straightforward implementation. We next suggest further optimizations.

Representations for data structures affect both time and space. The IDL translator maps the IDL specification into a C declarations file. Nodes are represented as C structures. Classes are represented as C unions. Sequences and sets are represented as linked lists. The IDL basic types, `Integer`, `Boolean`, `Rational` are represented as the C types `int`, `char`, and `float`. IDL `String`s are maintained in a hash table with only one copy of each string value resulting in a savings of both time as well as space since string comparison operations required only a test for pointer equivalence.

The time for displaying tuples is divided among the Object Synthesis module, the Time Displayer module, and the Interpreter. The time for each module is shown in Figure 11.1 for one tuple, two tuples, and eight tuples.

Figure 11.1: Time for Displaying Tuples of Relation `airplanestatus`

#tuples	Object Synthesis	Time Displayer	Interpreter	Total Time per tuple
1	0.02 sec	0.01 sec	0.01 sec	0.04
2	0.04 sec	0.05 sec	0.03 sec	0.06
8	0.12 sec	0.07 sec	0.13 sec	0.04

Also important in the analysis is the use of each module. The Object Synthesis time increases at a rate less than the increase in the number of tuples because tuples of the same relation share a portion of

their representation. The Time Displayer and Interpreter time can increase at a rate faster than the increase in the number of tuples because each tuple may have to be interpreted more than once if its representation changes over time.

Several optimizations are possible. For example, 16% of the time spent in the object synthesis module is due to storage allocation calls to the C memory allocator, *malloc*. Time for allocation could be decreased by providing a tailored memory allocator that was especially efficient for allocating certain sizes, namely the sizes for objects. The allocator could allocate 100 objects at a time. The first creation of an object would require a call to malloc but the next 99 calls would only require a pointer assignment. This will be done in the next version of the IDL runtime system.

The time delay in the Time Displayer and Interpreter modules could be decreased by using a tailored graphics package for displaying objects. The system currently uses the SUN core graphics package. A disadvantage of this package is that a segment cannot be constructed using portions of another segment. Therefore, a full interpretation is required for each object even if it shares portions of its representation with another previously interpreted object. Also, an object which changes over time must be interpreted several times. For example, if the xposition of an object has four different values in an interval, four segments must be created which differ only in the viewport. A tailored graphics package could provide for sharing of segment portions.

Space efficiency in data structures could be increased both by changes in the IDL specification and in changing the representations of unattributed nodes and integers. For example, the tuple structure described in Section 9.1 had a very inefficient representation in that attribute names are contained in both the relation nodes and tuple nodes. A more efficient representation could be to just list attribute values in the tuple nodes. Another change for space efficiency could be to represent a time stamp as a string rather than as a node containing six integers. For example, the time stamp "May 16, 1985 8:31:15" would be represented as "850516083115" using two digits each for year, month, day, hour, minute, and second values.

Changes in the representation of unattributed nodes and Integers are possible using representation specifications in IDL. One such specification is to represent unattributed nodes as enumerated types rather than C structures. Another specification is to represent integer attributes as C `shorts` or C characters. Both specifications would result in space savings but would require some overhead for accessing.

From the brief analysis of our system efficiency, we concluded that adequate efficiency is possible without compromising the flexibility to interactively choose representations. This efficiency is possible both through options in the IDL translator for generating different representations and through a tailored graphics package. The next section discusses possible extensions to the system.

11.6. Extensions

This section lists possible extensions to our system and gives a brief description of each, indicating its importance and its implementation difficulty.

(1) *Different levels of detail*

Providing the user with different levels of detail for the same data is a very effective technique in the display of data and may be necessary when displaying large databases. The graphics model can easily be extended to support different levels of detail. Each primitive in each of the object templates in the graphics knowledge base could contain a level at which it should be displayed. Temporal and attribute modifiers could change the level as well as the representation. The Interpreter would then only generate graphics calls for those primitives which had the appropriate level number.

(2) *Automatic resolution for object placement conflicts*

Providing automatic placement resolution could be an effective layout technique which also may be necessary when displaying large databases. One possibility is to use the third dimension so that conflicting objects are stacked (see also the next extension). A second possibility is to place conflicting objects in the closest non-occupied grid square. The screen model could be easily extended to support automatic placement resolution. Each square in the screen grid could include a bit indicating whether it was occupied. When a placement conflict occurred, the conflicting object could be stacked behind the occupying object or placed in the closest non-occupied square.

(3) *Use of the Third Dimension for Time Representation*

The third dimension could be used for representing time as well as other attributes. The different layers could be associated with different times. A progression through time would be possible by flipping through the layers much in the same way a stack of papers is looked through. This extension is not as important as the previous extensions since many representations are already supported for time. However, it would be an intersting enhancement. The graphics model could be extended to three dimensional space. Each coordinate would have an x, y, and z value. The z value could then be changed by temporal modifiers.

(4) *Incremental Synthesis of Objects*

The current system processes all objects at once. An interesting extension would be to allow an incremental synthesis and display of objects. This would allow for concurrency in the object synthesis, time display, and interpreter modules thus decreasing the perceived processing delay of the system. This change may be necessary for displaying large databases. The architecture of the system would have to be modified somewhat to support incremental processing. In particular, the object synthesis module would send each object to the time displayer immediately after it was fully created. This may result in some reduction of efficiency in the object synthesis module since all objects of the same template would not be processed together. Another modification would be in the representations of infinite attributes and time. Representations would no longer be able to

use the ordering of tuples on some attribute value since all tuples would not be available. Instead, absolute modification values would have to be used thus reducing infinite attributes to finite attributes. More study needs to be done to determine whether this loss in flexibility is compensated for by the decrease in processing delay.

(5) *Compilation of Representations*

The interactive specification of representations allows the system to be used for various applications. The disadvantage is that the efficiency can never be as good as when the representations are hard coded in the application program. An extension to our system would be to provide for compilation of chosen representations. The user could experiment with representations until the desired one was found. The system could then generate a file of graphical procedure calls by interpreting the object frame structure. The file could then be compiled. The resulting program would display the specified data much more efficiently than the flexible graphic system because it would not have to go through the object synthesis, time display, or interpreter modules. Compilation of representations would be useful if many users needed the same representation for a database. The interpreter could easily be extended to support this. Instead of making calls to the SUN core graphics package the interpreter could print these calls in a file, compile the file, and link in the SUN core library.

(6) *Design of Representation Language*

The Schema Editor commands are the basis for the interactive specification of representations. The design of the language could be much improved. One disadvantage is that it is too wordy to use in a practical way. The language could be designed to be more similar to TQuel or could possibly be combined with TQuel to allow for selection and representation at one time. A second disadvantage is that it is not general enough. In particular, it would be useful to be able to specify positions with expressions or with relative locations. This extension is a necessary addition to the system and would require only changes to the Schema Editor and possibly the Object Synthesis module.

CHAPTER 12

Conclusion

The primary goal of our research was to provide the flexibility to view temporal data in several ways. A secondary goal necessary for meeting the first goal was to provide the effective representation of static data. These goals were met by allowing the user to interactively specify representations for data and to experiment with different representations and combinations of representations. Efficiency considerations were secondary.

A graphics system can provide a convenient and powerful medium to display information in a temporal database. This thesis investigated techniques of varying function, generality, and performance and examined these techniques in several representative situations. The research improved on the techniques for representing static data and extended these techniques for displaying temporal data. The implementation of a prototype system demonstrates the feasibility of our approach.

Appendices

Appendices A, B, and C give the IDL specification for the graphics knowledge base, the object frame, and the tuple data structures. Appendix D gives the BNF for the Schema Editor commands. Appendix E contains the complete examples from which portions appear in the text. This appendix was generated automatically by a tool *gknbprint* which pretty prints a graphics knowledge base. Appendix F contains a glossary of words or phrases denoting important concepts.

Appendix A. Graphics Knowledge Base

```
Structure GraphicsKnowledgeBase Root graphicsknowledgebase Is

        graphicsknowledgebase => objecttemplates: Seq Of objecttemplate,
                                 screencoordinates: coordinatepair,
                                 background: Seq Of primitive,
                                 timerepresentation: Seq Of timerep,
                                 timestep: steptype,
                                 timespeed: speedtype,
                                 timemode: modetype,
                                 timeordering: orderingtype,
                                 timerange: rangetypeOrvoid,
                                 indeterminacyrep: indeterminacyrep;

        objecttemplate => name: String,
                          iconicrep: Seq Of primitive,
                          attributes: Seq Of attribute,
                          coordinates: coordinatepair,
                          xposition: Integer,
                          yposition: Integer,
                          width: Integer,
                          height: Integer,
                          color: color,
                          intensity: Rational;

        attribute ::= infinite_attribute | finite_attribute;
        attribute => name: String;

        finite_attribute => rep: Seq Of value_actions;
        value_actions => value: String,
                         actions: Seq Of action;
        action => applyto: objecttemplate,
                  code: primitive;

        infinite_attribute => rep: Seq Of infinite_attributerep;
        infinite_attributerep ::= text | intrinsicrep | functionrep | objecttemplate;

        intrinsicrep => ;       --attribute value is used as part of iconic rep

        functionrep => applyto: objecttemplate,
                       code: primitive,
                       ordering: orderingtype,
                       range: rangetypeOrvoid;

        orderingtype ::= forward | reverse;
        forward =>;
        reverse =>;

        primitive ::= iconicprimitive |
                      transformativeprimitive | stackprimitive;
        primitive => dependent: Boolean;   -- true if depends on some att value

        iconicprimitive ::= pointer | line | circle | point | polygon |
                            icon | curve | text;

        transformativeprimitive ::= colorprimitive | geometricprimitive;
```

```
colorprimitive ::= color | colorscale | intensity;
geometricprimitive ::= rotate | scale | translate | position;
stackprimitive ::= PUSHCPT | POPCPT | PUSHCOLOR | POPCOLOR;

position ::= xposition | yposition;
position => value: numbertypeOrvoid;
xposition =>;
yposition =>;

rangetype => lowvalue: Integer,
             highvalue: Integer;

pointer => vertex1: point,
           vertex2: point;
line => vertex1: point,
        vertex2: point;
circle => filled: Boolean,
          radius: numbertype,
          center: point;
polygon => filled: Boolean,
           vertices: Seq Of point;

color => name: String,
         index: Integer,
         r: Integer,
         b: Integer,
         g: Integer;
colorscale => name: String;

curve => points: Seq Of point;
icon => name: String,
        position: point,
        size: point;

text => value: String,
        position: point,
        size: point;

intensity => ival: numbertype,
             range: rangetype;

rotate => angle: numbertype;
scale => xs: numbertype,
         ys: numbertype;
translate => xt: numbertype,
             yt: numbertype;
PUSHCPT =>;
POPCPT =>;
PUSHCOLOR =>;
POPCOLOR =>;

point => x: numbertype,
         y: numbertype;
knownpoint => x: Integer,
              y: Integer;
coordinatepair => lowercorner: knownpoint,
                  uppercorner: knownpoint;
```

```
        numbertype ::= number | objectattributepr;
        number => num: Rational;
        objectattributepr => objname: String,
                             attname: String;


        timerep ::= geometricprimitive | colorscale | intensity |
                    motiontype | time_icon |
                    intrinsicxpos | intrinsicypos | void;

        time_icon ::= clock_icon | month_icon | year_icon;
        time_icon => position: point,
                     size: point;
        clock_icon ::= clockface_icon | digitalclock_icon;
        clockface_icon =>;
        digitalclock_icon =>;
        month_icon =>;
        year_icon =>;

        motiontype ::= animate | animatetrace | blinking;
        animate =>;
        animatetrace =>;
        blinking =>;
        intrinsicxpos =>;
        intrinsicypos =>;

        steptype ::= starttime | stoptime | unit;
        starttime =>;
        stoptime =>;
        unit ::= second | minute | hour | day | month | year;
        unit => val: Integer;
        second =>;
        minute =>;
        hour =>;
        day =>;
        month =>;
        year =>;

        modetype ::= continuous | stop;
        continuous =>;
        stop =>;

        speedtype => val: Integer;

        indeterminacyrep ::= blinking | fading | dashedlines | void;
        fading =>;
        dashedlines =>;

        numbertypeOrvoid ::= numbertype | void;
        rangetypeOrvoid ::= rangetype | void;
        void =>;
        For void Use Enumerated;
End
```

Appendix B. Object Frame

```
Structure ObjectFrame Root objectframe
                    From GraphicsKnowledgeBase iconlist InTuples Is

    objectframe => knowledgebase: graphicsknowledgebase,
                   iconlist: icons,
                   activetuples: relations,
                   objects: Seq Of object;

    object => template: objecttemplate,
              tuple: tuple,
              rep: Seq Of primitive,
              textreps: Seq Of text,
              coordinates: coordinatepair,
              xposition: Integer,
              yposition: Integer,
              height: Integer,
              width: Integer,
              color: color,
              intensity: Rational,
              starttimevalue: Integer,
              stoptimevalue: Integer;

    relations => lowtimevalue: Integer,
                 hightimevalue: Integer;

    attributetype => lowval: attributepair,
                     highval: attributepair;

    attributepair => numericval: Rational,
                     rep: Seq Of primitive;

    tuple => object: object;
End
```

Appendix C. Tuple Structure

```
Structure InTuples Root relations Is

        relations => list: Seq Of relation;
        relation => name: String,
                    tuples: Seq Of tuple,
                    attributes: Seq Of attributetype;

        attributetype => name: String,
                         type: valuetype;

        valuetype ::= numericvalue | stringvalue;
        numericvalue =>;
        stringvalue =>;

        tuple => key: String,
                 attributes: Seq Of attributepair,
                 time: timetype;

        attributepair => name: String,
                         value: String,
                         from: tupleORvoid;

        tupleORvoid ::= tuple | void;
        void =>;
        For void Use Enumerated;

        timetype ::= event | interval | static;
        static =>;
        event => at: timestamp;
        interval => start: timestamp,
                    stop: timestamp,
                    indeterminate: Boolean;

        timestamp => year: Integer,
                     month: Integer,
                     day: Integer,
                     hour: Integer,
                     minute: Integer,
                     second: Integer;
End
```

Appendix D. Schema Editor Command Syntax

```
<schemaEd cmd>  ::= {<range cmd>}* {<represent cmd>}* {<set cmd>}* {<where cmd>}*

<range cmd> ::= range of <control> is <objecttemplate>

<represent cmd> ::=
                represent time with {<time_rep>}+ {ordering <ordertype>}?
                                {step = <step_type>}? {mode = <mode_type>}?
                                {speed = <speed_type>}?
                represent <control> with { <iconicprimitive> }+
                represent <control> . <attname> with { <infrep> }+
                represent <control> . <attname> = <attval> with { <finrep> }+
                represent background with { <finrep> }+
                represent indeterminacy with <indetrep>

<set cmd> ::= setting <control>.<numericcharacteristic> to <num>
              setting <control>.coordinates to <int>,<int> to <int>,<int>
              setting <control>.color to <name>

<where cmd> ::= where <condition>
<condition> ::= <control>.<attOrchar> = <control>.<attOrchar>
<attOrchar> ::= <attname>
                <characteristic>

<time_rep> ::= <geometricprimitive>
                colorscale
                intensity
                blinking
                animationtrace
                animation
                clockface_icon
                digitalclock_icon
                month_icon
                year_icon
                nothing

<step_type> ::=starttime
                stoptime
                <int> second(s)
                <int> minute(s)
                <int> hour(s)
                <int> day(s)
                <int> month(s)
                <int> year(s)

<mode_type> ::= stop
                continuous

<speed_type> ::= <int>          --from 0-10 where 0 is the least amount of time
                                --between frames
<indetrep>  ::= blinking
                fading
                dashedlines
                nothing
```

```
<infrep>::=    <geometricprimitive> {ordering <ordertype>}? {range <rangetype>}?
               intensity {ordering <ordertype>}? {range <rangetype>}?
               colorscale <name> {ordering <ordertype>}?
               text at <point>

<finrep> ::=   <iconicprimitive>
               rotate <num>
               scale <num> <num>
               translate <num> <num>
               xposition <num>
               yposition <num>
               color <name>
               intensity <num>

<geometricprimitive> ::=
               rotate
               scale
               translate
               xposition
               yposition

<iconicprimitive> ::= point <point>
                      line <point> to <point>
                      pointer <point> to <point>
                      curve <point> {<point>}*
                      polygon <point> {<point>}*
                      circle center <point> radius <numtype>
                      icon <name> position <point> size <point>
                      text <name> at <point>

<ordertype> ::=forward
               reverse

<rangetype> ::=<num> to <num>


<point> ::=    <numtype> , <numtype>
<numtype> ::=  <num>
               <objtemplateref> . <attname>
               <objtemplateref> . <numericcharacteristic>
<objtemplateref> ::=   <objecttemplate>
                       <control>

<objecttemplate> ::= <name>
<attname> ::=  <name>
<attvalue> ::= <name>
<control>      ::= <name>

<numericcharacteristic> ::=   xposition
                              yposition
                              height
                              width
                              intensity
```

Appendix E. Schema Editor Commands for Examples

```
-------------------------------------------------
Representation for airplanestatus
-------------------------------------------------
airplanestatus(plane, model, status):

range of a is airplanestatus
represent a with icon plane
                    position (10.00,10.00) size (80.00,80.00)
represent a.model = 414A with icon wingpropeller
                    position (30.00,35.00) size (5.00,30.00)
                icon wingpropeller
                    position (65.00,35.00) size (5.00,30.00)
represent a.model = ArcherII with icon nosepropeller
                    position (45.00,85.00) size (10.00,5.00)
represent a.status = inrepair with yposition 1.00
                icon garage
                    position (5.00,5.00) size (90.00,90.00)
represent a.status = onground with yposition 1.00
represent a.status = trainingflight with yposition 2.00
represent a.status = demoflight with yposition 3.00
represent a.status = freeflight with yposition 4.00
represent a.plane with text at (37.00,50.00) size (4.00,4.00)
                  xposition
              ordering = forward
              range = 1 to 5
       setting a.xposition = 0
       setting a.yposition = 0
       setting a.width = 1
       setting a.height = 1
       setting a.color = black
       setting a.intensity = 0.50
       setting a.coordinates to 0,0 to 100,100

set screen.coordinates to  0,0 to 8,6
represent time with animation
                    time_icon clockface position (5.00,4.00) size (1.00,1.00)
       step = 30 minutes
       speed = 0
       mode = stop
       ordering = forward
represent indeterminacy with nothing
```

```
-------------------------------------------------
Alternate Representation for time for airplanestatus
-------------------------------------------------
airplanestatus(plane, model, status):

range of a is airplanestatus
represent a with icon plane
                        position (10.00,10.00) size (80.00,80.00)
represent a.model = 414A with color red
                    icon wingpropeller
                        position (30.00,35.00) size (5.00,30.00)
                    icon wingpropeller
                        position (65.00,35.00) size (5.00,30.00)
represent a.model = ArcherII with icon nosepropeller
                        position (45.00,85.00) size (10.00,5.00)
                    color blue
represent a.status = inrepair with yposition 1.00
                    icon garage
                        position (5.00,5.00) size (90.00,90.00)
represent a.status = onground with yposition 1.00
represent a.status = trainingflight with yposition 2.00
represent a.status = demoflight with yposition 3.00
represent a.status = freeflight with yposition 4.00
represent a.plane with text at (38.00,50.00) size (4.00,4.00)
                      xposition
                ordering = forward
                range = 1 to 5
        setting a.xposition = 0
        setting a.yposition = 0
        setting a.width = 1
        setting a.height = 1
        setting a.color = black
        setting a.intensity = 0.30
        setting a.coordinates to 0,0 to 100,100

set screen.coordinates to  480,-2 to 720,6
represent background with line (480.00,0.00) to (720.00,0.00)
                           line (480.00,0.00) to (480.00,6.00)
                           line (480.00,-0.20) to (480.00,0.20)
                           line (510.00,-0.20) to (510.00,0.20)
                           line (540.00,-0.20) to (540.00,0.20)
                           line (570.00,-0.20) to (570.00,0.20)
                           line (600.00,-0.20) to (600.00,0.20)
                           line (630.00,-0.20) to (630.00,0.20)
                           line (660.00,-0.20) to (660.00,0.20)
                           line (690.00,-0.20) to (690.00,0.20)
                           line (720.00,-0.20) to (720.00,0.20)
                           text "8:00" at (480.00,-0.50) size (2.00,0.20)
                           text "8:30" at (506.00,-0.50) size (2.00,0.20)
                           text "9:00" at (536.00,-0.50) size (2.00,0.20)
                           text "9:30" at (566.00,-0.50) size (2.00,0.20)
                           text "10:00" at (596.00,-0.50) size (2.00,0.20)
                           text "10:30" at (626.00,-0.50) size (2.00,0.20)
                           text "11:00" at (656.00,-0.50) size (2.00,0.20)
                           text "11:30" at (686.00,-0.50) size (2.00,0.20)
                           text "12:00" at (716.00,-0.50) size (2.00,0.20)
represent time with xposition
```

```
        step = 30 minutes
        speed = 5
        mode = stop
        ordering = forward
represent indeterminacy with nothing
```

```
-------------------------------------------------
Representation for Indeterminacy
-------------------------------------------------
Jflights(plane):

range of J is Jflights
represent J with icon plane
                      position (10.00,10.00) size (80.00,80.00)
represent J.plane with text at (38.00,50.00) size (4.00,4.00)
        setting J.xposition = 0
        setting J.yposition = 0
        setting J.width = 1
        setting J.height = 1
        setting J.color = black
        setting J.intensity = 0.10
        setting J.coordinates to 0,0 to 100,100

set screen.coordinates to  540,-1 to 930,7
represent background with line (540.00,0.00) to (930.00,0.00)
                          line (540.00,0.00) to (540.00,15.00)
                          line (570.00,-0.20) to (570.00,0.20)
                          line (600.00,-0.20) to (600.00,0.20)
                          line (630.00,-0.20) to (630.00,0.20)
                          line (660.00,-0.20) to (660.00,0.20)
                          line (690.00,-0.20) to (690.00,0.20)
                          line (720.00,-0.20) to (720.00,0.20)
                          line (750.00,-0.20) to (750.00,0.20)
                          line (780.00,-0.20) to (780.00,0.20)
                          line (810.00,-0.20) to (810.00,0.20)
                          line (840.00,-0.20) to (840.00,0.20)
                          line (870.00,-0.20) to (870.00,0.20)
                          line (900.00,-0.20) to (900.00,0.20)
                          line (930.00,-0.20) to (930.00,0.20)
                          text "9:00" at (540.00,-0.50) size (2.00,0.20)
                          text "9:30" at (565.00,-0.50) size (2.00,0.20)
                          text "10:00" at (595.00,-0.50) size (2.00,0.20)
                          text "10:30" at (625.00,-0.50) size (2.00,0.20)
                          text "11:00" at (655.00,-0.50) size (2.00,0.20)
                          text "11:30" at (685.00,-0.50) size (2.00,0.20)
                          text "12:00" at (715.00,-0.50) size (2.00,0.20)
                          text "12:30" at (745.00,-0.50) size (2.00,0.20)
                          text "1:00" at (775.00,-0.50) size (2.00,0.20)
                          text "1:30" at (805.00,-0.50) size (2.00,0.20)
                          text "2:00" at (835.00,-0.50) size (2.00,0.20)
                          text "2:30" at (865.00,-0.50) size (2.00,0.20)
                          text "3:00" at (895.00,-0.50) size (2.00,0.20)
                          text "3:30" at (925.00,-0.50) size (2.00,0.20)
represent time with xposition
        step = 30 minutes
        speed = 5
        mode = stop
        ordering = forward
represent indeterminacy with dashedlines
```

```
-------------------------------------------------------
Representation for first game plays
-------------------------------------------------------
firstgameplays(typeplay, startpos, stoppos):

range of f is firstgameplays
represent f with line (f.from,f.startpos) to (f.to,f.stoppos)
represent f.startpos with intrinsicrep
represent f.stoppos with intrinsicrep
represent f.typeplay = forwardpass with intensity 1.00
represent f.typeplay = lateral with intensity 0.60
represent f.typeplay = running with intensity 0.20
        setting f.xposition = -10
        setting f.yposition = -10
        setting f.width = 130
        setting f.height = 65
        setting f.color = black
        setting f.intensity = 1.00
        setting f.coordinates to -10,-10 to 120,55

set screen.coordinates to  -10,-10 to 120,55
represent background with line (0.00,0.00) to (0.00,100.00)
                          line (0.00,0.00) to (120.00,0.00)
                          line (-1.00,10.00) to (1.00,10.00)
                          line (-1.00,20.00) to (1.00,20.00)
                          line (-1.00,30.00) to (1.00,30.00)
                          line (-1.00,40.00) to (1.00,40.00)
                          line (-1.00,50.00) to (1.00,50.00)
                          line (-1.00,60.00) to (1.00,60.00)
                          line (-1.00,70.00) to (1.00,70.00)
                          line (-1.00,80.00) to (1.00,80.00)
                          line (-1.00,90.00) to (1.00,90.00)
                          line (-1.00,100.00) to (1.00,100.00)
                          line (10.00,-1.00) to (10.00,1.00)
                          line (20.00,-1.00) to (20.00,1.00)
                          line (30.00,-1.00) to (30.00,1.00)
                          line (40.00,-1.00) to (40.00,1.00)
                          line (50.00,-1.00) to (50.00,1.00)
                          line (60.00,-1.00) to (60.00,1.00)
                          line (70.00,-1.00) to (70.00,1.00)
                          line (80.00,-1.00) to (80.00,1.00)
                          line (90.00,-1.00) to (90.00,1.00)
                          line (100.00,-1.00) to (100.00,1.00)
                          line (110.00,-1.00) to (110.00,1.00)
                          line (120.00,-1.00) to (120.00,1.00)
                          text "time in seconds" at (40.00,-8.00) size (2.00,2.00)
                          text "00:20" at (18.00,-3.00) size (1.00,1.00)
                          text "00:40" at (38.00,-3.00) size (1.00,1.00)
                          text "01:00" at (58.00,-3.00) size (1.00,1.00)
                          text "01:20" at (78.00,-3.00) size (1.00,1.00)
                          text "01:40" at (98.00,-3.00) size (1.00,1.00)
                          text "10" at (-5.00,10.00) size (1.00,1.00)
                          text "20" at (-5.00,20.00) size (1.00,1.00)
                          text "30" at (-5.00,30.00) size (1.00,1.00)
                          text "40" at (-5.00,40.00) size (1.00,1.00)
                          text "50" at (-5.00,50.00) size (1.00,1.00)
represent time with nothing
```

```
        step = 1 second
        speed = 2
        mode = continuous
        ordering = forward
represent indeterminacy with nothing
```

```
-------------------------------------------------------
Representation for plays distance
-------------------------------------------------------
playsdistance(typeplay, totaldistance):

range of p is playdistance
represent p with polygon (p.from,p.totaldistance) (p.to,p.totaldistance)
                         (p.to,0.00) (p.from,0.00)

represent p.totaldistance with intrinsicrep
represent p.typeplay = forwardpass with color blue
represent p.typeplay = lateral with color red
represent p.typeplay = running with color green
        setting p.xposition = -10
        setting p.yposition = -10
        setting p.width = 130
        setting p.height = 30
        setting p.color = black
        setting p.intensity = 1.00
        setting p.coordinates to -10,-10 to 120,20

set screen.coordinates to  -10,-10 to 120,20
represent background with line (0.00,0.00) to (0.00,100.00)
                          line (0.00,0.00) to (120.00,0.00)
                          line (-1.00,5.00) to (1.00,5.00)
                          line (-1.00,10.00) to (1.00,10.00)
                          line (-1.00,15.00) to (1.00,15.00)
                          line (-1.00,20.00) to (1.00,20.00)
                          line (-1.00,25.00) to (1.00,25.00)
                          line (10.00,-1.00) to (10.00,1.00)
                          line (20.00,-1.00) to (20.00,1.00)
                          line (30.00,-1.00) to (30.00,1.00)
                          line (40.00,-1.00) to (40.00,1.00)
                          line (50.00,-1.00) to (50.00,1.00)
                          line (60.00,-1.00) to (60.00,1.00)
                          line (70.00,-1.00) to (70.00,1.00)
                          line (80.00,-1.00) to (80.00,1.00)
                          line (90.00,-1.00) to (90.00,1.00)
                          line (100.00,-1.00) to (100.00,1.00)
                          line (110.00,-1.00) to (110.00,1.00)
                          line (120.00,-1.00) to (120.00,1.00)
                          text "time in seconds" at (40.00,-8.00) size (2.00,2.00)
                          text "00:20" at (18.00,-3.00) size (1.00,1.00)
                          text "00:40" at (38.00,-3.00) size (1.00,1.00)
                          text "01:00" at (58.00,-3.00) size (1.00,1.00)
                          text "01:20" at (78.00,-3.00) size (1.00,1.00)
                          text "01:40" at (98.00,-3.00) size (1.00,1.00)
                          text "5" at (-5.00,5.00) size (1.00,1.00)
                          text "10" at (-5.00,10.00) size (1.00,1.00)
                          text "15" at (-5.00,15.00) size (1.00,1.00)
represent time with nothing
        step = 1 second
        speed = 2
        mode = continuous
        ordering = forward
represent indeterminacy with nothing
```

```
-------------------------------------------------
Representation for first touchdown plays
-------------------------------------------------
first_touchdownplays(typeplay, startpos, distance):

range of f is firsttouchdown_plays
represent f.typeplay = running with icon barbell
                       position (f.startpos,f.from) size (f.distance,1.00)
represent f.typeplay = lateral with icon dotted_barbell
                       position (f.startpos,f.from) size (f.distance,1.00)
represent f.typeplay = forwardpass with icon curved_barbell
                       position (f.startpos,f.from) size (f.distance,1.00)
represent f.distance with intrinsicrep
represent f.startpos with intrinsicrep
        setting f.xposition = 50
        setting f.yposition = 165
        setting f.width = 70
        setting f.height = 54
        setting f.color = black
        setting f.intensity = 0.50
        setting f.coordinates to 50,165 to 120,219

set screen.coordinates to  50,162 to 120,219
represent background with line (52.00,164.00) to (120.00,164.00)
                          line (52.00,164.00) to (52.00,219.00)
                          line (60.00,163.50) to (60.00,164.50)
                          line (70.00,163.50) to (70.00,164.50)
                          line (80.00,163.50) to (80.00,164.50)
                          line (90.00,163.50) to (90.00,164.50)
                          line (100.00,163.50) to (100.00,164.50)
                          line (110.00,163.50) to (110.00,164.50)
                          text "60" at (59.50,162.50) size (0.40,0.50)
                          text "70" at (69.50,162.50) size (0.40,0.50)
                          text "80" at (79.50,162.50) size (0.40,0.50)
                          text "90" at (89.50,162.50) size (0.40,0.50)
                          text "100" at (99.50,162.50) size (0.40,0.50)
                          text "110" at (109.50,162.50) size (0.40,0.50)
                          text "2:50" at (50.00,170.00) size (0.40,1.00)
                          text "3:00" at (50.00,180.00) size (0.40,1.00)
                          text "3:10" at (50.00,190.00) size (0.40,1.00)
                          text "3:20" at (50.00,200.00) size (0.40,1.00)
                          text "3:30" at (50.00,210.00) size (0.40,1.00)
represent time with nothing
        step = 1 second
        speed = 1
        mode = stop
        ordering = forward
represent indeterminacy with nothing
```

```
-------------------------------------------------
Representation for Monitoring Example
-------------------------------------------------
process(id, state):
processor(id):
mailbox(id):
runningon(process, processor):
sendmessage(process, mailbox):
waiting(process, mailbox):

range of p is process
represent p with circle center (50.00,50.00) radius 25.00
point (25.00,)
represent p.id with text at (45.00,50.00) size (8.00,8.00)
                    yposition
                ordering = forward
                range = 0 to 4
represent p.state = Ready with intensity 0.70
represent p.state = Running with intensity 1.00
represent p.state = Blocked with intensity 0.40
represent p.state = Done with intensity 0.10
        setting p.xposition = 0
        setting p.yposition = 0
        setting p.width = 1
        setting p.height = 1
        setting p.color = black
        setting p.intensity = 0.40
        setting p.coordinates to 0,0 to 100,100

range of p is processor
represent p with icon rectangle
                    position (10.00,10.00) size (70.00,90.00)
represent p.id with text at (60.00,80.00) size (8.00,8.00)
                    yposition
                ordering = forward
                range = 4 to 8
        setting p.xposition = 0
        setting p.yposition = 0
        setting p.width = 1
        setting p.height = 1
        setting p.color = black
        setting p.intensity = 0.40
        setting p.coordinates to 0,0 to 100,100

range of m is mailbox
represent m with icon oval
                    position (30.00,25.00) size (40.00,50.00)
represent m.id with text at (40.00,50.00) size (8.00,8.00)
                    yposition
                ordering = forward
                range = 1 to 8
        setting m.xposition = 10
        setting m.yposition = 0
        setting m.width = 1
        setting m.height = 1
        setting m.color = black
        setting m.intensity = 0.40
```

```
        setting m.coordinates to 0,0 to 100,100

range of r is runningon
represent r.processor with processor r
represent r.process with process p
                    yposition r.yposition
                ordering = forward
        setting r.xposition = 0
        setting r.yposition = 0
        setting r.width = 1
        setting r.height = 1
        setting r.color = black
        setting r.intensity = 0.40
        setting r.coordinates to 0,0 to 100,100

range of s is sendmessage
represent s with pointer (p.xposition,p.yposition) to (m.xposition,m.yposition)
represent s.mailbox with mailbox m
represent s.process with process p
        setting s.xposition = 0
        setting s.yposition = 0
        setting s.width = 12
        setting s.height = 8
        setting s.color = black
        setting s.intensity = 0.50
        setting s.coordinates to 0,0 to 12,8

range of w is waiting
represent w with pointer (m.xposition,m.yposition) to (p.xposition,p.yposition)
represent w.mailbox with mailbox m
represent w.process with process p
        setting w.xposition = 0
        setting w.yposition = 0
        setting w.width = 12
        setting w.height = 8
        setting w.color = black
        setting w.intensity = 0.50
        setting w.coordinates to 0,0 to 12,8

set screen.coordinates to  0,0 to 12,8
represent time with animationtrace
                    time_icon digitalclock position (11.00,7.00) size (1.00,1.00)
        step = 1 second
        speed = 1
        mode = stop
        ordering = forward
represent indeterminacy with nothing
```

Appendix F. Glossary

attribute Column of a relation "table".

attribute (IDL) A named value with a domain specified by an attribute type.

attribute type The type of an IDL attribute which can be a basic type, a set or sequence, a node type, or a class type.

basic type The IDL types of boolean, integer, rational, and string.

class A collection of IDL nodes sharing common aspects.

coordinate pair The portion of the screen template which specifies the lower corner and upper corner of the screen grid.

current position transformation
A matrix which defines the current transformations which are applied to each graphical primitive.

database schema A set of relation schema.

description of time
The area of human perception of time which is concerned with how one describes time. Time is described in terms of duration and/or succession.

display aiding A technique for representing time which displays a trail of displacements of an object which fade over time.

domain A set of values such as integers or character strings which are valid for an attribute.

duration The length of time.

entity relation A relation which contains an entity identifier or key plus other descriptive attributes.

event Data which is valid at a instance of time.

experience of time
The area of human perception of time which is concerned with how one interprets temporal information. Time is experienced as flowing or as a succession of events.

finite attribute Attributes of a relation with finite domains.

frame buffer A rectangle of pixels containing values.

graphic characteristic
Graphical aspects of an object such as color and size.

graphics knowledge base
The data structure containing a set of object templates and a set of temporal characteristics. This structure is used to construct an object for each active tuple.

historical database
A database which contains the time when the information being modeled was valid.

historical event relation
A relation augmented with one temporal attribute named `at`.

historical interval relation
A relation augmented with two temporal attributes named `from` and `to`.

historical relational database
A relational database with one or two temporal attributes added to each relation schema.

iconic representation
A graphical shape such as an icon, line or polygon.

iconic template The template defining a graphical shape.

IDL The Interface Description Language used to specify data [Nestor, et al. 1982].

indeterminate data
Information in a database which may contain incomplete or incorrect information.

infinite attribute
Attributes of a relation with infinite domains.

Interpreter The module of the display system which interprets the embedded graphical primitives of each object.

interval Data which is valid within an interval of time.

intrinsic position modifier
A special modifier which uses the value of an attribute as the x or y coordinate of an iconic representation thus changing the intrinsic structure of the representation.

key The identifier for a relation consisting of one or more attributes. No two tuples of a relation agree on all attributes of the key.

members The elements of an IDL class.

modifier The definition of what type of modification should be made to an object under what conditions. Modifications include changing graphical characteristics, adding iconic representations, and changing the intrinsic structure.

node A named collection of 0 or more IDL attributes.

object The structure which contains the information for the graphics model. This information includes iconic representations and graphical characteristics. Objects are created using object templates.

object frame A collection of objects constructed from the object templates in the graphics knowledge base.

object template The defining template for an object which contains iconic templates, modifiers, and graphical characteristics.

primitive image A graphical description containing graphical primitives such as color, polygons, and size defined with a unit square.

Quel The query language for Ingres [Stonebraker et al. 1976].

relation A subset of the Cartesian product of a list of domains.

relation schema The definition for a relation which contains the name of the relation and the name and domains of its attributes.

relational database
A set of relation specified by a database schema.

relationship relation
A relation containing two or more entity identifiers plus other descriptive attributes.

rollback database
A database which contains all past states of the static database as it is updated over time.

Schema Editor The module of the display system which constructs the graphics knowledge base given user commands.

screen template The definition for a screen which includes the coordinate pair of the screen plus a sequence of primitive images.

segment A portion of the entire screen containing a sequence of scaled images.

set-theoretic relation
The mathematical concept underlying the relational model which is a subset of the Cartesian product of a list of domains.

static database A database which is updated by replacing information resulting in data values only at the "current" time.

static state A sequence of objects and the time that these objects are valid. The valid time can be an instant of time or an interval of time.

structure A collection of IDL nodes and classes.

structured type The types of set and sequence which are provided in IDL notation.

succession A sequence of time values.

temporal characteristic
Characteristics of time including the representation of time and properties of time such as the direction and the interval step.

temporal database
A database which contains both valid and transaction time for information.

temporal modifier
Special modifiers which use the time value for a tuple to determine if conditions are met and modifications should be made.

Time Displayer The module of the display system which manages the display of time.

TQuel A temporal query language designed by Richard Snodgrass [Snodgrass & Ahn 1986].

transaction time The time information was stored in the database.

true data Information in a database which is believed to be accurate.

tuple Rows of a relation "table."

valid time The time when the information in a database is valid.

viewport The portion of the frame buffer which can be written into.

Bibliography

[Ariav & Morgan 1982] Ariav, G. and H. L. Morgan. *MDM: Embedding the Time Dimension in Information Systems.* TR 82-03-01. Department of Decision Sciences, The Wharton School, University of Pennsylvania. 1982.

[Bertin 1981] Bertin, J. *Graphics and Graphic Information Processing.* New York: Walter de Gruyter, 1981.

[Brown & Sedgewick 1984] Brown, Marc H. and Robert Sedgewick. *A System for Algorithm Animation. ACM SIGGRAPH Proceedings,* , July 1984, pp. 177-186.

[Chen 1976] Chen, P. P-S. *The Entity-Relationship Model -- Toward a Unified View of Data. ACM Transactions on Database Systems,* 1, No. 1, Mar. 1976, pp. 9-36.

[Codd 1972] Codd, E. F. *Relational Completeness of Data Base Sublanguages,* in Data Base Systems. Vol. 6 of Courant Computer Symposia Series. Englewood Cliffs, N.J.: Prentice Hall, 1972. pp. 65-98 .

[Codd 1970] Codd, E.F. *A Relational Model of Data for Large Shared Data Bank. Communications of the Association of Computing Machinery,* 13, No. 6, June 1970, pp. 377-387.

[Donelson 1978] Donelson, W. C. *Spatial Management of Information. ACM SIGGRAPH Proceedings,* , August 1978, pp. 203-209.

[Friedell, et al 1982] Friedell, M., J. Barnett and D. Kramlich. *Context-Sensitive Graphic Presentation of Information. ACM SIGGRAPH Proceedings,* , July 1982, pp. 181-188.

[Friedell 1984] Friedell, Mark *Automatic Synthesis of Graphical Object Descriptions. ACM SIGGRAPH Proceedings,* , July 1984, pp. 53-62.

[Held et al. 1975] Held, G.D., M. Stonebraker and E. Wong. *INGRES--A relational data base management system. Proceedings of the 1975 National Computer Conference,* 44 (1975), pp. 409-416.

[Herot 1980] Herot, C. *Spatial Management of Data. ACM Transactions on Database Systems,* 5, No. 4, December 1980, pp. 493-514.

[Herot, et al 1980] Herot, C., R. Carling, M. Friedell and D. Kramlich. *A Prototype Spatial Data Management System. ACM SIGGRAPH Proceedings,* (1980).

[Kramlich, et al 1983] Kramlich, D., G. Brown, R. Carling and C. Herot. *Program Visualization: Graphics Support for Software Development. IEEE Proceeding of the 20TH Design Automation Conference,* (1983), pp. 143-148.

[Kummel 1966] Kummel, Friedrich *Time as Succession and the Problem of Duration,* in The Voices of Time. New York, New York: George Braziller, Inc., 1966.

[Lodding 1983] Lodding, K. N. *Iconic Interfacing. IEEE Computer Graphics and Applications,* 3, No. 2 (1983), pp. 11-20.

[Morse 1979] Morse, A. *Some Principles for the Effective Display of Data. ACM SIGGRAPH Proceedings,* , August 1979, pp. 94-100.

[Nestor, et al. 1982] Nestor, J.R., W.A. Wulf and D.A. Lamb. *IDL Formal Description, Part I.* SoftLab Document No. 2. Computer Science Department, University of North Carolina at Chapel Hill. June 1982.

[Newman & Sproull 1979] Newman, W.M. and Sproull R.F.. *Principles of Interactive Computer Graphics.*

New York, New York: McGraw Hill Book Company, 1979.

[Shallis 1983] Shallis, M. *On Time: An Investigation into Scientific Knowledge and Human Experience.* New York: Schocken Books, 1983.

[Snodgrass 1982] Snodgrass, R. *Monitoring Distributed Systems: A Relational Approach.* PhD. Diss. Computer Science Department, Carnegie-Mellon University, Dec. 1982.

[Snodgrass 1984] Snodgrass, R. *The Temporal Query Language TQuel,* in *Proceedings of the Third ACM SIGAct-SIGMOD Symposium on Principles of Database Systems,* Waterloo, Ontario, Canada: Apr. 1984, pp. 204-212.

[Snodgrass 1985] Snodgrass, R. *A Relational Approach to Monitoring Complex Systems.* Technical Report TR85-035. Computer Science Department, University of North Carolina at Chapel Hill. Dec. 1985.

[Snodgrass 1986] Snodgrass, R. *A Temporal Query Language. ACM Transactions on Database Systems (to appear),* (1986).

[Snodgrass & Ahn 1986] Snodgrass, R. and I. Ahn. *Temporal Databases. IEEE Computer,* 19, No. 9, Sep. 1986.

[Stonebraker et al. 1976] Stonebraker, M., E. Wong, P. Kreps and G. Held. *The Design and Implementation of INGRES. ACM Transactions on Database Systems,* 1, No. 3, Sep. 1976, pp. 189-222.

[Tsichritzis & Lochovsky 1982] Tsichritzis, D. and F. Lochovsky. *Data Models.* New Jersey: Prentice-Hall, 1982.

[Ullmann 1982] Ullman, J. D. *Principles of Database Systems.* Rockville, Maryland: Computer Science Press, 1982.

[Weller & William 1976] Weller, D. and R. William. *Graphic and Relational Database Support for Problem Solving. Computer Graphics,* 10, No. 2, August 1976.

[Whitrow 1978] Whitrow, G.J. *The nature of time.* New York, NY: Holt Rinehart and Winston, 1978.

[Whitrow 1980] Whitrow, G.J. *The natural philosophy of time.* New York, NY: Oxford University Press, 1980.

[Yedwab, et al 1981] Yedwab, L., C. Herot, R.L. Rosenberg and C. Gross. *The Automated Desk. Proceeding of the Second Symposium on Small Systems,* (1981).