8-2004

A Data Model for Exploration of Temporal Virtual Reality Geographic Information Systems

Jorge Alberto Prado de Campos

A DATA MODEL FOR EXPLORATION OF TEMPORAL VIRTUAL REALITY GEOGRAPHIC INFORMATION SYSTEMS

By

Jorge Alberto Prado de Campos

B.S. Universidade Federal da Bahia - Brazil, 1986

M.S. Pontifícia Universidade Católica do Rio de Janeiro - Brazil, 1991

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

(in Spatial Information Science and Engineering)

The Graduate School

The University of Maine

August, 2004

A DATA MODEL FOR EXPLORATION OF TEMPORAL VIRTUAL REALITY GEOGRAPHIC INFORMATION SYSTEMS

By Jorge Alberto Prado de Campos

Thesis Advisor: Dr. Max J. Egenhofer

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy
(in Spatial Information Science and Engineering)
August, 2004

Geographic information systems deal with the exploration, analysis, and presentation of geo-referenced data. Virtual reality is a type of human-computer interface that comes close to the way people perceive information in the real world. Thus, virtual reality environments become the natural paradigm for extending and enhancing the presentational and exploratory capability of GIS applications in both the spatial and temporal domains. The main motivation of this thesis is the lack of a framework that properly supports the exploration of geographic information in a multi-dimensional and multi-sensorial environment (i.e., temporal virtual reality geographic information systems).

This thesis introduces a model for *virtual exploration of animations*. *Virtual exploration of animations* is a framework composed of abstract data types and a user interface that allow non-expert users to control, manipulate, analyze, and present objects' behaviors in a virtual-reality environment.

In the model for *virtual exploration of animations*, the manipulation of the dynamic environment is accomplished through a set of operations performed over abstractions that represent temporal characteristics of actions. An important feature of the model is that the temporal information is treated as first-class entities and not as a mere attribute of action's representations. Therefore, entities of the temporal model have their own built-in functionality and are able to represent complex temporal structures.

In an environment designed for the manipulation of the temporal characteristics of actions, the knowledge of relationships among objects' behaviors plays a significant role in the model. This information comes from the knowledge base of the application domain and is represented in the model through constraints among entities of the temporal model. Such constraints vary from simply relating the end points of two intervals to a complex mechanism that takes into account all relations between sequences of intervals of cyclic behaviors.

The fact that the exploration of the information takes place in a virtual reality environment imposes new requirements on the animation model. This thesis introduces a new classification of objects in a VR environment and describes the associated semantics of each element in the taxonomy. These semantics are used to direct the way an object interacts with an observer and with other objects in the environment.

DEDICATION

To Teca, Lucas, and Rodrigo.

ACKNOWLEDGEMENTS

This thesis was completed with the contribution of many people to whom I would like to express my gratitude.

I would like to acknowledge the members of my advisory committee Max Egenhofer, Kate Beard, Kathleen Hornsby, Marcelo Gattass, and Michael Worboys. I would like to thank specially my advisor Max Egenhofer for believing in this project.

I would like to thank the staff and faculty members at the Department of Spatial Information Science and Engineering and specially Alfred Leick, Harlan Onsrud, Peggy Agouris, and Karen Kidder.

I would like to acknowledge the support of the staff members at the Fogler Library and the Graduate School.

I would like to thank all my friends at the Department of Spatial Information Science and Engineering and specially Jim Farrugia, Fred Fonseca, and Mike Hendricks.

I would like to thank the financial support of the Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq, under grant number 200020/00-5.

Finally, I would to say thank to my family both in Brazil and in Maine and specially Teca, Lucas, and Rodrigo, to whom I dedicate this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Virtual Reality (VR) is a type of human-computer interface (Brodlie *et al.* 2002) that comes close to the way people perceive information in the real world (Jacobson 1991). Because VR communicates information to a user by exploiting every sensory channel, VR promises to reduce the impedance between the representation of information and people's mental conceptualizations of space and time (Raper 2000). It gives a user the impression of being part of a synthetic environment and the ability to interact with this environment in a more natural way. These characteristics are particularly useful for exploring natural environments.

Geographic information systems (GISs) deal primarily with the exploration, analysis, and presentation of geo-referenced data. Traditional user interfaces, which are static and present only a two-dimensional view of the data, have proven to be difficult to use in situations that require the analysis of an increasing amount of three-dimensional data changing over time (Verbree *et al.* 1999). In this sense, VR becomes the natural paradigm for extending and enhancing the presentational and exploratory capability of GIS applications in both spatial and temporal domains. In the spatial domain, for instance, the use of immersive VR environments in applications of the oil and gas industry has suggested that VR interfaces can help geoscientists to analyze in more detail geophysical and geological data (Lin *et al.* 2000). In a field where the process of gathering data is always very expensive, the advantage of using VR becomes a strategic issue. In the temporal domain, the visualization of information through animations has been recognized as a natural means to explore and analyze time-varying information and processes (DiBiase *et al.* 1992). Animation is a powerful mechanism that enhances the understanding of the data under investigation and fosters new insight into the underlying processes (Brodlie *et al.* 1992).

The use of animations as an exploratory tool and the advances of computational resources raise many challenges for the GIS community. On the one hand, the complexity of time and the diversity of geographic phenomena and their behaviors represent barriers for the conceptualization of data models that better capture the richness of temporal geographic information. On the other hand, the development of more cognitive computational environments, which explore a user's sensory channels and the sense of immersion in the communication process, demands the development of new metaphors and methods for exploration of the information.

This thesis focuses on the exploration of geographic spaces in VR settings. The main motivation for this thesis is the current lack of a theory that properly supports exploratory analysis of spatio-temporal data sets using VR technology. This research presents a data model that supports manipulation, analysis, and presentation of dynamic geographic objects in VR environments, giving attention to the representation of interactions between the user and the data set in the spatial and temporal domains.

1.1 Virtual Reality Geographic Information Systems

GIS can be roughly defined as a combination of database management systems, a set of operations for examining data, and a graphic display for spatial analysis (Rhyne 1997). For presentation and analysis purposes, GIS applications have been relying on two-dimensional displays. The increasing amount of multi-dimensional and geo-referenced data, however, demands a more cognitive computational environment (i.e., hardware and software) to handle this kind of information (Kraak *et al.* 1999). In this sense, VR is a natural candidate to fill such a position.

The combination of VR with GIS becomes known in the literature as VRGIS (Faust 1995). In a first attempt to combine VR with GIS, some GIS applications simply provide VR as an alternative interface for the presentation of three-dimensional geographic information. The second generation of VRGIS applications increased the level of integration by incorporating some GIS functionality into the VR interface. These applications have been reported in many areas such as urban planning (Verbree *et al.* 1999; Zlatanova 2000), environment and ecology (Raper 2000), data visualization (Kraak *et al.* 1999), terrain visualization (Koller *et al.* 1995; Neves *et al.* 1999; Reddy *et al.* 1999), animations (Dollner and Hinrichs 1997; Luttermann and Grauer 1999; Hardisty *et al.*), archaeology (Ogleby 2002), military recognition and training (Macedonia 2002), simulation (Wenzel and Jensen 2001), navigation, orientation, and usability issues (Fuhrmann and MacEachren 1999; Kraak *et al.* 1999; Verbree *et al.* 1999; Chitaro and Scagnetto 2001), and education (Dykes *et al.* 1999).

The increasing number of VRGIS applications and the acknowledgment of the potential of this integration encourage the search for a more formal definition for the term VRGIS. Williams (1999) defined VRGIS as "a multi-dimensional, computer-based environment for the storage, modeling, analysis, interrogation of, and interaction with, geo-referenced spatio-temporal data and processes." This definition depicts a strong connection between the general goals of VR and GIS applications. From the VR perspective, the VR definition (Kalawsky 1993) is extended to incorporate geo-referenced data and to highlight the exploratory nature of GIS applications. On the GIS side, the traditional GIS's definition (USGS 2003) is simply complemented with the term multi-dimensional.

The integration of VR and GIS, however, is more than a simple combination of two different disciplines. This integration has the potential of enhancing the functionality of both fields in a way that surpasses the sum of each field separately. When VR and GIS are combined in a single application, new possibilities and opportunities arise, helping to solve old problems and address new questions not attempted yet. These features are enhanced when the temporal dimension is added to the VRGIS framework, producing what became known as *temporal VRGIS* (Williams 1999). In this way, *Temporal VRGIS* represents a new tool in the GIS arsenal to be used wherever the cartographic view of the world is no longer adequate or sufficient to support spatio-temporal reasoning.

1.2 Temporal VRGIS

Temporal VRGIS is the symbiotic convergence of the advances in three salient features of a GIS application: *presentation*, *spatial interaction*, and *temporal interaction* (Figure

1.1). *Presentation* deals with formats and devices used to display the information (e.g., a digital image in a computer screen). *Spatial interaction* is concerned with the interplay between the observer and the data (e.g., changing the scale of an image or selecting an object for deletion). *Temporal interaction* refers to the control and presentation of the dynamic information (e.g., playing an animation).

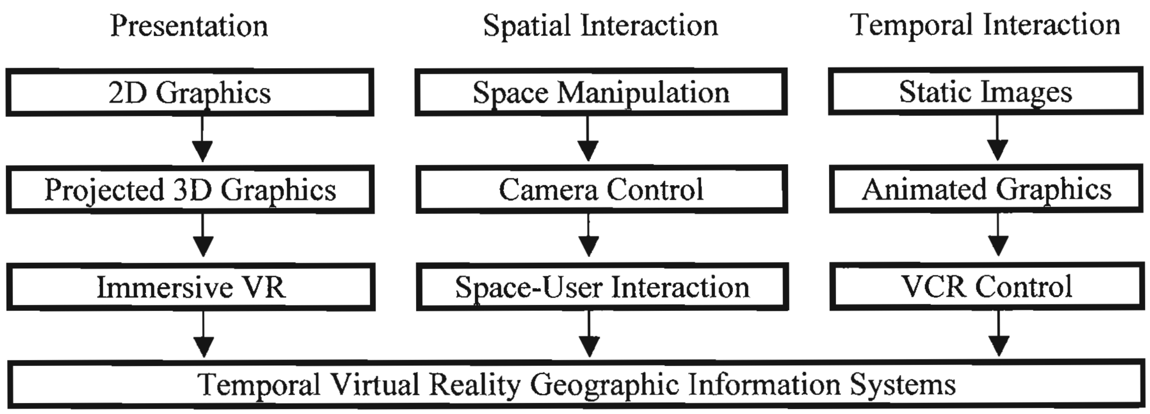


Figure 1.1. The convergence of presentation media and devices, spatial interaction and temporal interaction yields the model of Temporal VRGIS.

1.2.1 Presentation

The presentation of information can be accomplished in a wide combination of formats and devices. The most rudimentary set is a digital image or two-dimensional graphic rendered on a flat computer screen. This configuration has been used for decades and represents the traditional cartographic view of the world in a digital environment.

The second stage of presentation of spatial information is the display of three-dimensional information in a two-dimensional media. This transformation is accomplished through a mapping from the three-dimensional to the two-dimensional space (e.g., a perspective projection). This mechanism produces more realistic views and allows an observer to perceive volume and distinguish objects located at different distance. The major drawback of perspective projections is that geometric properties are not preserved under such a mapping. In this way, the user cannot extract any metric information from the display (e.g., distance and area). Since topological properties are preserved, it is still possible to extract qualitative information about the environment. This approach is used in traditional 3D graphics and non-immersive VR applications.

A more sophisticated form of presentation of three-dimensional graphics is immersive VR. In this environment, the computer generates two slightly different views of the three-dimensional space and a special device (e.g., a pair of goggles) coordinates the presentation of these images for the observer. This stereo view produces a strong depth cue and allows an observer to "see" in three-dimensions (Foley *et al.* 1997). If in addition, an observer is surrounded by stereographic displays, it gives an observer the illusion of full immersion in the environment. The sense of immersion experienced by a user exploring such environments constitutes a communication medium that cannot be simulated by any other computational resource. In this sense, VR can be seen as the leading edge of computer-generated environments that starts with a simple digital version of an image (Haklay 2002).

Despite the differences in the level of sophistication and the computational effort involved in each visualization technique, there is no better form to present the information. Each technique has its own characteristics and serves different purposes. Verbree *et al.* (1999) suggest the combination of different visualization techniques in a single application. The application provides different views of the environment (i.e., 2D, projected 3D, and VR) that may be used either simultaneously or intermittently. In this approach, the decision of the appropriate form of visualization is left for the observer. The observer decides which view is more adequate based on the task at hand and the functionality available for each view. Moreover, the observer can select a view that he or she feels is more comfortable for performing the desired task. Some users are not trained to read two-dimensional maps, while others get lost in navigating VR environments. The combination of different views can help the observer to understand more symbolic representations and enhance the orientation in large environments.

1.2.2 Spatial Interaction

Spatial interaction is usually discussed under the perspective of an active observer changing his or her view of the environment, manipulating an objects' appearance or spatial location, and performing some kind of request about geometric characteristics of the image. In this sense, a zoom or pan of a map, the selection of an object for deletion, and a request about the distance between two points are all types of spatial interaction.

The increasing support of the media for richer forms of representation and the immersive characteristic of the observer in the data extends the number of possibilities of spatial interactions. The presentation of three-dimensional graphics together with the ability to interactively and continuously define the vantage point of the observer allow for a new type of spatial interaction. The observer, immersed in the environment, can now perform a walk-through and analyze the data from different perspectives.

In VR environments the observer becomes increasingly part of the data set (Kraak 2002). The ability of existing computer devices to output sensory feedback makes interactions an even more complex topic. Interaction in VR can include constraints imposed by the data on an observer's actions (e.g., a building that blocks the path of the observer or a heavy object that cannot be moved). This type of spatial interaction needs further investigation. In the GIS context, it is important to identify the type of constraints imposed by geographic objects on the objects themselves and on the observer acting in VR environments. It is also important to determine which kinds of constraints need to be represented and to evaluate the cognitive impact of imposing such constraints on a user who is exploring the environment.

1.2.3 Temporal Interaction

Temporal interaction is related with the exploration and presentation of information that changes over time. The portrayal of static images is the paradigm of applications that offer only a limited support for the presentation of the time-varying information. In these applications the user is constrained to explore snapshots of different configurations of the information in which each snapshot represents the state of the environment at a certain instant in time. The user can alternate the display and analyze a single state of the environment at a time. If the application supports the presentation of a coherent sequence of images in an automatic fashion and in a way that gives the observer the illusion of movement, the application provides animations.

The most rudimentary animation mechanism is animated graphics. This mechanism presents a coherent sequence of images giving a user the illusion of movement. Animated graphics, however, does not give the user the ability to control the flow of information. Some VRGIS applications that support animations use animated graphics as the only mechanism to present time-varying information. Thus, the level of interactions with the temporal domain is limited to a passive and contemplative observation of the flow of dynamic objects in the environment. In these settings there is no way to pause, rewind, or fast-forward the animation. It is equivalent to the mechanism of *animated gifs* used with raster images in two-dimensional displays. By giving a user the ability to control the flow the animation constitute a stepforward towards a “truly” temporal interaction mechanism.

The set of operations used to control the animation defines the level of interaction that can be accomplished in the temporal domain. In the majority of applications that support animations, users control the presentation of the animation using a small set of operations that resembles operations of a video-cassette recorder (VCR) control (e.g., play, stop, rewind, and fast forward). The use of the VCR metaphor to control animations has pros and cons. Its main advantages are the reduced number of operations and the fact that almost everybody is very familiar with the functionality of such a set of operations. The main drawbacks of this set of operations are the limited number of different views of the dynamic environment that can be created and the fact that the user is constrained to control the animation as a whole, without any means to address the behavior of an individual or a set of individuals. With a VCR-like metaphor, for instance, it is impossible to observe the behavior of an object been performed in a chronological order, while other objects are observed performing their activity in a reverse chronological order. A user controlling the animation as a whole cannot generate this interesting and possibly insightful view of the dynamic environment.

1.3 Motivation

The basic requirements of a temporal VRGIS application include the support for the representation of time, behavior evolution, and processes (Williams 1999). Moreover, this kind of application needs to support operations that allow a user to observe, distort, and modify the temporal dimension, to gain insights and discover relationships among geographic phenomena. Existent temporal VRGIS applications do not accomplish these requirements.

While the integration of VR and GIS in the spatial domain is growing fast, this integration in the temporal domain is still incipient. The multi-dimensionality of VRGIS applications cannot be reduced to the representation, perception, and analysis of the three-dimensional space. The temporal dimension and other semantics dimensions of the geographic space require the same support and investigation effort as experienced by that of the spatial dimensions.

This thesis pursues a model for *virtual exploration of animations*. The model of *virtual exploration of animations* covers specific aspects of the more general model that supports Temporal VRGIS applications (Figure 1.2). *Virtual exploration of animations* is a framework composed of abstract data types and a user interface that allow non-expert users to control, manipulate, analyze, and present objects' behaviors. The term *exploration* suggests that the interface and methods used to control the animation move beyond the mere presentation of the animation. The exploratory nature of an interface for *virtual exploration of animations* must afford users the ability to create their own view of the dynamic environment. Thus, the user can learn from the act of creating and not just by the result of the creation. The term *virtual* suggests that the exploration of the animation takes place in a VR environment. Thus, objects in the environment interact with other objects and the observer and this interaction may modify the progression of the animation.

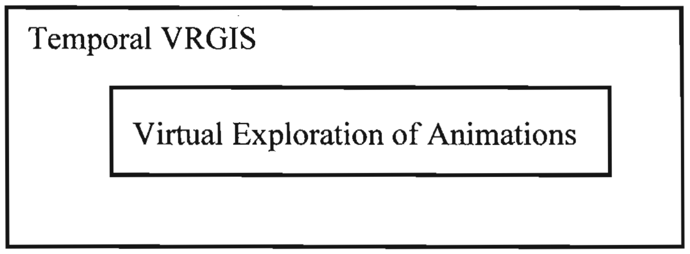


Figure 1.2. A model for virtual exploration of animations in the context of a model that supports Temporal VRGIS applications.

The model for *virtual exploration of animations* addresses a major research topic for the scientific visualization community that has not been given the deserved attention in the context of temporal VRGIS applications. Animation is an important mechanism for the visualization and analysis of dynamic phenomena; therefore, it is helpful in the cognitive process of exploration of the environment.

Computer animations have been used for decades to present time-varying phenomena or information (Tobler 1970). The low-level of abstraction of existent animation models makes it difficult to isolate pieces of the animation (e.g., the behavior of a single object) and, therefore, limits the user's ability to produce a view of the dynamic environment that is appropriate for the task at hand. An animation model that supports entities representing different granularities of the dynamic environment and presentation operations that have such entities as argument provides the foundation for finer control over the animation. This approach is equivalent to extending the traditional set of operations used to control the animation of each object's behavior in the environment (i.e., to have a VCR control attached to each dynamic object). Such a hypothetical model pushes the VCR metaphor to its limits. This model is called *individual VCR*.

In an animation model that supports presentation operations over individual objects' behavior (i.e., the *individual VCR* model) the number of different views that can be accomplished by a user grows exponentially. In the context of GIS, however, this extended version of the animation model is still not expressive enough to support *virtual exploration of animations*. First, the representation of individual object's behaviors is not necessarily the best abstraction for a user work with. Sometimes the user is more interested in analyzing and manipulating a small piece of the behavior of an object, the behaviors of a group of objects, or the behaviors of all objects in the environment. Second, the VCR-like style of operations aims at the presentation of the animation. These operations, even when performed over individual objects' behavior, constitute a limited mechanism to compose different views of the environment. These operations, for example, do not support any kind of selection, filtering or grouping mechanism over the set of dynamic objects. In most GIS applications the number of dynamics objects in the environment can easily become unmanageable for a person. Third, the conceptualization of the temporal domain in current animation models does not properly capture the complexity of time in the GIS field. In such models, the representation of temporal information is usually accomplished by an absolute and quantitative representation of time. This simple temporal representation does not explore qualitative representation of time and the temporal relation between objects' activities and does not support the representation of more elaborate temporal structures (e.g., the temporal structure associated with objects that have a cyclic behavior). Fourth, in existent animation models, objects' behaviors are not related through temporal constraints. This information is important to prevent unrealistic views of the environment due to the manipulation of the animation (e.g., a cause that happens after its effect). Fifth, in existent animation models for VR environments, the semantics of the objects are not represented. These semantics are almost always assumed by the observer and derived from the context of the application. The semantics of the object, however, play a significant role in directing interaction between the object and the observer, as well as in directing the interaction among the objects themselves.

1.4 Goal and Hypothesis

The main goal of this thesis is to develop an animation model to support exploratory analysis of dynamic VR environments. In order to accomplish such a goal we need an animation framework in which (1) the level of abstraction of the model's entities is rich enough to allow a non-expert user to perform qualitative spatio-temporal reasoning about the patterns of VR object's behaviors at different levels of granularities, (2) the set of operations are small, intuitive, and sufficient to give a user effective control over the presentation, combination, and manipulation of objects' behavior, (3) the temporal model is rich enough to support the representation of complex geographic phenomena over time, (4) temporal constraints among objects' behaviors are captured, and (5) the semantics of each object in the environment is explicitly represented. These requirements lead to major research questions that direct this thesis:

1. What are the levels of abstraction needed to represent objects' behaviors?
2. What are the operations over objects' behaviors needed to combine and manipulate different configurations of the dynamic environment?
3. What are the temporal structures needed to capture the richness of geographic phenomena?
4. What types of temporal constraints are needed to capture known dependencies among geographic phenomena?
5. What are the semantic characteristics of an object in VR environments?
6. How do objects' semantics interact with the behaviors of others objects and with an observer immersed in a VR environment?

The answer of these questions yields a model for *virtual exploration of animations*, which is used to support the hypothesis of this thesis:

The model of virtual exploration of animations produces views of an animation that cannot be accomplished by any combination of operations of the individual VCR model.

In the context of this thesis, we consider a view as an animation in which an observer perceives the progression of objects' behavior. Thus, an animation with two objects performing their behavior in the normal temporal order and an animation in which only one object performs its behavior in the reverse temporal order are different views of the animation. The number of different views of the animation is directly proportional to the number of dynamic objects in the environment. Since the user cannot create new objects or behaviors, but only modify the temporal configurations of existent behaviors, the number of possible views of the animation is finite. In this thesis we assume that all views are equally important and have the same weight in the exploration of the animation.

The method used to support the hypothesis of this thesis is the comparison of the number of views of an animation produced by two different animation models. Animations produced with the first model extend the traditional VCR control to every dynamic object in the environment (i.e., the user produces animations through a VCR-like control that affects each object individually). The second model produces animations based on the abstractions and operations of the model for *virtual exploration of animation.*

1.5 Research Approach

The scientific-visualization community has proposed many different data models to represent time variation of information through animations. Among the great number of animation models, keyframe animation is still the dominant paradigm (Chandru *et al.* 2000). In this kind of animation, representative scenes (called key frames), together with the instant at which a scene should be presented, are explicitly stored. The computer uses interpolation functions to automatically generate intermediate frames of the animation. The low level of abstraction of existing implementations of keyframe animations limits the manipulation of animations and the scalability of the model. The conceptualization of an animation model with different levels of abstraction and a reasonable set of operations can easily overcome this deficiency. Users working with different levels of abstractions of objects' behaviors can construct and manage efficient, meaningful, and huge animations.

This thesis proposes an animation model based on the keyframe paradigm. The conceptualization of the model conceives a framework composed by four main logical parts (i.e., geometric, action, temporal, and semantics).

The geometric model describes the geometry and appearance of the objects (i.e., the static content of the animation). There exist many data models to represent the static content of an animation. Some of these data models are standards and have a wide acceptance while others are very specific and used for a certain task. The main characteristic of such data models is that they are relatively decoupled from the dynamic content of the animation and, therefore, they can be used interchangeably with minor efforts.

The action model encompasses different granularities of the objects' behaviors. Such granularity is modeled through a hierarchical structure that represents increasing abstractions of the animated environment. The animation is built-up constructively from the lowest to highest level of abstraction in the model. Entities at a low-level of abstraction represent the building blocks of an object's activity. These entities model different pieces of the object behavior. At a mid-level of abstraction, entities representing a collection of small pieces an object's activity model the entire behavior of an object. At a high-level of abstraction, individual behaviors are combined forming groups of objects' behaviors. Entities of this level of abstraction represent the behaviors of semantically related groups of objects. The action model encapsulates all information needed to generate the time variation information of an object's attributes, except time itself.

Different from many animation models, in which time is treated as a simple attribute, entities of the action model do not carry temporal information. In this thesis time is an abstract dimension that deserves its own model.

The temporal part of the framework is modeled by data abstractions that represent a hierarchical structure of time. Entities of the dynamic and temporal part of the framework are strongly related (i.e., the structure of the temporal model follows the structure of the action model). This tight connection between the dynamic and temporal parts of the framework enforces the user's understandings and facilitates the implementation of the model. Moreover, this coupled representation allows the depiction of an object behavior as time interval, representing the period when the object is performing its associate activity. The set of operations used to combine and manipulate objects' behaviors having temporal intervals as arguments are effective abstractions even for naive users and have the convenience of being easily represented with a simple graphical user interface (Schmitz 2002). This abstract representation also has the advantage of hiding information that cannot be manipulated by a user, such as type and values of objects' attributes, interpolations functions, and so on.

The semantic part of the framework models the evolution of VR objects' semantics over time. In the context of GIS, the interaction with virtual geographic objects is a critical issue. The immersive nature of the observer in the data requires that the objects embody semantic information that directs its response in a VR environment. This thesis identifies individual semantic characteristics of VR objects and proposes a classification of such objects based on an exhaustive combination of their semantic characteristics. Objects with different combinations of characteristics carry distinct semantics in the model (i.e., the object interact with the observer or with other object in the environment in a different way).

This thesis presents a prototype implementation for the model of *virtual exploration of animations*. This prototype is implemented using the Java programming language. A possible graphical user interface for the model is also introduced. This interface carries all functionalities of the model that employed in the production of new views of the animation. Thus, it is used to compute the expressive power of the model of *virtual exploration of animations*.

1.6 Scope of the Thesis

This thesis is concerned with the conceptualization of an animation model tailored for users of multidimensional GISs. The major concern of this work is the conceptualization of an animation model with high-level abstractions and operations that allow a user to manipulate objects' behavior in VR environments, getting insights and discovering relationships among dynamic phenomena.

This thesis does not treat the production of the animation. The user can manipulate pre-orchestrated behaviors of objects by selecting pieces of the movement, changing the temporal order or speed, or modifying the instant when an object performs its activity. We assume that the animation is generated in an automatic fashion by a simulation application, by an application that translates the information stored in a spatial-temporal database, or by a human using any authoring tool that supports the proposed model.

This work is not about behavioral animations or those animations found in interactive computer games. The animation framework requires that the entire behavior of an object be known ahead of time. In this way, this model can act as a post-processor of a simulation application or as a front-end of a spatio-temporal database.

The thesis does not treat film-based animation. We do not work with the presentation of a sequence of snapshots of the environment or raster representation of the information. Entities of the animation framework represent the three-dimensional geometry and appearance of the data set and the behavior of dynamic objects.

1.7 Major Results

The major result of this thesis is a framework that can be used for exploratory analysis of multidimensional environment. The framework has the following advantages over existent animation models:

- The representation of objects' behavior at different levels of granularity allows a non-expert user to control elements of a single object's behavior, the entire behavior an object, the behavior of groups of object, or all objects in the environment.
- The separation of spatial and temporal characteristics of objects' behaviors and a set of operations over temporal representations give a user an intuitive framework to manipulate the animation, creating new views of the dynamic environment.
- The representation of more elaborated temporal structures captures complex geographic phenomena and support temporal reasoning over cyclic behaviors.
- The representation of temporal constraints among objects' behaviors. These constraints are used to preserve some known relationships among objects behavior during the manipulation of the animation by a user.

- The representation of the evolution of the VR objects' semantics over time. The semantics associated with VR objects provide valuable information in the process of the virtual exploration of an environment and play a significant role in interactions with observers and the data.

1.8 Intended Audience

The intended audience of this thesis is any person interested in the conceptualization, representation, and visualization of three-dimensional dynamic objects. This thesis addresses especially users, analysts, designers, and implementers of systems that support three-dimensional graphics and animations, as well as researchers from the fields of geographic information science, visualization, computer animation, virtual reality, and software engineering.

1.9 Organization of Remaining Chapters

The remainder of the thesis is organized into seven chapters structured as follows.

Chapter two reviews related work of multi-dimensional geographic information systems. This review includes topics related with geographic visualization and animation models. These topics are analyzed in the context of an exploratory tool of geographic phenomena and processes.

Chapter three introduces the action part of a framework for virtual exploration of animations. In this chapter we introduce data abstractions and relevant operations associated with the spatial characteristics of objects' behaviors.

Chapter four presents a conceptualization of the temporal part of a framework. Different conceptualizations of time are discussed. A structure of classes representing the temporal characteristics of actions was presented and attributes and methods of these classes were discussed in an informal way.

Chapter five discusses the temporal constraints mechanism. This mechanism is used to represent in the animation model some known relationships among the temporal characteristics of entities of the real world.

Chapter six introduces compositions operations. Compositions operations allow an observer to modify the temporal configuration of the animation and create different views of the dynamic environment.

Chapter seven presents the semantic model. This chapter introduces a new classification of VR Objects and discusses the semantics associated with each class of object in the taxonomy. This chapter presents also a model to represent the evolution of the objects' semantics over time and outlines the rationale used to modify such semantics during the manipulation of the object behavior.

Chapter eight presents a user interface of a prototype implementation of the model and provides an evaluation of the expressive power of the model.

Chapter nine draws conclusions and indicates further work.

CHAPTER 2

MULTI-DIMENSIONAL GIS

This chapter reviews relevant research topics and literature concerned with the representation of dynamic geo-referenced information in multi-dimensional environments. The next section discusses the role of visualization of geographic information and the use of new visualization devices in GIS applications. Subsequently, we discuss some important data models used to represent three-dimensional and dynamic information. Finally, we highlight the major strengths and weakness of current animation models in supporting exploratory analysis of dynamic information in a VR setting.

2.1 Geographic Visualization

Visualization has been defined as the mapping of data to representations that can be perceived by a person using any sensorial channel (Foley *et al.* 1997). Geographical Visualization (GVis) focuses on representations of geographic spatio-temporal data and their applications in supporting all tasks of geographic analysis (Buckley *et al.* 2000; Gahegan 2001; MacEachren and Kraak 2001).

Two important research topics for the GVis community and of special interest for this thesis are related to the use of animations as a tool for exploratory analysis (Fairbain *et al.* 2001) and the incorporation of technological advances in GIS applications (MacEachren and Kraak 2001).

The first topic investigates the role and effectiveness of animations in communicating geographic phenomena and processes. The majority of GIS applications use animations as a simple mechanism to depict time-varying information. In these applications the observer becomes a mere spectator of previous modeled behaviors. The exploratory nature of GIS, however, requires an environment where the observer has full control over the content and flow of the animation (Fairbain *et al.* 2001; Kraak 2002).

The second topic looks into technological advances in hardware and software and their cognitive significance for GIS applications. While two-dimensional displays are still the dominant paradigm among GVis applications, the next generation of GISs will certainly benefit from the increasing capability of hardware and software to support and output more sophisticated data representations. In the visual domain, for instance, the presentation of stereoscopic three-dimensional scenes explores the sense of immersion of the user into the data to better communicate information derived or not from the physical world (Jacobson 1991; Brodlie *et al.* 2002). In the non-visual domain, the use of sound (Krygier 1994) and haptic feedback (Neves *et al.* 1997) promise to enhance the cognitive gain of the observer in exploring and analyzing spatial information (Raper 2000).

Although these topics have been an active research area among the GVis community, the combination of animations as an exploratory tool and VR environments is still lacking. The synergetic integration of these disciplines launches many research questions that can greatly benefit each other and are beyond the scope of each topic individually.

2.2 Computer Animation

Animation is the process of creating, storing, and presenting a sequence of different images that gives the observer the illusion of motion (Thalmann and Thalmann 1985). The term computer animation refers to a technique in which the computer is used in at least one phase of the animation process. Different computer techniques are used in each phase of the animation. The creation of the animation, for instance, ranges from a person using the computer to produce every image of the animation to the production of the animation by a computer in an automatic fashion. The storage of the animation varies from recording a linear sequence of every image of the animation to storing a description of the evolution of three-dimensional objects. The presentation of the animation can be accomplished by any visual display, ranging from the small screen of a personal device to the set of large displays of a CAVE system (Dam *et al.* 2000). To simulate motion, the computer presents a new, slightly different image many times per second (Foley *et al.* 1997).

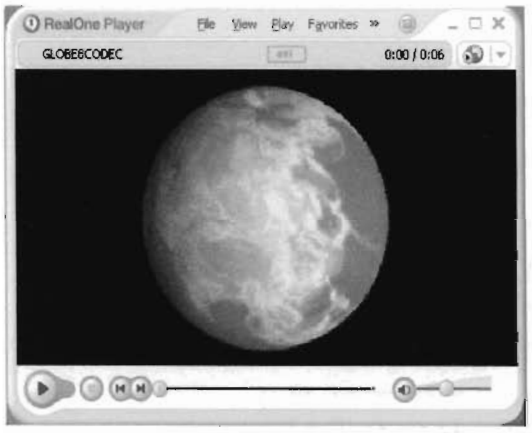
One salient characteristic of computer animations is related with the data model used to store and process the imagery. Based on this characteristic, the animation can be classified as image-based or content-based (Lee 1998a).

2.2.1 Image-Based Animations

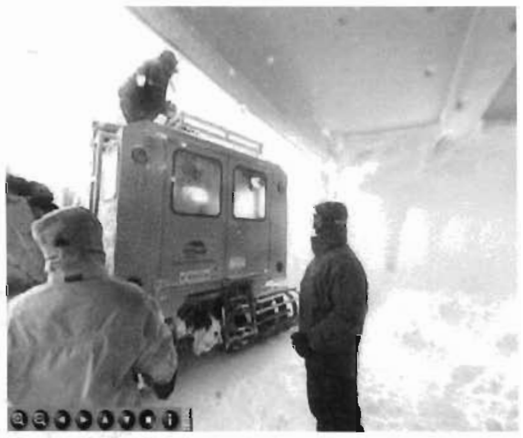
Image-based data models store a sequence of images (Gall 1991; Miller 1993; Chen 1995). Each image is called a frame or a scene of the animation. In these models the computer stores matrices of picture elements (pixels) representing frames of the animation. The position of each element in the matrix corresponds to a spatial position in the image and the value of the element represents the pixel's brightness (Gonzalez and Woods 1992).

Image-based data models can be used to store both recorded digital videos and computer-generated images. In the former case, the level of realism of the animation can only be compared to the one achieved by the human visual system (Raper 2000). In the latter case, the level of realism may vary, and only in few cases can be compared to those of digital videos (Cosatto and Graf 2000). Computer-generated images, however, have a significant advantage over recorded videos. The computer can generate images of objects that do not exist in the physical world (i.e., the object no longer exists, will be created, or represents the graphical realization of a concept). The computer also can generate images of objects that cannot be captured by a digital camera (e.g., objects of the size of an atom or located at an astronomic distance).

The elementary data structure of image-based animations imposes some limitations on interactions between the user and the information depicted in the screen. In this model, interactions are usually restricted to the presentation of the animation. At the presentation phase, an application (i.e., a player) reads the information stored in a file and renders the correct sequence of images on a computer screen. The functionality of each player defines the level and type of interactions that can be accomplished with the animation. This functionality depends on the organization of the frames in the model, which can be ordered in a linear (Miller 1993) or non-linear (Chen 1995) fashion. A player for linear models (Figure 2.1a), for instance, allows the observer to set some playback parameters and watches the flow of animation with the selected configuration. With this type of player the observer, using a VCR metaphor to interact with the information, can play or stop the animation, watch the animation at different speeds, and examine, step by step, each frame of the animation. A player for non-linear models (Figure 2.1b) allows the observer to change his or her vantage point, simulating a “walk through” or “looking around” the environment. Controls associated with these players allow the observer to perform a complete turn of the camera along the principal directions (i.e., horizontal and vertical), zoom in, zoom out, or jump to a “hot spot” (i.e., a recorded configuration of the camera highlighting some feature of the image). These players are used mainly to present panoramic images (i.e., they support only still objects). Thus, the illusion of movement comes from the movement of the camera in a static environment.



a)



b)

Figure 2.1. Different types of image-based animations: a) A linear player with controls to manipulate the flow of the animation and b) a non-linear player with controls that directs the movement and position of the camera.

2.2.2 Content-Based Animations

Content-based data models store a description of a scene, behaviors of animated objects, and the position and direction of a camera. The computer uses this information to generate and render each image of the animation. The manipulation of any piece of information stored in the model produces a new animation. In content-based data models the level of interaction between the observer and the information is potentially high. The observer, for instance, can continuously change the position of the camera while other objects are performing their associated behaviors. This configuration corresponds to the combinations of the functionalities of linear and non-linear image-based animations. The observer can also hide a group of objects or direct the computer to generate only the movement of certain objects. This configuration cannot be accomplished in image-based animations where the content of the animation is fixed.

There are many different content-based data models. Some of them have been designed to accomplish a specific task and are used in a narrow context. Others are generic and experience a wide acceptance. Unlike image-based models (Gall 1991), there is no standard among them yet. Content-based data models differ mainly in the strategy used to structure and process the information of each scene. Based on the methods used to control the motion, the animation model can be classified as geometric, physical, or behavioral (Thalmann and Thalmann 1994).

- Geometric models (Zeleznik *et al.* 1991; Koved and Wooten 1993; Strauss 1993; Elliott *et al.* 1994; Najork and Brown 1995; Green and Halliday 1996; Dollner and Hinrichs 1997; Lee 1998b) are based on the description of the evolution of the objects' geometry and appearance (e.g., position, shape, or texture). These models "know" relevant states of the objects in the scene (e.g., the initial and final positions of the object) and, through some internal mechanisms, they compute intermediate states of the movement. These mechanisms are based solely on the knowledge of the evolution of objects' behavior and do not consider any force that drives the movement. Geometric models sacrifice the realism of the scene for the sake of a fast computation of the animation. Thus, these models are suitable for real-time animations with pre-orchestrated objects' behavior (i.e., behaviors completely known before the presentation of the animation).

- Physical models (Pentland and Williams 1989; Cohen 1992) compute the evolution of an object behavior based on the physical laws that govern the movement or deformation of the object. These models produce very realistic results and rely on the support of external applications to solve differential equations or non-linear systems. Physical models have applications in engineering, medicine, and realistic animations of characters for the cinematography industry.
- Behavioral models (Reynolds 1987; Funge 2000) emphasize the information and rationale used to represent objects' behaviors. In this model objects are capable of adapting to social and physical constraints. These models are strongly based on concepts from artificial intelligence and behavioral science and have the game industry as their major target audience.

Geometric data models are usually the most suitable representation for GIS applications. In a typical GIS application the behaviors of the objects are known ahead of time. They are customarily stored in spatio-temporal databases (Erwig *et al.* 1998; Chomicki *et al.* 1999; Forlizzi *et al.* 2000) or outputted from a simulation application (Lieberman 1991; Wenzel and Jensen 2001).

A number of GIS application use VRML (Web3D 2003) as the mechanism to encode and present three-dimensional and sometimes animated graphics. VRML is an interchange format for the representation and presentation of animated three-dimensional graphics. The data model behind a VRML file (Strauss and Carey 1992) was designed aiming at an efficient and interactive presentation of the information in VR environments. VRML, however, has critical limitations in interacting with temporal information. VRML does not have any native support for controlling the animation or changing the temporal configuration of the objects' behavior.

The presentation of animation with commercial VRML players does not allow a user to manipulate the content of the animation. Thus, users are constrained to seeing animations in which the number of animated objects in a scene can easily exceed the users' capacity to understand the dynamic environment. In order to overcome this deficiency some applications are extending the VRML syntax and exploring the potential offered by the manipulation of the temporal domain (Dollner and Hinrichs 1997; Luttermann and Grauer 1999; Reddy *et al.* 2000; Hardisty *et al.*). These applications extend the set of operations to control the animation with operations that allow the user to perform a selective view of the animation (Manoharan *et al.* 2002). This selection, however, is solely based on the users' judgments and does not consider temporal constraints among the objects. In this way, some important behaviors can be inadvertently hidden so that they are not taken into account in the exploration of the animation. In such a model, users should have the opportunity to explore the rich set of relations that exists among objects.

Although the specification of VRML contemplates a mechanism to extend the language, this mechanism is insufficient to accommodate all requirements of temporal VRGIS applications. VRML lacks the extensible power of an object-oriented data model.

Thus, VRGIS applications based on VRML are restricted to offer only a few basic GIS functionalities. The implementation of a "true" temporal VRGIS application requires the conceptualization of a data model that incorporates the GIS functionality in both spatial and temporal domains.

2.2.3 Animations for GIS

Animations in GIS serve different purposes. The task involved and the target audience are critical for determining the appropriate type of animation. In an explanatory approach, for instance, the animation is used to communicate the information for a broad audience with distinct backgrounds and different degrees of expertise. In this context, image-based animations, with realistic images and embedded in multimedia documents, have been used successfully (Cartwright 1999; Peterson 1999). The use of realistic and animated representations of geographic phenomena avoids considerations about previous knowledge or training of the observer. Moreover, the use of a VCR metaphor to control the presentation of the animations provides an effective user interface with a limited but universal set of operations.

The simple use of realistic animations, however, is insufficient for supporting an exploratory analysis of time-varying information (Andrienko *et al.* 2000). In an exploratory approach, the observer needs a more interactive environment that gives the observer complete control over the flow and the content of the animation (Kraak 2002). These requirements need investigations on data models that better represent multi-dimensional geographic information and a set of operations that control the flow and the content of the animation. In this sense, geometric content-based animations with a coherent representation of geographic phenomena and high-level abstractions representing the dynamic environment can be used to move the user interface beyond the VCR metaphor and support an exploratory analysis of animations.

2.3 Data Models for Temporal VRGIS

The representation of information is a fundamental issue for the integration of VR and GIS applications. The major problem with current temporal VRGIS applications is related with the data model used to support the VR interface. An appropriate data model of the conceptualized world is the foundation of an efficient storage, management, and presentation of geographic information (Fairbain *et al.* 2001). These basic functional units, however, have different requirements in a GIS application. For storage and management purposes, the focus is on the form, structure, and properties of geographic phenomena. In this context, the aim is a data model that captures the complexity of geographic phenomena and processes, and supports a wide variety of GIS functions. For presentation purposes, the objective is an effective exploration of the information. In this context, the goal is a data model that supports different media (e.g., three-dimensional graphics, sound, and dynamic imagery), real time feedback, and interactions. Due to these different and sometimes conflicting goals, it is very difficult to achieve efficiency using a single data model for VR and GIS domains. Thus, GIS, like other graphic-intensive applications, needs to deal with two different data models: one for the application domain and another for the presentation domain.

For the application domain, the discretisation of the spatial and the temporal domains (discrete or continuous) and the integration of spatio-temporal structures (integrated or hybrid) are critical factors in determining the appropriate data model (Raper 2000). The integrated approach assumes a world that is fully four-dimensional in nature. The hybrid approach recognizes differences in the nature of space and time and treats them differently. Due to the diversity and conceptualizations of the GIS domain, a single data model that accommodates all abstractions or functional requirements is unlikely (Herring 1991).

For the presentation domain, the type of information to be visualized (e.g., raster, two-dimensional or three-dimensional geometric representations) and types of interactions with the data are major requirements in specifying the appropriate data model. The conceptualization of a data model for the presentation domain must be robust and extensible to support the wide variety of conceptualizations of the application domain and incorporate the desired functionality for an exploratory analysis of the information.

In this thesis we are interested in the presentation and exploration of three-dimensional objects and their dynamics. It thus excludes from this review data models for the application domain, as well as two-dimensional and raster representations of the presentation domain.

2.3.1 Three-Dimensional and Interactive Graphics

Prior to the early 1990s, almost all three-dimensional applications were based on some kind of graphic package such as GKS (ANSI 1985), PHIGS (ANSI 1988), and OpenGL (OpenGL 1992) to produce visual representations of two and three-dimensional objects. These graphical packages were attempts to create a device-independent model for the developers of graphic applications. In this way, these packages can be positioned in an intermediate level between the application program and the graphic hardware (i.e., they act as the data model of the presentation domain).

Despite the success experienced by some graphics packages, they have critical limitations. First, the level of interaction with the model is limited. Most applications rely on programmers' skills and hard-coding to achieve the required interactivity. Second, the level of abstraction of the presentation domain is very low and keeps no relation with the modeled application domain. The internal structure of these packages is based on a list of drawing commands representing polygons and faces to be rendered by the graphic engine. Third, these packages have no explicit notion of time. Time can be specified only implicitly, as a sequence of operations on the display-list representation (Foley *et al.* 1997).

In order to overcome the limitations of these graphic packages, the scientific-visualization community has proposed many different data models to represent interactive and dynamic three-dimensional graphics (Zeleznik *et al.* 1991; Strauss and Carey 1992; Koved and Wooten 1993; Elliott *et al.* 1994; Najork and Brown 1995; Green and Halliday 1996; Dollner and Hinrichs 1997; Lee 1998b; Java-3D 2004). These models are presented in the form of object-oriented toolkits or high-level API, which give developers an abstract and extensible representation to construct presentation applications.

2.3.2 A Framework for Animated Three-Dimensional Graphics

Although there are some differences in the architecture and functionality of three-dimensional toolkits and APIs, the majority of proposed data models partitions the presentation domain into three main logical parts. This framework models the animated objects (who), the actions these objects undergo (what), and the times during which the objects undergo the actions (when).

The who-part refers to the geometric model, representing objects that compose the scene. The what-part of the framework represents the object's behavior and is called the action model. Entities of the action model represent the evolution of an object's attributes with the passage of time. The when-part of the framework represents the temporal model. Entities associated with such part of the framework model the temporal configuration of objects' behavior.

2.3.2.1 Geometric Models

The geometric model describes the geometry, position, and appearance of visual objects that populate the environment. Objects of the geometric model are instances of classes provided by the toolkit or sub-classes inherited by the developer of the application. These objects are collected and organized in a direct acyclic graph structure called a *scene graph* (Strauss and Carey 1992; Koved and Wooten 1993; Najork and Brown 1995; Dollner and Hinrichs 1997; Lee 1998b; Java-3D 2004). In this tree-like structure, each object represents a node and performs some specific function in the model. Links between nodes represent a parent-child relationship. These links draw the structure of the graph; grouping related objects and directing how operations and properties are propagated along the structure.

A scene graph is a powerful graph-based construction paradigm for three-dimensional applications. The hierarchical structure of the scene graph and the grouping mechanism allow for the representation of meaningful structures, which can be used to incorporate the semantics of the application domain. Nodes in the scene graph have instances data called fields. Values associated with fields define the state of an object. These values can be either a simple value or a reference to another object. Some models use a "pure" object-oriented approach and encapsulate all information of the object in its *fields* (Koved and Wooten 1993; Java-3D 2004). In other models, object's *fields* capture only attributes that are strictly related with the class of the object (Strauss and Carey 1992; Najork and Brown 1995; Dollner and Hinrichs 1997; Lee 1998b). In this model, objects inherit other attributes from their parents or share a common attribute with their siblings.

The organization of a scene graph is constrained by the structural role of the node. Based on its structural role, a node can be classified as a root, a group, or a leaf node. The root node is the unique node in the structure without a parent. This node marks the start point of the graph and it is used to define default characteristics of the environment. A branch-group node is a node that supports children. This type of node creates new branches in the scene graph and it is used to group semantically related objects, imposing a meaningful and coherent structure on the tree. A leaf node is a terminal node in the structure (i.e., a leaf node does not admit child nodes). Leaf nodes are responsible for the visual content of the scene.

Although the structure of scene graphs is preserved among different toolkits, the semantics of the nodes may have slight variations. As a rule, leaf nodes are divided into four basic categories: shape, property, camera, and light. Shape nodes represent geometric objects (e.g., volume primitives, boundary representation primitives, and texts). Property nodes are objects that represent all visual aspects of the object that are not related to its geometry (e.g., color, texture, location). The camera node defines the position and orientation of the camera in the environment. This node models the vantage point of the observer in the environment. Lights nodes illuminate the environment, creating more realistic scenes.

Consider, for instance, an application depicting some geometric objects (Figure 2.2a). In the application domain context, these objects represent a clown and a box. In the presentation domain context, these objects are geometric primitives provided by virtually all three-dimensional toolkits (i.e., sphere, cones, and cube).

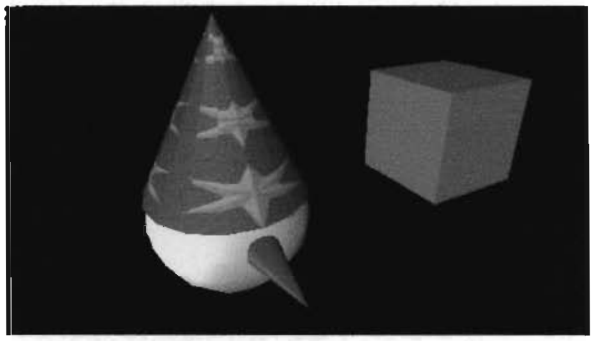
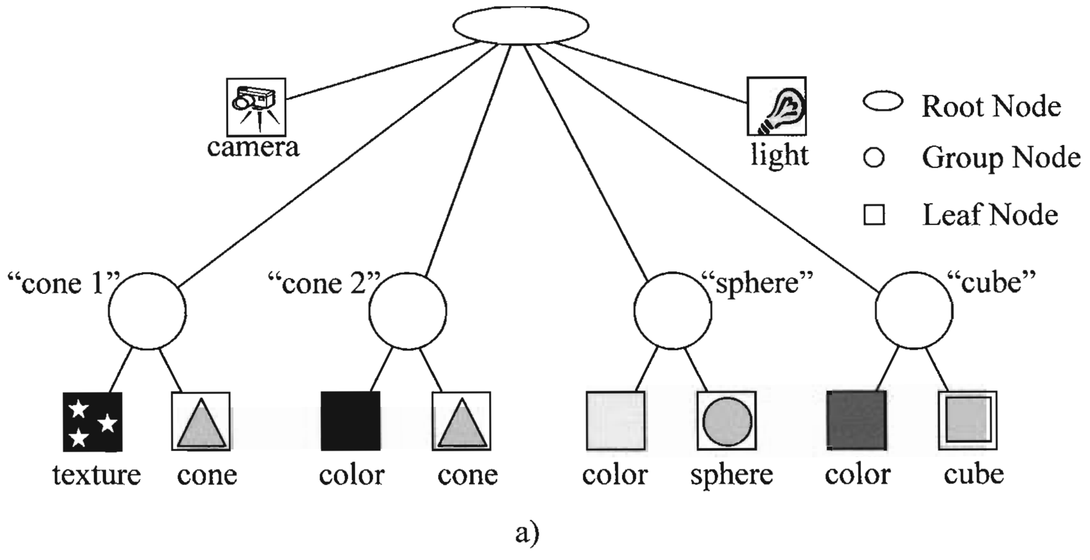


Figure 2.2. Graphical realization of a clown and a box based on geometric primitives

The geometric primitives that model the clown and the box are stored in a scene graph. There are different strategies to organize these primitives in a scene graph structure. The simplest way is to add the geometry of each object directly to the scene graph structure without using any grouping mechanisms (Figure 2.3a).

A presentation application, however, can take advantage of the grouping mechanism of the scene graph and impose a meaningful structure on the graph, that is, a structure that incorporates semantics of the application domain (Figure 2.3b). The strategy used does not change the content of the model nor the information perceived by an observer.



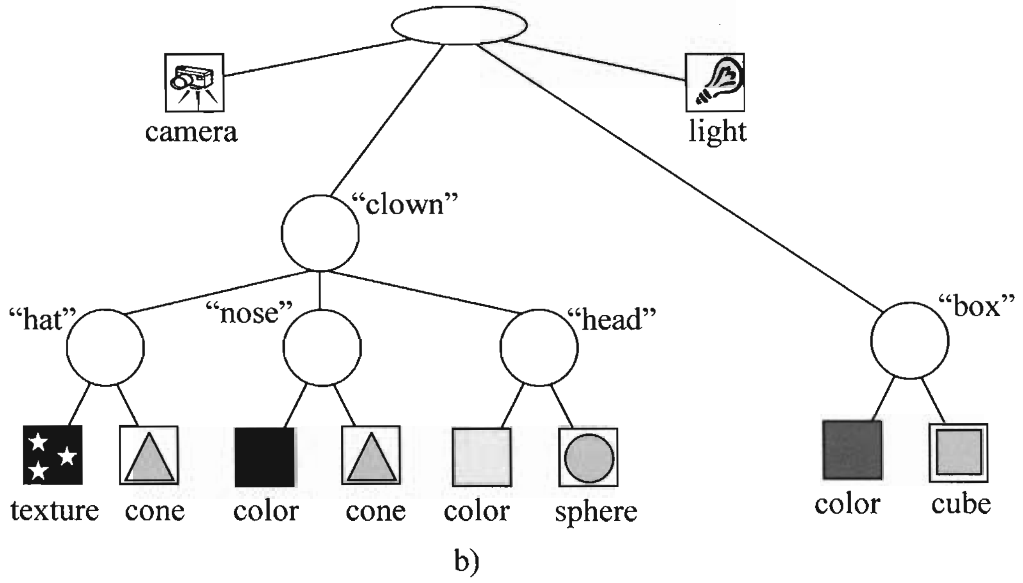


Figure 2.3. Representation of geometric primitives in a scene graph structure: a) a scene graph without the grouping mechanism, and c) a scene graph with primitives grouped accordingly the semantics of the application domain.

2.3.2.2 Action Models

The action part of the framework represents behaviors of objects. Behaviors are any modification in the state of an object that can be perceived by an observer. A behavior can be directed either by a user action or by the passage of time. Based on the type of object that is experiencing the action and the agent that causes the change, behaviors can be classified into four categories: *interaction*, *navigation*, *guided navigation*, and *animation* (Table 2.1).

	Object Changing	
Cause of Change	Camera	Shapes, Properties, Transforms, and Lights
User	*Navigation*	*Interaction*
Time	*Guided Navigation*	*Animation*

Table 2.1. Classification of types of changes based on the object changing and the agent that causes the change.

Navigation and *Guided Navigation* are behaviors that change the position and orientation of the camera, thus defining the vantage point of the observer in the environment. The difference between these two behaviors is the agent that causes the change. *Navigation* is a type of behavior entirely controlled by the observer. The observer using some navigation mechanisms directs the movement of the camera, performing a walk-through or a fly-by in the environment. *Guided Navigation* performs the same kind of movement, but does not dependent on the observer action. In this kind of behavior, the movement of the camera is previously defined or recorded in the model and can be played at any time, giving the observer a tour around the environment. Time in *Guided Navigation* is implicitly represented by the evolution of the movement of the camera.

Interactions and *Animations* are behaviors associated with visual objects or objects that change the appearance of the scene. *Interactions* are triggered and fed directly by user actions. The act of an observer examining an object by continuously rotating it around a specific axis, for example, is supported by a *rotate* behavior. This kind of interaction is called active, since it depends on the user's desire to execute the action. Other types of interactions are performed in automatic fashion by the application and are called passive. Passive interactions are transparent for the observer and do not depend on the intention of the user to perform the action. The *level of detail* behavior is a type of passive interaction, which is based on the position of the observer in the environment. The *level of detail* behavior loads different versions of the object based on the distance between the observer and the object. The idea is to present a refined version of the object only when the observer can perceive details, thus eliminating the computational cost of rendering pieces of the object that cannot be seen. Other types of interactions are constraints imposed by the objects on an observer action. The *collision detection* mechanism is an example of such kind of interaction. This mechanism limits the movement of the observer in the environment (e.g., prohibiting the observer to walk through a wall). The *collision detection* mechanism can be extended to treat interactions among the objects in the environment as well (e.g., an object that blocks the passage of a moving object).

Animations are behaviors directed by the passage of time. *Animation* behaviors define how some objects' visual attributes change their values as time passes. The action that triggers animation behaviors can be a direct user action, another behavior, or an instant in time. Once triggered, however, the only external stimulus that keeps the animation running is time.

The mechanisms associated with each kind of object's behaviors (i.e., *interaction, navigation*, *guided navigation*, and *animation*) are not mutually exclusive. Instead, it is a desired characteristic of presentation applications that they support all types of behaviors. In this thesis, however, we are interested in animations behaviors. These behaviors represent the dynamic characteristics of application domain's objects.

The approach used to implement animation behaviors varies among different toolkits. Some toolkits implement these behaviors as member functions of the visual objects (Koved and Wooten 1993), as a list of commands representing the evolution of an object's action (Zeleznik *et al.* 1991), or as an object behavior connected to a visual object (Strauss and Carey 1992; Koved and Wooten 1993; Najork and Brown 1995; Green and Halliday 1996; Dollner and Hinrichs 1997; Lee 1998b; Java-3D 2004). The latter approach has the advantage of isolating the action and geometric parts of the framework. This characteristic enhances scalability and facilitates the implementation of the model.

The toolkits that model animation behaviors as first class entities provide a set of built-in classes that represent basic behaviors (e.g., position, color, and orientation). These classes carry information and functionality to produce an object behavior, that is, an attribute specifying visual objects to be animated, the end states of the object's behavior, and an interpolation function to generate intermediate values (Figure 2.4). The object behavior based on stimuli received from objects of the temporal model makes computations and informs the visual object of its new state.

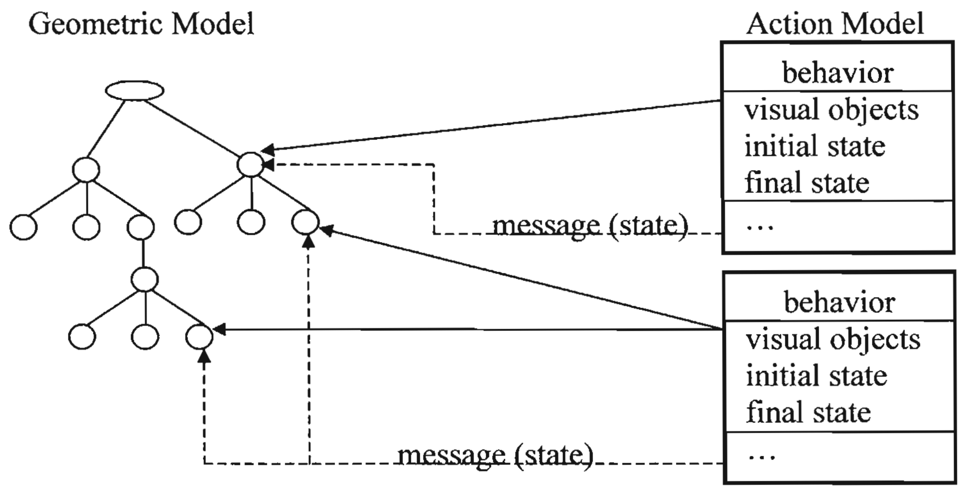


Figure 2.4. Two behaviors of the dynamic model and their associated visual objects of the geometric model. The object behaviors continuously inform the new state of their associate visual objects.

Since animation behaviors are directly connected with an object of the scene graph, they are highly specialized classes (i.e., each object behavior models the evolution of a specific type of visual object). If an object changes its color, position, and shape, for instance, it is necessary to assign one behavior for each changing characteristic of the object.

A simple list of objects is the traditional approach used to structure the collection of behavior objects that forms the dynamic part of the framework (Strauss and Carey 1992; Najork and Brown 1995; Green and Halliday 1996; Java-3D 2004). In this approach, the position of the behavior in the list is irrelevant. This highly unstructured organization has several drawbacks. First, the structure of the dynamic domain does not follow the structure of the geometric domain. There is no entity in the model that aggregates groups of related behaviors. Second, this list of object behaviors is neither semantically nor temporally indexed. These characteristics make it difficult, for instance, to retrieve a temporal sequence of behaviors of a certain object. Third, the unstructured nature of this list of objects makes the resulting code hard to parallelize and difficult to manage as the complexity of the animation increases.

In order to overcome these drawbacks some toolkits impose a more structured organization of the dynamic domain (Dollner and Hinrichs 1997; Lee 1998b). These models propose an entity with a high-level abstraction encompassing the activity of a certain object (e.g., an activity object). The activity object stores a collection of behaviors associated with a visual object. Although activity objects represent a step forward in the organization of the action model, these entities fail to support the representation of complex behaviors. If an object has a period of inactivity, for instance, its behavior is modeled using different activity objects (i.e., one activity for each period when the object is performing some movement). In this way, it is impossible to aggregate the entire behavior of the object in a single representation. Thus, activity objects transfer the problems found in other data models to a high-level of abstraction.

2.3.2.3 Temporal Model

In existent three-dimensional graphics toolkits, the temporal model is by far the least abstract part of the framework. The simple temporal structures of such models permit only the representation of quantitative and absolute conceptualizations of time. In these models non-linear representation of time (e.g., cyclic time) is a weak approximation, and qualitative and relative representations of time are difficult to accomplish.

Data abstractions of the temporal part of the framework are critical in a model for exploratory analysis of animations. The conceptualization of this part of the framework needs to support the richness and complexity of the temporal structures of the application domain. Failing to satisfy these requirements means that information of the application domain is lost or represented only by an approximation in the data model that supports the presentation application.

The majority of temporal models have a unique type of object that carries all temporal characteristics of an objects' behavior (Strauss and Carey 1992; Najork and Brown 1995; Green and Halliday 1996; Java-3D 2004). In these models, a temporal object has an associate behavior, the duration and the condition that trigger the behavior (e.g. a certain instant in time), and a mechanism to feed the behavior with information that translates the passage of time (Figure 2.5).

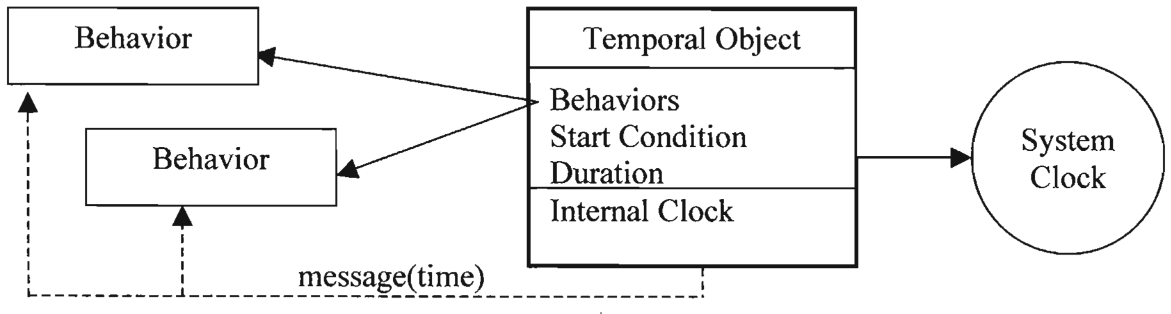


Figure 2.5. A temporal object representing the temporal characteristics of two behaviors. The temporal object continuously sends messages to its associated behavior.

A temporal object can have more than one associated behavior. The behaviors associated with a temporal object, however, share the same temporal characteristics (e.g., start condition and duration) and receive the same set of messages from the temporal object. Temporal objects have a link to a system clock. As a consequence, all behaviors, even those based on different temporal objects, are synchronized.

The major problem of existent toolkits for animated three-dimensional graphics is that they were designed for efficient running of the animation and not the user of virtual exploration of animations. Some models have suggested extensions for the temporal part of the framework to incorporate qualitative and relative representation of time and temporal relations (Dollner and Hinrichs 1997; Lee 1998b). These models, however, are limited by a poor conceptualization of the action domain and they still are not concerned with the user of the animation. The exploration of an environment's dynamic content is based on the ability of a user to change the temporal configuration of modeled behaviors. Therefore, the user needs a more abstract representation of the object behavior and a set of operations that allows him or her to combine these behaviors to create new views of the dynamic environment.

2.4 Summary

This chapter reviewed related work on representation of multi-dimensional GIS. Different types of animations and their use in GIS applications were investigated. In the context of temporal VRGIS, basic requirements for the integration of VR and GIS and animated three-dimensional data models were discussed.

CHAPTER 3

ACTIONS

In a virtual environment, objects are roughly classified in two major categories: static and dynamic. This classification is based on the capability of the object to change the values of its sensorial attributes. Dynamic objects are those objects that allow the modification of their sensorial attributes with the passage of time. Sensorial attributes are attributes that can be communicated to a user by exploiting any user's sensorial channels. Thus, sensorial attributes include visual attributes such as shape and texture, as well as non-visual attributes such as sound, weight, smell, or temperature. Dynamic objects have associated objects that inform how their sensorial attributes evolve over time. These objects are collectively called *action* objects. *Action* objects represent the behavior of dynamic objects in the environment.

The mechanism that directs the evolution of dynamic object's attributes with the passage of time is called animation. A critical issue in exploring an animation is the user's ability to manipulate dynamic objects. This ability depends on the level of abstraction used to represent objects' behavior (i.e. actions), as well as on the set of operations available to manipulate this information. This chapter presents a conceptualization of the action part of a framework for virtual exploration of animations. In the next section we discuss the major requirements of the model and introduce its general structure. Then we present some characteristics of each element in the model through an informal specification.

3.1 Structure of the Action Model

In order to allow the user to manipulate and control the presentation of animations, the animation model needs to address two major requirements (Campos *et al.* 2003a). First, the model needs to support the representation of pre-orchestrated behaviors (i.e., behaviors that are completely known ahead of time). The fact that behaviors are known in advance gives the user the *possibility* of manipulating the dynamic environment. Second, the model needs a reasonable structure representing different granularities of objects' behaviors. Such a structure gives the user the *ability* to manipulate the dynamic environment. A model based on the keyframe paradigm, using cognitively plausible representations of objects' behaviors, addresses these requirements.

The conceptualization of the action part of the framework considers distinct granularities of the dynamic environment. Elements at different granularities represent increasing abstractions of the animated objects' behaviors, which are built from the highest to the lowest level of granularity (Figure 3.1).

Act and *Course of Actions* represent different granularities of the behavior of a single object. *Acts* represent pieces of an object's behavior. For example, a car making a turn or moving beteween two locations. *Course of Actions* aggregates *Acts* forming the entire behavior of an object. Each *Course of Actions* has an associated *Performer*. *Performer* is an abstraction that represents the geometry and appearance of the dynamic object. At the coarsest level of granularity, *Animation* aggregates pairs of *Course of Actions* and

Performer representing the behavior of groups of objects. *Animations* can also aggregate other *Animations* forming complex *Animations*.

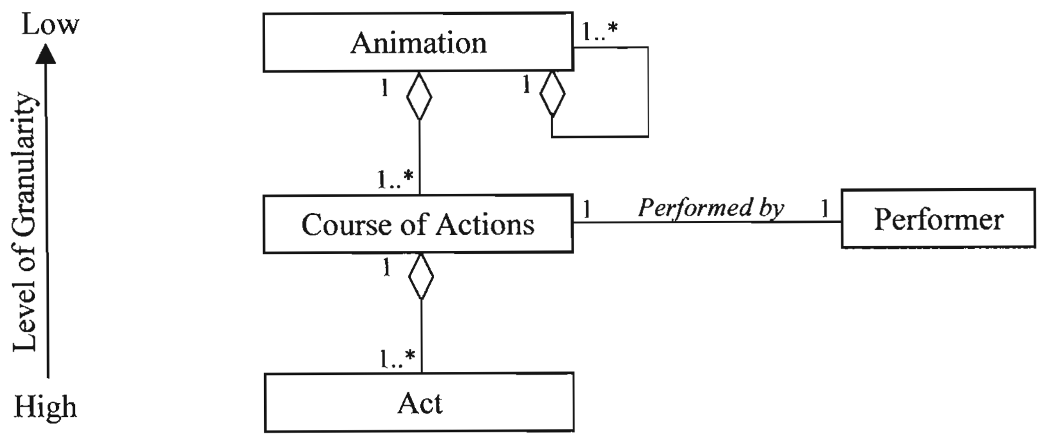


Figure 3.1. Class diagram representing the general structure of the action model.

The next sections present an informal specification of each class of the action part of the framework. These specifications discuss only fundamental operations used to define the structure of the model and to impose constraints on the association of their instances. A more detailed definition of these classes is presented together with a prototype implementation in chapter 8.

3.2 Acts

Acts are the building blocks of an object's behavior. The behavior of an object can be represented by a single *Act* or by any combinations of *Acts*. The number of *Acts* needed to represent the behavior of an object depends on the complexity of the behavior and on the requirements of the application domain. Consider, for instance, the behavior of a car traveling between two locations (Figure 3.2a). The behavior of the car can be divided in small pieces of information representing different segments of the trip. Each segment corresponds to an *Act* of the object's behavior. The movement of the car making a turn, for example, represents one act of the entire trip (Figure 3.2b).

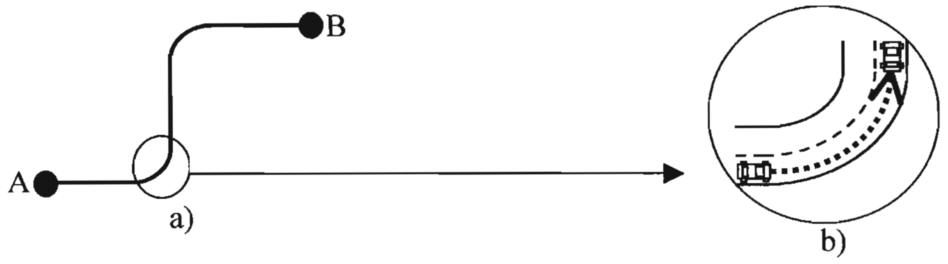


Figure 3.2. A car traveling between two locations: a) the entire behavior of the car and b) the act of the car making a turn.

From the user's perspective, an *Act* is the smallest piece of an object's behavior that can be manipulated. As far as a user is concerned, *Acts* represent the finest granularity of an object's behavior. The manipulation of *Acts* by a user is accomplished through the manipulation of their temporal characteristics discussed in chapter 4.

From the modeling perspective, an *Act* is an abstraction that encapsulates all the information needed to generate the evolution of an object's attributes with the passage of time. Each *Act* has four components: (1) a list of types of attributes being interpolated, (2) a list of key values for every attribute being interpolated, (3) interpolators to compute the intermediate attribute values, and (4) temporal information about the *Act*. These components are modeled as a separate classes and associated with the main class *Act* (Figure 3.3). The class diagram shows a dependency (dashed arrows) among some components of the *Act*. These dependencies will be discussed later in this chapter.

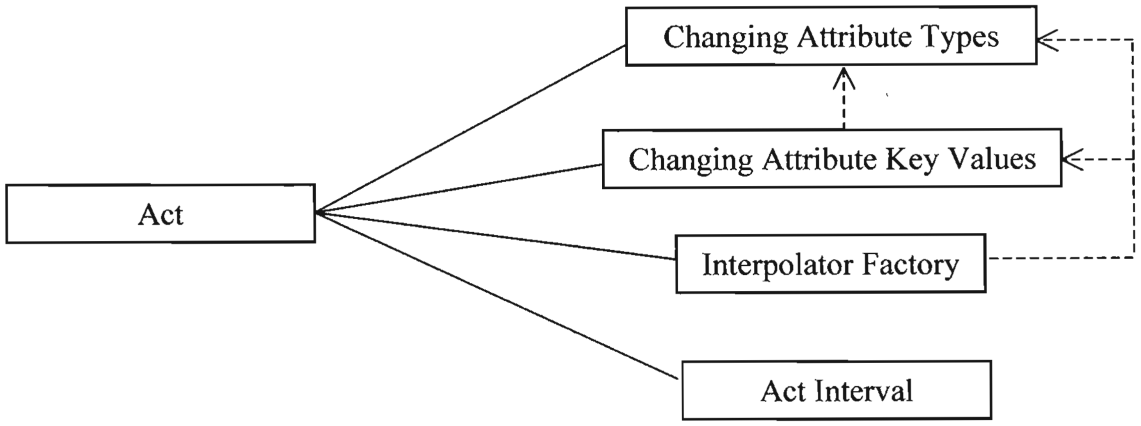


Figure 3.3. Class diagram of the act and its components.

In order to understand the role of each component in the model, consider the act of a car making a turn. Consider yet that the bend of the road is unpaved. Thus, cars get dirty when going through this segment of the road. The dirt is modeled by gradually darkening the color of the car (Figure 3.4).

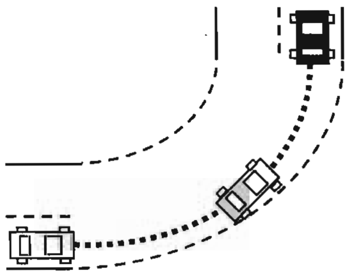


Figure 3.4. The car changes its color when traversing an unpaved segment of the road.

The *Act* of the car making a turn and getting dirty involves the evolution of three types of attributes (i.e., position, direction, and color). In this example, the car, originally with a yellow color, becomes gradually brown as it moves along the bend. Suppose in addition, that the car decelerates in the first half of the bend, accelerates in the second half, and takes 5 seconds to complete the turn. All these pieces of information are encapsulated in an *Act* object. The state of this *Act* and its associated components can be depicted in an object diagram (Figure 3.5).

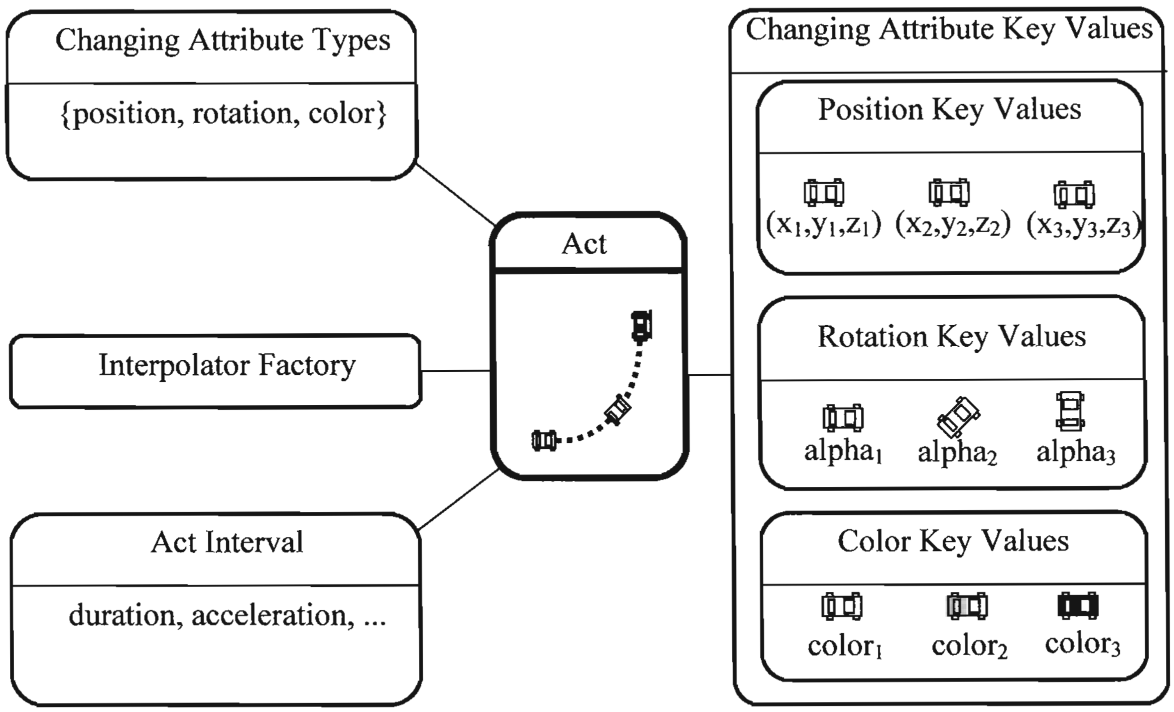


Figure 3.5. Object diagram of a car making a turn and getting dirty.

An instance of *Changing Attributes Types* class carries the types of attributes being interpolated. An object *Changing Attributes Key Values* stores a list of relevant values of each attribute listed in the previous object. An object *Act Interval* encapsulates all temporal-related information of the act (e.g., duration, and acceleration). An object *Interpolator Factory* takes all this information into account and produces the object *Interpolator* that, when properly executed by a system clock, produces the "continuous" movement of the car.

One important characteristic of an *Act* is that all attributes have an active state during the entire act (i.e., at any instant each attribute specified by the *Changing Attributes Types* object has a different value from the value it had in the previous instant of observation). If the car changes its color only during a fraction of the duration of the *Act* (e.g., only half of the bend is unpaved), the color attribute cannot be used together with attributes representing the position and direction of the car making a turn. To handle this situation, the evolution of the color attribute must be modeled as a separate *Act* and combined with other *Acts* at a higher level of abstraction.

The next sections discuss some components of the *Act* (i.e., *Changing Attributes Types*, *Changing Attributes Key Values*, and *Interpolator Factory*). The class *Act Interval* is discussed together with other temporal characteristics of actions, presented later in this thesis.

3.2.1 Changing Attributes Types

The class *Changing Attributes Types* represents a collection of attributes types that change during the *Act*. These types correspond to any sensorial attribute of the *Performer* (e.g., color and position). For the sake of simplicity, we deal in this thesis only with sensorial attributes that have a graphical realization. This fact, however, does not constitute a limitation of the model. The model is robust enough to accommodate all types of sensorial attributes.

Figure 3.6 shows the specification of the class *Changing Attributes Types*. Instances of this class can be seen as a set of strings in which each element of the set represents a type of attribute. One important operation shown in the specification is `hasIntersection`. This operation checks if two sets of attributes (i.e., two instances of *ChangingAttributeTypes*) have at least one element in common. The operation `hasIntersection` is used in the context of coordination of multiple changes, discussed later in this chapter.

class ChangingAttributeTypes {

Overview: An unbounded and non-empty set of strings. A typical *ChangingAttributeTypes* is $\{S_1,\ldots, S_n\}$, where S_i is a string with the name of the type of attribute (e.g., position and color).

// Constructors

...

// Methods

boolean `hasIntersection`(ChangingAttributeTypes cat)

Effects: Returns true if the argument has at least one common type of attribute with this object.

...

Figure 3.6. Specification of the class ChangingAttributeTypes.

The class *Changing Attributes Types* deals only with types of attributes being interpolated. Thus, there is no information in this class about the key states of these attributes or how to generate the evolution of these attributes with the passage of time. These functionalities are part of the *Changing Attributes Key Values* and *Interpolator Factory* objects associated with the *Act*.

3.2.2 Changing Attributes Key Values and Interpolator Factory

In an animation model based on the keyframe paradigm the information needed to generate the evolution of an object's attributes over time are: (1) some representative states (key states) of the attribute, (2) an interpolator function to compute intermediate states between key states, and (3) temporal information (e.g., the duration of the act). In our model, an instance of the class *Changing Attributes Key Values* stores representative states of the *Act*. An *Interpolator* object generates intermediate state values. The *Interpolator Factory* object builds the appropriate *Interpolator* based on the type of attribute and the number of key states. Finally, an *Act Interval* object carries out the temporal characteristics of the *Act*.

Changing Attributes Key Values is a class that encapsulates key states of the *Act*. These states are stored in a list for each type of attribute being interpolated in the act. There are as many lists of values as there are types of attributes being interpolated. This characteristic is represented by a dependency between the classes *Changing Attributes Key Values* and *Changing Attributes Types* in the class diagram (Figure 3.3). Elements of each list store the value of a certain attribute and its relative position inside the act (i.e., each state is associated with a number between 0 and 1, inclusive). These numbers are called *normalized times*. *Normalized times* represent the relative position of key states of an object inside the act. The first and last states of the attribute hold at the *normalized time* 0 and 1, respectively. Other key states, if they exist, hold in the interval between 0 and 1, exclusively. *Normalized times* allow the representation of an object's behavior without an explicit reference to time. This characteristic allows the encapsulation of all temporal information and temporal related operations in a single object (i.e., *Act Interval*).

Figure 3.7 depicts some *key values* and *normalized times* for the example of the car making a turning. This example shows that there is no constraint that the number of elements in the lists be equal. The attribute position, for instance, has three key states, while the attribute direction has only two.

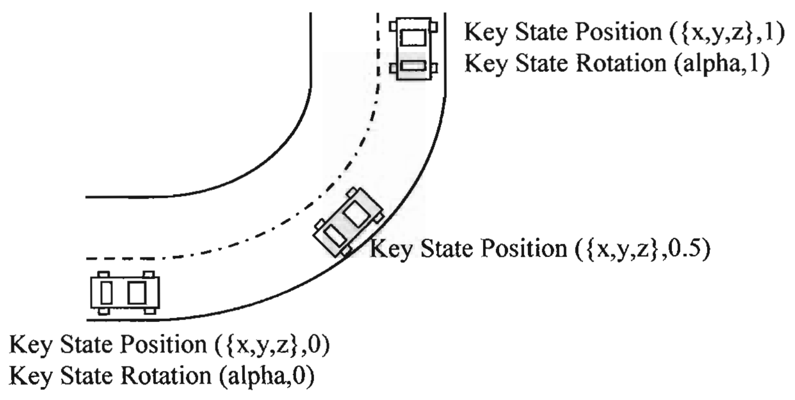

Key State Position ({x,y,z},1)
Key State Rotation (alpha,1)
Key State Position ({x,y,z},0.5)
Key State Position ({x,y,z},0)
Key State Rotation (alpha,0)

Figure 3.7. Key states of the position and direction of a car making a turn.

The *Interpolator Factory* object is responsible for building *Interpolator* objects for the *Act*. There is a wide variety of *Interpolators* that can be used to generate intermediate states of an act. The best option depends on the type of attributes being interpolated and the number of key states available. This characteristic is represented by a dependency between the classes *Interpolator Factory* and the classes *Changing Attributes Types* and *Changing Attributes Key Values* (Figure 3.3).

Depending on the requirements of the application domain, the level of realism can also be taken into account by the *Interpolator Factory* Object. Figure 3.8 shows two possible configurations of a car making a turn. The first configuration has two key states and a non-linear interpolation function (Figure 3.8a). The second configuration has a sequence of key states and a linear interpolation function (Figure 3.8b). The graphical realization of these two *Acts* represents different approximations of the real movement. The first one generates a more realistic movement of the car, while the second generates a coarse approximation of the real movement.

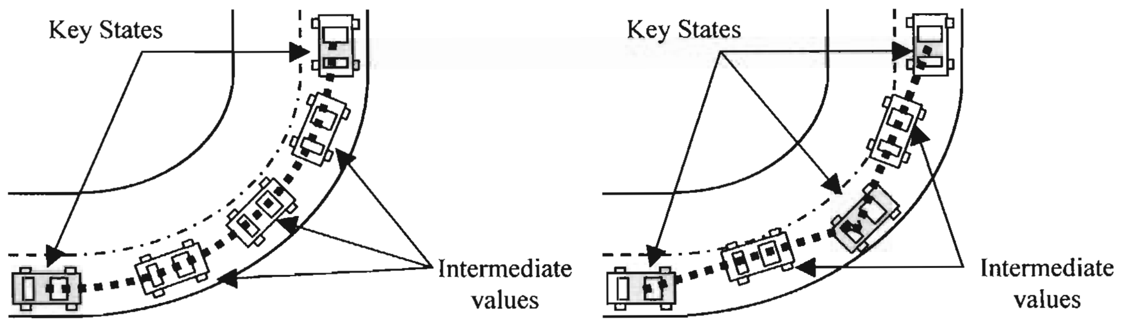


Figure 3.8. Two possible configurations of a car making a turn: a) a non-linear approximation of the movement, and b) a linear approximation of the movement.

Interpolator Factory and *Changing Attributes Types Values* are powerful abstractions for both users and implementers of the model. On one hand, the user "sees" each act as a single object, and there is no need for the user to think about types of attributes or interpolation functions. On the other hand, implementers can use the *Interpolator Factory* abstraction to build interpolators that generate the desired act of an object. It means that the complexity of the act does not require implementers to artificially break acts with a complex behavior in more simple acts. Implementers must use the *Interpolator Factory* class to provide the code to generate complex acts. The act of an object must be defined by the application domain only, and not constrained by limitations of the implementation.

3.2.3 The Act Specification

The components of the *Act* object encompass almost all functionalities of this small piece of an object's behavior. In this sense, the specification of the *Act* class itself becomes very simple. The class *Act* has four attributes, each one representing a specific component of the *Act* (Figure 3.9).

```
class Act {

    Overview: acts represent a piece of a performer's behavior.
    // Attributes
        ChangingAttributesTypes cat;
        ChangingAttributesValues cav;
        ActInterval ai;
        Interpolator i;
    // Constructors
        ...
    // Methods
    boolean isRelated(Act a)
      Effects: Returns true if the component ChangingAttributesTypes of this object
               has a non-empty intersection with the same component of the act used
               as argument.
```

Figure 3.9. Specification of the class Act.

For the effort of coordination of multiple changes, the *Act* implements the operation `isRelated`. This operation defines whether two *Acts* are related, that is, if two *Acts* have at least one attribute in common. Objects of a higher level of abstraction use this operation to avoid certain combinations of *Acts*.

It is important to emphasize that there is no operation that allows a user to manipulate the types of attributes and their key states, or to define the *Interpolator* used in the *Act*. All manipulation of an object's behavior is accomplished through the manipulation of the objects' temporal characteristics, which is discussed in Chapter 4.

3.3 Course of Actions

The *Act* abstraction represents the finest granularity of an object's behavior. In a virtual exploration of an animation, however, a user needs the flexibility to explore coarser representations of the dynamic environment as well. Sometimes a user is more interested in knowing if an object starts its behavior before or after another object's behavior or if two objects are performing their associated behavior simultaneously. In this sense, it is important for a user to reason over abstractions that represent the behavior of the object as a whole and not its constituent parts.

Course of Actions is an abstraction that represents the entire behavior of an object. A *Course of Actions* is a combination of *Acts*. Consider, for instance, the entire behavior of the car moving between two locations (Figure 3.10). This behavior is modeled with five individual *Acts* that capture "representative" segments of the trip. Those *Acts* are combined forming the *Course of Actions* of the car. The second *Act* of the car's trip, for example, corresponds to the case in which the car is traveling an unpaved segment of the road discussed earlier in this chapter.

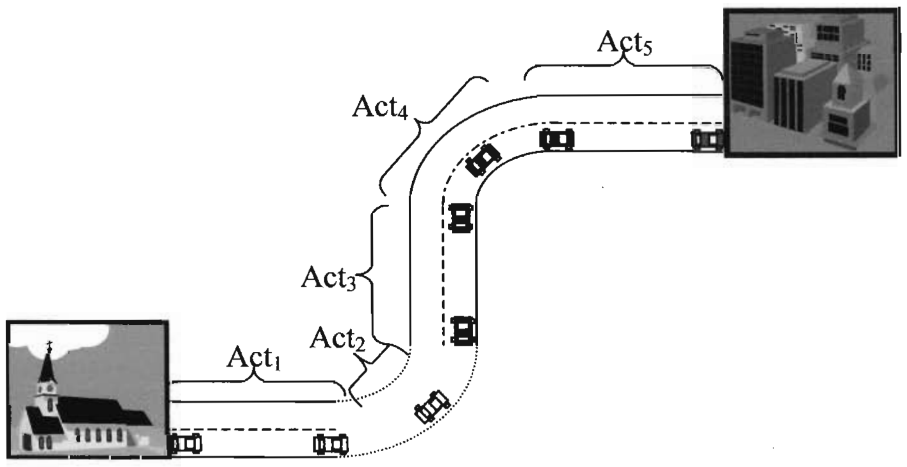


Figure 3.10. A Course of Actions of a car traveling between two locations.

The object diagram in Figure 3.11 depicts an instance of the class *Course of Actions* representing the behavior of the car traveling between two locations. This object has an association with individual *Acts, a Performer*, and a *Course Of Actions Interval* Objects. The later object is not depicted in the structure of the action's conceptual model (Figure

3.1). For the sake of simplicity, the class diagram in Figure 3.1 does not show the association between entities of the action part of the framework and their temporal characteristics. This class is discussed in details in Chapter 4.

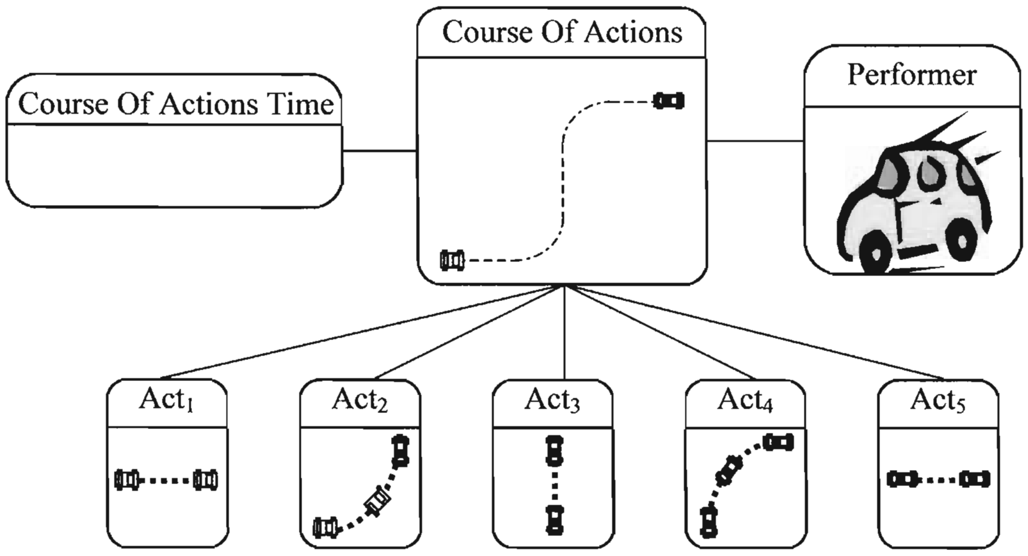


Figure 3.11. Object diagram of the car traveling between two locations.

The Specification of the *Course Of Actions* class (Figure 3.12) shows the attributes used to store its components (i.e., a *Performer*, a *Course of Actions Interval*, and a list representing a set of *Acts* objects).

The process of building a *Course of Actions* is accomplished by instantiating an object with its associate *Performer* and *Course of Action Interval* objects. This process continues with the addition of *Acts* to the *Course of Actions* set of *Acts*. Individual *Acts* of an object's behavior, however, cannot be arbitrarily added to the set of *Acts*. The coordination of multiple changes prohibits that two *Acts* that "compete" be stored in the set. The operation `compete` of the *Course of Actions* class handle this task.

```
class CourseOfActions {

    Overview: Course of actions represents the entire behavior of a performer's
              behavior.
    // Attributes
        Performer p;
        CourseOfActionsInterval coai;
        Acts[] acts;
        ...
    // Constructors
        ...
    // Methods
    boolean compete(Act a, Act a1)
    // Effects: If a compete with a1 returns true else returns false.
              Return a.isRelated(a1) ∧ coai.isConcurrent(a,a1)
        ...
```

Figure 3.12. Specification of the class Course of Actions.

The operation `compete` identifies when two *Acts* are competing to interpolate the same attribute. Two *Acts* "compete" if they are "related" and their *Acts Intervals* objects are "concurrent". The first requirement is verified by the operation `isRelated` of the *Act* class. The second requirement is checked by the operation `isConcurrent` of the *Course of Actions Interval* class. This operation is discussed in the next chapter. Intuitively, the operation `isConcurrent` informs if the temporal intervals representing the periods of time when each *Act* occurs overlaps along the animation timeline. In the case of the car traveling between two locations, for example, all *Acts* are related (i.e., every *Act* interpolates at least the attribute position of the car). These *Acts*, however, can be combined forming the desired *Course of Actions* of the car. This combination is possible only because in such a *Course of Actions* of the car one *Act* follows the other and they do not overlap in the temporal domain (i.e., the *Acts* are not concurrent).

Although the *Course of Actions* abstraction represents already a coarser granularity of an object's behavior, this representation is not always suitable for a real application. In a typical animation the number of *Performer* objects can easily overcome the user's capability to analyze the dynamic environment. An environment composed of hundreds or thousands of animated objects requires increasing coarse representations of objects' behaviors. In this way, we need to extend the model with abstractions that represent groups of possibly semantic-related *Course of Actions* objects.

3.4 Animations

The coarsest level of granularity of the dynamic environment is modeled by means of the abstraction *Animation.* An *Animation* can represent the behavior of a single object, a group of objects, or all the objects in the environment. Consider, for example, an application running an animation of all vehicles traveling between the two locations. Based on the semantics of the application domain, it is possible to build increasing abstractions of the dynamic environment. Objects with related semantics can be grouped in a single *Animation. Animations* can also be grouped forming an even more abstract collection. Consider, for instance, that the vehicles are traveling between two buildings in a university campus and that the vehicles belong to students, faculties, and staff personal. In this scenario, it is possible to have one *Animation* for every vehicle, different *Animations* for each class of vehicle owner, or an *Animation* with all vehicles in the environment. The first option is often unmanageable for humans, giving a large number of objects in the environment. The second option gives rise to a more reasonable number of objects (*Animations*), which can be manipulated by users. The last option is the approach used by existing applications (Strauss and Carey 1992; Najork and Brown 1995; Green and Halliday 1996; Java-3D 2004) and has the disadvantage of limiting users to exploring the animation as a whole. An object diagram with the second option is shown in the Figure 3.13.

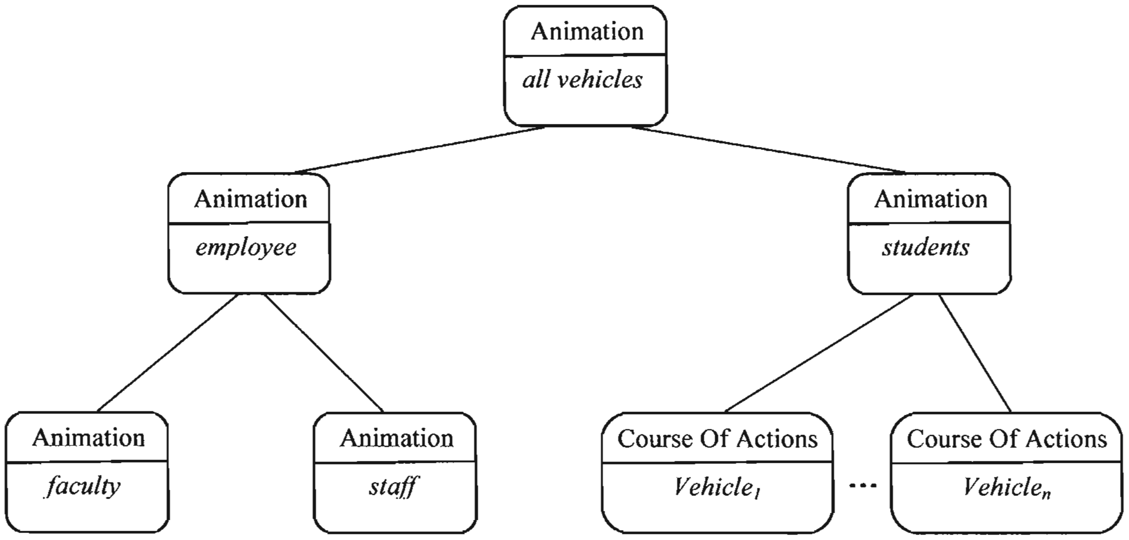


Figure 3.13. Object diagram of a possible configuration of Animations and Course of Actions of all vehicles running on a university campus.

Animations objects are formed by a combination of pairs of *Course of Actions* and *Performers* or by a combination of *Animations*, creating an even more complex animation. Thus, an Animation can be seen as a multi-sort collection of *Course of Actions* and *Animation* objects.

Figure 3.14 shows some details of the of the *Animation* specification. An attribute of this class represents a set of *Actions* objects. Due to the heterogeneous characteristic of the set, some operations of the animation specifications are overloaded (i.e., they have the same name but different types of attributes). Consider, for example, the operation `insert`. This operation has two versions, one adds a *Course of Actions* object and the other adds an *Animation* object in the set of *Action* objects. Although the *Act* abstraction is also of the type *Action*, this type of object is not allowed in the set of actions of the *Animation* class.

```
class Animation {

    // Attributes
        Actions[] actionSet;

        // Constructors
        ...
        // Methods
        void insert (CourseOfActions ca)
        // Effect: Insert a Course of Actions object in the set of actions.

        void insert (Animation a)
        // Effect: Insert an Animation object in the set of actions.
        ...
```

Figure 3.14. Specification of the class Animation.

The operation `insert` does not specify any constraint for the inclusion of *Animation* or *Course of Actions* objects in the set. The coordination of multiple changes (i.e., an attribute of a *Performer* object being interpolated at the same time) does not apply at this level of granularity. The behavior of a single object is completely encapsulated in the *Course of Action* abstraction and each *Course of Action* participate only once in the set of actions of the Animation object.

The conceptualization of the action part of the framework for virtual exploration of animation with increasing abstraction of objects' behaviors gives a user a cognitively plausible representation of the dynamic environment. The ability of the user to manipulate this representation, however, is strongly dependent on the configuration of the animation produced by the application that converts the information stored in the database to the internal representation of the animation model. Here, the maxim "garbage-in-garbage-out" is still valid. The configuration of the animation does not change the content of its presentation, but a poor configuration of the animation limits users' creativity and flexibility for manipulating the dynamic environment.

3.5 Summary

This chapter introduced the structure of the action part of a framework for virtual exploration of animations. Data abstractions of the model were presented and relevant operations were discussed through an informal specification.

Although the data abstractions of the action part of the framework does not allow a user to direct the presentation of the animation, these classes represent the foundation of the animation framework, which serves as the basis of all manipulations that can be accomplished by a user. All manipulations of the dynamic environment are accomplished by the modification of the temporal characteristics of actions objects (i.e., through operations of classes of the temporal part of the framework). These classes carry all temporal information about the behavior of dynamic objects. The next chapter introduces data abstractions that represent the temporal characteristics of *Actions* objects.

CHAPTER 4

TEMPORAL CHARACTERISTICS OF ACTIONS

In a model for virtual exploration of animations, the manipulation of the dynamic environment is accomplished through a set of operations performed over abstractions that represent temporal characteristics of actions. Hence, the exploration of dynamic environment is strongly dependent on the way that the observer perceives and manipulates time. This chapter presents a conceptualization of the temporal part of a framework for virtual exploration of animations. The next section discusses different temporal domains related with exploration of animations. Subsequently, we present relevant abstractions of the temporal model and discuss their characteristics.

4.1 Temporal Domains

Applications that support animations deal with information that has a temporal component. Such information is typically represented at different temporal granularities and based on distinct calendars (e.g., hours, days, academic terms, or geologic eras). In order to present these data in a way that is suitable for a user to analyze the temporal evolution of the information, it is necessary to perform mappings among different temporal domains.

In some GIS application (Dollner and Hinrichs 1997; Luttermann and Grauer 1999; Hardisty *et al.*), the presentation of temporal information through animations requires mappings among three temporal domains (Figure 4.1).

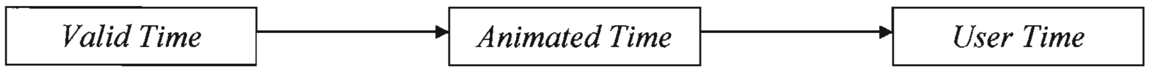


Figure 4.1. Mappings among valid, animated, and user time domains.

Valid time represents the time when the fact is true in the modeled reality (Jensen *et al.* 1992). For example, *valid time* is time generated by a simulation application or stored in a spatio-temporal database. This kind of information cannot be modified through operations of the animation model, but only observed by a user at a special and ephemeral point along the *user time* domain (i.e., the user present).

User time is the time in which the user senses the facts. Thus, *user time* is time as experienced by a user. This experience of time by the user can only be in the present and "going forward" at a fixed rate. Since *valid times* are fixed and *user time* cannot be manipulated, a direct mapping from the *valid time* domain to the *user time* domain constrains the user to explore the information as it “happened” or “will happen” in the modeled world. In order to allow the user to control the flow of information coming from the *valid time* domain, we need to represent such information in an intermediate temporal domain that can be manipulated. This intermediate temporal domain is called *animated time*.

The mapping from *valid times* to *animated times* is made by an application in an automatic fashion. These mappings are performed during the production of the animation without the involvement of the user. The mapping from *animated time* to *user time*, however, is controlled by a user through a set of operations that modifies *animated time*. The simplest mapping between these time domains aligns the user present with a certain instant in the *animated time* domain, allowing the user to sense a single snapshot of the modeled reality. By continuously mapping subsequent *animated times* to the unfolding user's present, that user can sense the evolution of the modeled reality. We call this continuous process an *animation*.

Consider, for example, the behavior of two objects, as these behaviors are stored in a spatio-temporal database. The activity of each object can be represented as a temporal interval spanning the *valid time* timeline (Figure 4.2a). The temporal structure of the *valid time* space is linear, discrete, and unbounded at both ends. In this thesis we adopted the former representation. This choice is motivated by the fact that discrete time is the usual representation of both animations and GIS applications.

Valid time intervals are mapped onto the *animated time* domain by converting time units of the former temporal domain to time units of the *animated time* domain (usually milliseconds). The *animated time* domain, however, is bounded on the start and, eventually, bounded on the end. An arbitrary point (i.e., the start point of the animation) defines the origin of the *animated time* space (Figure 4.2b). It means that activities occurring before the animation start point are not mapped onto the *animated time* space and, therefore, cannot be seen. Finally, the animation start point is mapped to the user present. As time goes by, all subsequent *animated time* instants are mapped to the ever-evolving user present (Figure 4.2c). This mapping allows the user to perceive the temporal evolution of all activities of the *animated time* domain. Such a mapping continues until *animated time* instants reach the upper bound of the *animated time* space, or indefinitely, in the case where no end point is specified.

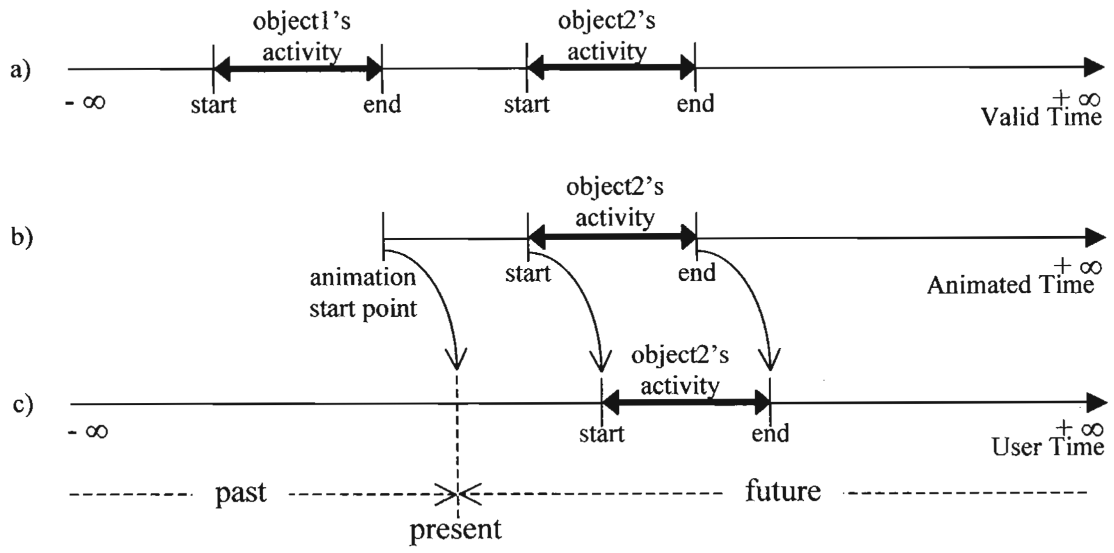


Figure 4.2. Mapping of temporal information among different temporal domains.

In the previous example, the evolution of *animated time* and *user time* spaces are synchronized (i.e., both temporal spaces evolve at the same pace and in the same direction). In this way, a user becomes a spectator of an animation of the modeled reality (i.e., a user is merely observing an animated version of the information stored in a database). The manipulation of the *animated space* gives a user the opportunity to play a more active role in the exploration of the animation. Thus, a user can create more appropriate views of the dynamic environment (i.e., views more suitable to the task at hand).

In existing animation models (Strauss and Carey 1992; Koved and Wooten 1993; Elliott *et al.* 1994; Green and Halliday 1996; Hardisty *et al.* 2001), the user's control over the presentation of the animation is limited to the manipulation of the *animated space.*

This manipulation is accomplished by means of three basic operations. First, a user can slow down or speed up the evolution of the information by directing the *animated time* to evolve at a different pace from the *user time*. This operation allows the user to adjust the pace of the evolution of the information to be presented in a reasonable time frame (e.g., a geological phenomenon that took millions of years to evolve can be explored in a few minutes in a *user time* scale). Second, a user can change the temporal order of the animation by inverting the evolution of the *animated time* space. This operation allows a user to explore the environment where the evolution of the information is observed in the reverse chronological order (e.g., instead of seeing glaciers receding, seeing glaciers advancing). Third, a user can explore a single snapshot of the dynamic environment by stopping the evolution of the *animated space*. This operation allows the user to investigate the configuration of the environment at a certain instant in time. Other operations can be defined as a combination of these basic operations (e.g., by combining the first and second operation, a user can speed up the presentation of an animation in the reverse chronological order).

The manipulations of the *animated time* space are usually presented to a user through a graphical interface that mimics operations of a VCR control. The use of the VCR metaphor has a cognitive appeal, that is, it has a small number of operations and many users are very familiar with the semantic of this set of operations. The main disadvantage of the VCR-style of operations, however, is that the user is constrained to control the animation as a whole, without any means to address the behavior of an individual or group of objects. Moreover, these operations manipulate only the temporal space. The

VCR metaphor, for example, does not provide operations that change the temporal organization of the animation.

Consider, for example, a scenario where a user needs to explore the behavior of two phenomena that occur at different times (i.e., there is a temporal gap between the occurrence of these phenomena in *valid time*). In a typical animation environment, a user can start the animation with the behavior of the first object, wait a certain amount of time, and continue to explore the behavior of the second object. In order to minimize the temporal gap between these behaviors, a user can speed up the evolution of the animation. By doing so, however, all dynamic objects in the environment move faster, which it is not necessarily the best pace to explore the dynamic information. Using VCR-like operations, the user cannot generate an animation where one phenomenon follows the other with no temporal gap, or where both phenomena occur at the same time, facilitating the comparison of their behaviors.

The exploratory nature of GIS applications, requires an environment where the observer has full control over the content and flow of the animation (Fairbain *et al.* 2001; Kraak 2002). These requirements can be accomplished with a data model that captures different granularities of both the temporal space and of objects' behaviors. The first characteristic allows the manipulation of the temporal space associated with pieces of the animation. This characteristic gives a user a finer control over the flow of the animation. It is equivalent to having a VCR control attached to different components of the animation (e.g., part of an object behavior, the entire behavior of an object or the behavior of a group of objects). The second characteristic allows the modification of the temporal organization of pieces of the animation. This characteristic gives a user the ability to change the temporal arrangement of an object's behavior or of groups of objects' behaviors. The conceptualization of such a data model provides the means to move the user interface beyond the VCR metaphor and supports an exploratory analysis of dynamic geographic phenomena.

In the model for virtual exploration of animations the presentation of the temporal information is still done by mappings among *valid*, *animated*, and *user time* domains. In order to give a user finer control of the temporal space, however, a more elaborate conceptualization of the *animated time* domain is necessary. Thus, the *animated time* space is partitioned into a hierarchical representation of time (Figure 4.3). Each element in the structure works as a local temporal coordinate system for representations that embody the temporal characteristics of action's objects (e.g., an object carrying the temporal characteristic of an *Act* are represented over the *act time* space).

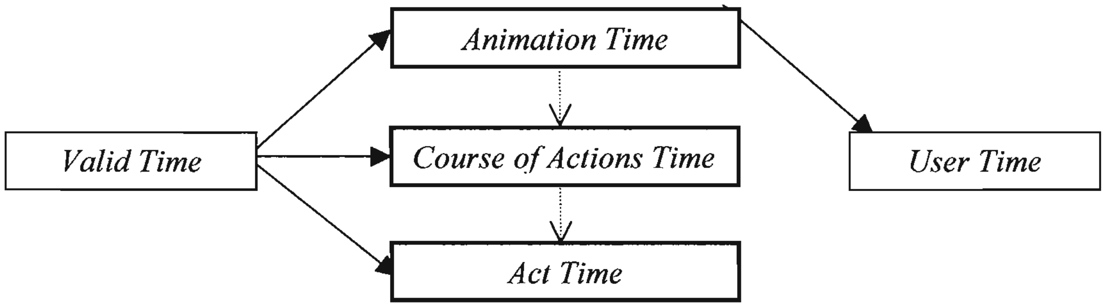


Figure 4.3. Mappings among valid times, different elements of the animated time domain, and user time.

In this setting, the process of mapping the information available in a database (*valid time*) to an application that presents the information to an observer (*user time*) becomes more elaborate. The mapping from *valid time* to *animated time* is spread among different elements of the *animated time* domain. Since *valid times* have all temporal information about an object's behavior, there is no simple abstraction in the model that incorporates all information coming from the *valid time* domain. The dotted arrows in the figure 4.3 represent a dependency among elements of the *animated time* domain. Thus, transformations of an element of the *animated time* space are consistently propagated for all elements in the hierarchy.

The next section discusses the data abstractions represented over each element of the structure of the *animated time* space. These abstractions capture temporal information about certain pieces of the dynamic environment. The specification of each data abstraction describes operations to combine these pieces of information and to manipulate their temporal characteristics. The specification describes also operations that modify the underlying temporal space of each representation (i.e., the elements of the structure of the *animated time* domain).

4.2 Structure of the Temporal Model

The conceptualization of the temporal model deals with abstractions that represent the temporal characteristics of action objects (Figure 4.4). This fact is modeled through a one-to-one association between classes of the temporal and action parts of the framework (i.e., each object of the action model has an associated representation in the temporal model).

The structure of the temporal model is similar to the structure of the action model (i.e., the temporal characteristic of entities representing a coarser granularity of objects' behaviors aggregates the temporal characteristics entities representing a finer granularity). The aggregation between classes of the temporal model, however, is redundant. This information can be retrieved through the association between entities of the temporal and action models and the same type of aggregation between classes of the action model. We replicate the aggregation in the temporal part of the framework to enhance the structural characteristic of the model and facilitate the model implementation.

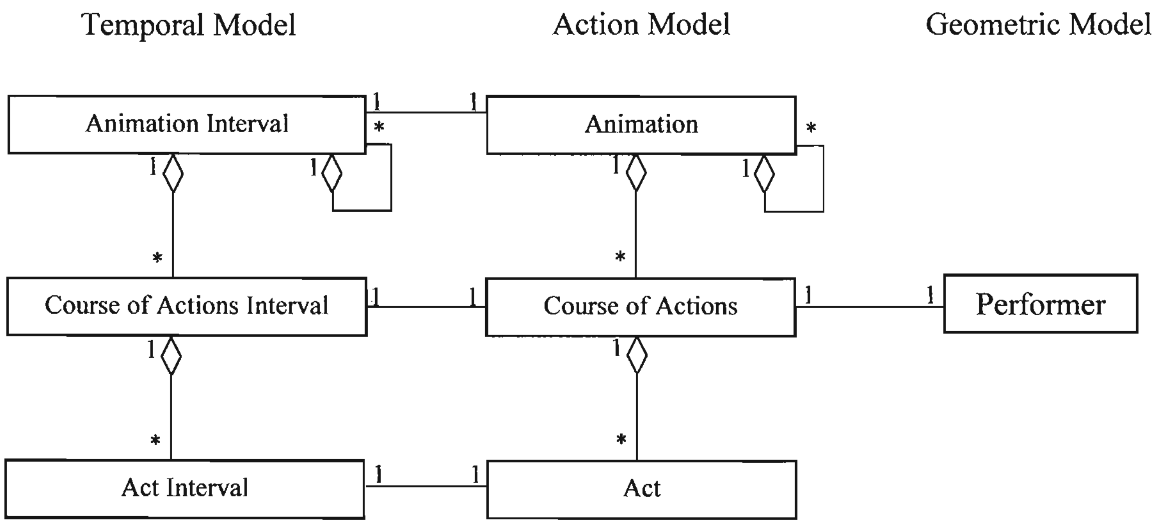


Figure 4.4. Class diagram of the dynamic, temporal, and geometric parts of the framework for virtual exploration of animations.

The next sections present the specification of each class of the temporal model (i.e., *Act Interval*, *Course of Actions Interval*, and *Animation Interval*). Operations of these classes build the hierarchical structure of the model and allow the manipulation of temporal characteristics of their instances. Each class of the temporal model encapsulates operations that simulate some kind of transformation of the underlying temporal space (i.e., an operation that inverts the *act time* space is part of the *Act Interval* class).

4.2.1 Act Interval

Act Interval refers to the temporal characteristic of acts. The *Act Interval* abstraction captures the duration of its associated *Act* object. Each *Act Interval* is defined over its own temporal space (i.e., *act time*). The temporal structure of the *act time* space is linear, discrete, and bounded at both ends. *Act Interval* is a finite subset of integers defined as $\{n \in Z \mid 0 \leq n \leq d\}$, where d is the duration of the act.

The specification of the *Act Interval* abstraction (Figure 4.5) shows a constructor method with two arguments (i.e., an *Act* object and a duration). The first argument (an *Act*) links objects of the temporal model with their associated representation in the action model. The second argument (an integer) represents the duration of the *Act* converted from *Valid Time* units to *Act Time* units (i.e., the duration of the act as stored in the database). The *Act Interval* specification does not provide any operation to designate the start point of the interval. This information is defined at a high level of abstraction in the model and will be discussed later in this chapter.

Other important temporal characteristics of the act are modeled through operations that simulate some kind of transformation of the underlying temporal space (i.e., the *act time* space). The operations `setDuration`, `setOrder`, `setFlow`, and `setPace` perform such transformations. These operations affect the way that an observer perceives the evolution of objects' states with the passage of time. Taken together, these operations direct the mechanism that links each state of an object's behavior with an instant in the *act time* domain. Changing the order an object's states are presented to the user, for example, simulate a transformation that inverts the *act time* space.

```
class ActInterval {

    Overview: represents the temporal characteristics of Acts objects.

    // Constructors
        ActInterval (Act a, int duration)
        // Effects: initializes this with the duration of its associated Act.

    // Methods
        void setDuration  (int duration)
        void setOrder (String o)
        void setFlow (String f)
        void setPace (String p)
```

Figure 4.5. Specification of the class Act Interval.

In order to understand the effect that each transformation operation has in the way that a user perceives the evolution of the act, it is important to review the mechanism used to generate states of an object with the passage of time introduced in chapter 3. Consider, for example, the act of a car traveling in a straight line between two locations (Figure 4.6a). This *Act* is modeled with three key states (i.e., the initial, intermediate, and final position of the car). Although the intermediate state of the car is redundant for this act, we introduce it here to enhance the understanding of some concepts discussed in this section.

Each key state of an object *Act* is associated with a *normalized time* value. A *normalized time* is a real number in the interval [0,1]. The lower and upper bound of the *normalized time* interval are associated with the state of an object at the beginning and end of the act, respectively. An *Interpolator* object takes into account key states of the movement, and, upon request, returns the state associated with a given *normalized time*

(Figure 4.6b). The continuous arrow in the figure represents an incoming message to the *Interpolator* object with a *normalized time* value of 0.25 as argument. The dashed arrow represents an outgoing message carrying out the result of the computation.

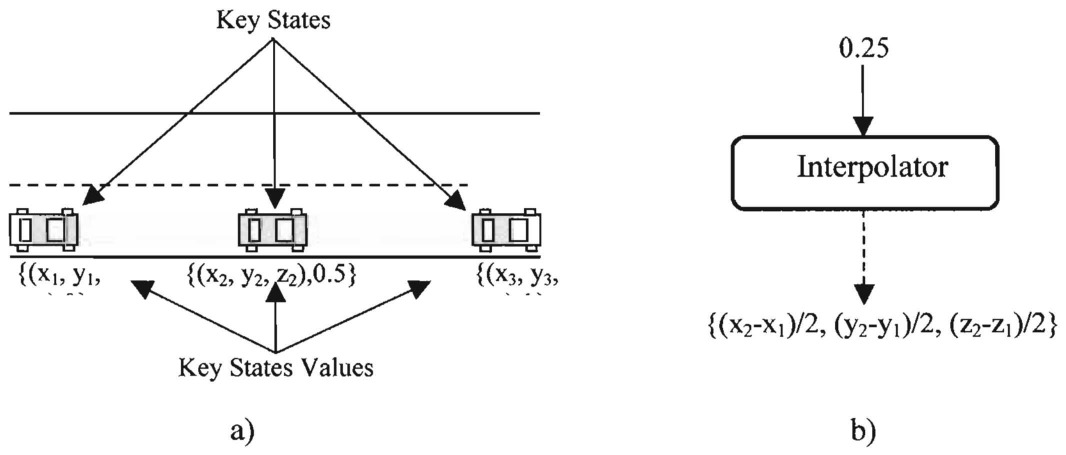


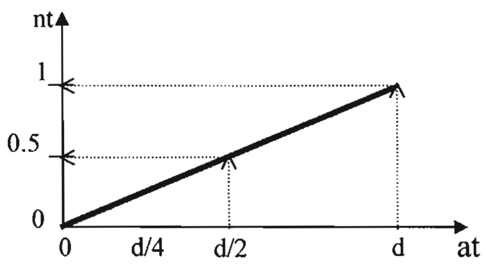
Figure 4.6. An act of a car moving in a straight line: a) Key states of the position of the car and b) an interpolator object computes the position of a car for a given normalized time value.

Interpolator objects accept only *normalized time* values as input. In order to evaluate the state of an object at a certain instant in time, it is necessary to map instants in the *act time* space to values in the *normalized time* space. A *normalize* function performs such a mapping. The domain of this function is *act time* values (i.e., values in the interval [0, duration]) and its range is *normalized time* values (i.e., a value in the interval [0, 1]). Based on the output of the *normalize* function, the object *Interpolator* returns the respective state of the object (i.e., the car's position) at the instant used as argument.

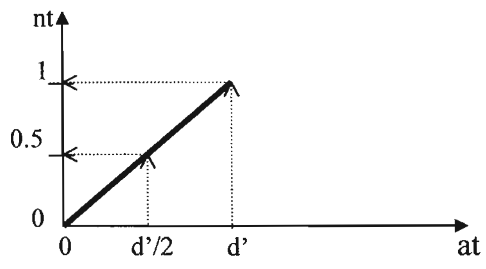
The combination of transformation operations (`setDuration`, `setOrder`, `setFlow`, and `setPace`), depending on the value of their arguments, produces different shapes of the *normalize* function (Figure 4.7). The semantics of each operation and the effect that each kind of argument has on the shapes of the *normalize* function follow.

- The `setDuration` operation changes the duration of *Act Intervals*. This operation has as an argument the new duration of the act. Changing the duration of the *Act Interval* slows down or speed up the movement of an object.
- The operation `setOrder` defines the temporal order of the presentation of the act. This operation has two kinds of arguments: *reverse* and *normal*. The *reverse* argument, for example, causes the movement of the car to be perceived in the reverse chronological order, while the argument *normal* preserves the modeled evolution of the car.
- The operation `setFlow` alternates the shape of the *normalize* function between a constant function and a non-constant function. The `setFlow` operation has two kinds of arguments: *stepwise* and *continuous*. The argument *stepwise* constrains the range of the *normalize* function to *normalized time* values associated with key states of the act. In this way, only the discrete movement of an object's act is presented. The *stepwise* argument is used to highlight key states of an object behavior. The argument *continuous* indicates that the range of the *normalized* function is any value between 0 and 1. In this way, any increment in the *act time* space produces a different state from the previous instant of observation. It causes the illusion of the car moving continuously between two locations.
- The operation `setPace` alternates the shape of the *normalize* function between a linear and a non-linear function. This operation has five kinds of argument:

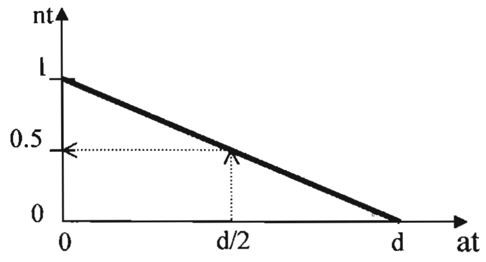
constant_speed, *accelerated*, *decelerated*, *accelerated_decelerated*, and *decelerated_accelerated*. The argument *constant_speed* defines a linear *normalize* function. Such a function produces an animation where the car is seen moving at a constant speed along the entire act. The arguments *accelerated*, *decelerated*, *accelerated_decelerated*, and *decelerated_accelerated* define different shapes of non-linear *normalize* functions. A non-linear *normalize* function implies that different unitary increment in the *act time* space represents a different increment in the *normalized time* space. Non-linear *normalize* functions produce an animation where objects' states are perceived changing at different speeds during the act. In the case of the car, the car is seen moving at variable speeds (e.g., accelerating or decelerating). The argument *accelerated* defines a non-linear function where an unitary increment of the act time space cause a small increment in the *normalized time* space in the beginning of the act and a large increment in the *normalized time* space near the end of the act. Such a function causes the illusion of the car moving initially at a low speed and gradually increasing its speed with the passage of time. The argument *decelerated* defines a function that produces the opposite effect. The argument *accelerated_decelerated* produces an accelerated movement in the first half of the act and a decelerated movement on the second half of the act. The opposite effect is accomplished through the *decelerated_accelerated* argument.



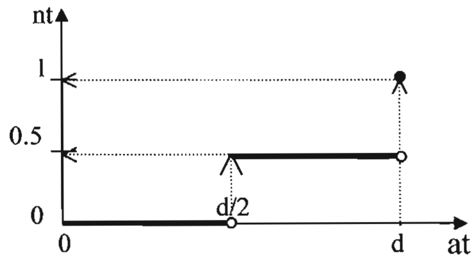
setDuration(*d*) setOrder(*normal*)
setFlow(*continuous*) setPace(constant)



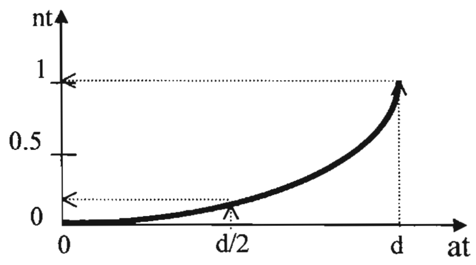
setDuration(*d'*) setOrder(*normal*)
setFlow(*continuous*) setPace(constant)



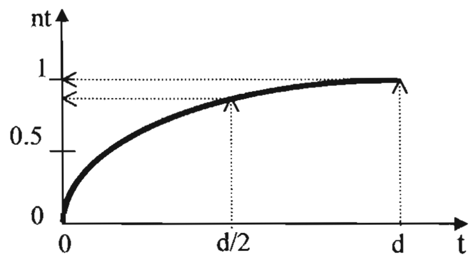
setDuration(*d*) **setOrder(*reverse*)**
setFlow(*continuous*) setPace(constant)



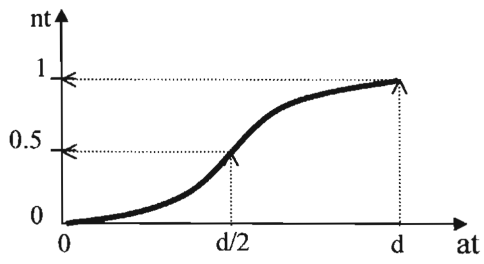
setDuration(*d*) setOrder(*normal*)
setFlow(*stepwise*) setPace(constant)



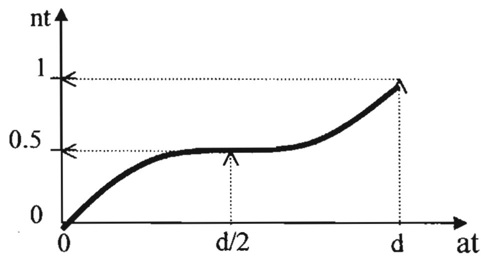
setDuration(*d*) setOrder(*normal*)
setFlow(*continuous*) **setPace(*accelerate*)**



setDuration(*d*) setOrder(*normal*)
setFlow(*continuous*) **setPace(*decelerate*)**



setDuration(*d*) setOrder(*normal*)
setFlow(*continuous*) **setPace(*acc/dec*)**



setDuration(*d*) setOrder(*normal*)
setFlow(*continuous*) **setPace(*dec/acc*)**

Figure 4.7. Different shapes the normalize function.

Transformation operations are high-level abstractions used to define the shape of the *normalize* function. In a GIS context, these operations simulate the most usual manipulations of the temporal space that can be accomplished by a user at the act level of granularity. If an application of a specific domain needs even more control of the temporal space, however, the set of transformation operations or their arguments can be extended to incorporate such requirements.

4.2.2 Course of Actions Interval

Course of Actions Interval represents the temporal characteristics of *Course of Actions*. Each *Course of Actions Interval* is defined over its own temporal space (i.e., *course of actions time*). The temporal structure of this temporal space is linear, discrete, and unbounded at both ends (i.e., isomorphic with the set of integers ordered by the "less than" relation).

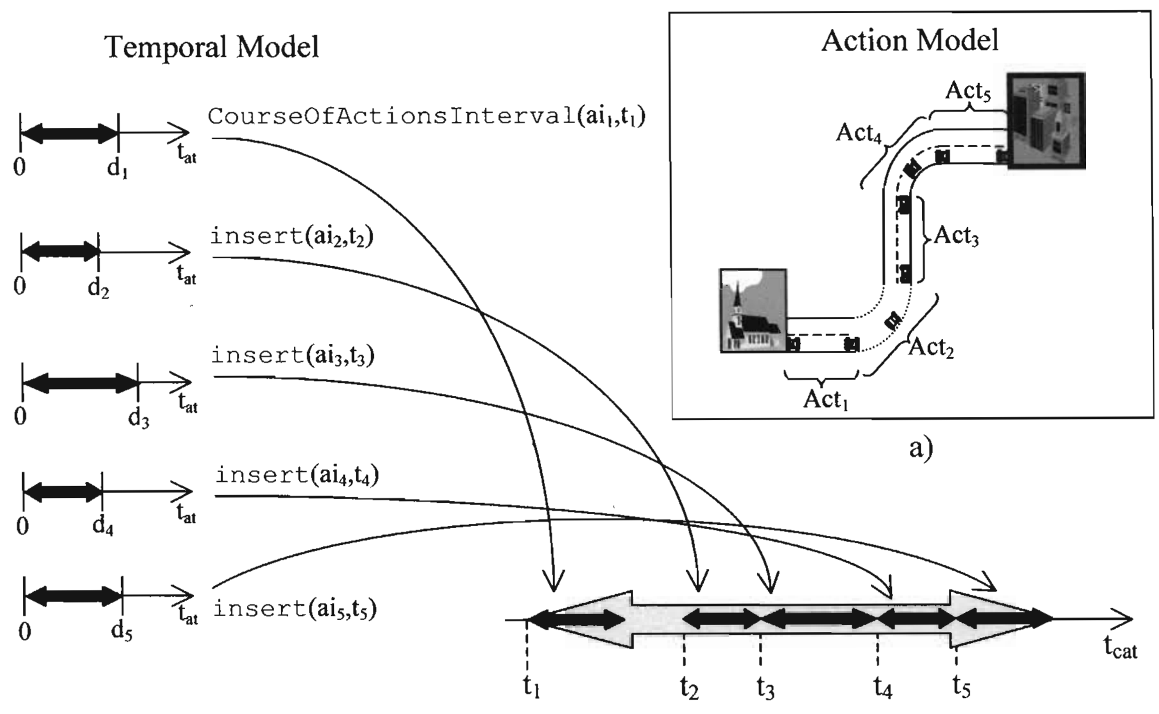
Course of Actions Intervals are modeled as a collection of *Act Intervals* positioned along the *course of actions* timeline. Instances of this class are built in the same fashion as their associated objects of the action part of the framework. A constructor method instantiates a *Course of Actions Interval* and adds the first *Act Interval* in the collection (Figure 4.8). The constructor also specifies the start point of such an interval. This argument (an integer) represents the start point of the *Act* converted from *valid time* units to *course of actions time* units (i.e., the modeled start point of the *Act* as stored in the database). *Acts Intervals* are positioned along the *course of actions* timeline by mapping the origin of the *act time* space to a point in the *course of action* space. This point corresponds to the start point of the act. The process of building the *Course of Actions*

Interval continues with the method `insert`. This method is similar to the constructor method in the sense that it adds an *Act Interval* to the collection and defines the *Act Interval* start point in the *course of actions time* space.

```
class CourseOfActionsInterval {

// Overview: represents the temporal characteristics of Course of Actions objects.

        // Constructors
        CourseOfActionsInterval (ActInterval ai, int start)
        // Effects: initializes this with the first act interval in the set and specifies the
        // start point of the interval in the course of actions temporal space.

        // Methods
        void insert (ActInterval ai, int start)
        // Effects: inserts an act interval in the collection and specifies the start point
        // of the interval in the course of actions temporal space.
```

Figure 4.8. Specification of the class Course of Actions Interval.

In order to illustrate the process of building the temporal characteristic of an object's behavior consider, for example, the *Course of Actions* of a car traveling between two locations (Figure 4.9a). The entire behavior of the car is modeled with five *Acts*. Each *Act* object has an associated *Act Interval* in the temporal model. Consider, also, that the car completely stops for a few seconds before traveling the unpaved bend of the road. Since the lack of activity is not modeled as an act of the object, this information does not have an associated representation on the action model. The *Course of Actions Interval* representing such a behavior is instantiated with the first *Act Interval* of the car and complemented with a sequence of calls to the `insert` method with other *Act Intervals* of the object's behavior. The resulting *Course of Action Interval* can be seen as an aggregation of *Act Intervals* spanned over the course of actions timeline (Figure 4.9b).



b)

Figure 4.9. Representations of the course of actions of a car moving between two locations: a) Entities of the action model and b) entities of the temporal model.

The graphical representation of the *Course of Actions Interval* in the Figure 4.10b is depicted at two different levels of granularity. At the finest level of granularity, the *Courses of Actions Interval* is seen as a collection of convex intervals (i.e., *Act Intervals*). At the coarsest level of granularity the *Course of Actions Interval* is represented by the smallest convex interval that encompasses all associated *Act Interval* in the collection. In this way, at a high level of abstraction periods of inactivity are incorporated in the graphical representation of the *Course of Actions Interval.*

Act Intervals are congruent with the intuitive notion of intervals, a duration of time associated with some event occurring in the world (Allen and Ferguson 1997). *Course of Actions Intervals*, on the other hand, do not fit well with this notion. We can have a

Course of Actions Interval composed of *Acts Intervals*, which do not meet or overlap, implying a period of inactivity inside the *Course of Actions Interval*. The advantage of reducing such entities to a simple interval is that operations at different levels of abstractions become very similar (i.e., these operations have the same name and a convex temporal interval as argument). This characteristic enhances the understanding of the model and facilitates the user's assimilation of the functionalities of different versions of many operations that are used throughout the model at different levels of granularities.

Due to the fact that *Course of Actions Intervals* represent the temporal characteristics of the entire behavior of an object, the set of operations used to manipulate their instances increases in number and in complexity. A set of transformation operations similar to operations performed at the *Act Interval* level of granularity is used to simulate transformations of the *course of actions time* space. Other operations allow a finer control over an object's behavior by changing the temporal organization of its constituent parts (i.e., their associated *Act Intervals*). The later group of operations is introduced in the context of temporal constraints, discussed in chapter 5.

4.2.2.1 Transformation Operations

A set of transformation operations modifies the underlying temporal space of *Course of Actions Interval.* At the *Course of Actions Interval* level of abstraction, the set of transformation operations consist only of the operations `setDuration`, `setFlow`, and `setOrder`. The operation `setPace` is not available at this level of abstraction. The acceleration/deceleration of an object behavior can be specified only for individual pieces of the movement (i.e., at the *Act Interval* level of abstraction). This constraint is due to the difficulty of identifying the precise semantics of the `setPace` operation when this operation is performed over an abstraction that represents the entire behavior of an object. The behavior of an object can be composed of multiple periods of acceleration, deceleration, and constants speeds, which is hard to model with a single argument.

Syntactically, transformation operations at the *Course of Actions* level of abstraction are identical to their version performed over a finer granularity (i.e., they have the same name and accept the same kinds of arguments). Semantically, these operations are similar, but with more elaborate functionalities. Intuitively, the operation `setDuration` changes the duration of the *Course of Actions Interval*, the operation `setFlow` alternates between continuous and stepwise presentation of the behavior, and the operation `setOrder` changes the temporal order of an evolution of the object's behavior. The hierarchical structure of the *animated time* space, however, imposes that the manipulation of the temporal space at a certain level of granularity be consistently propagated to its associated temporal spaces at a lower level of granularity. In this way, manipulations of the *course of action time* space affect all *Acts Intervals* associated with the *Course of Actions Interval.*

In order to understand the semantics of each transformation operation consider, for example, the *Course of Actions Interval* representing the behavior of the previous example of the car moving between two locations (Figure 4.10a). Consider yet, that a user wants to create a new view of the environment where the car is seen performing its modeled behavior twice as fast. In this way, the user can reduce the duration of the *Course of Actions Interval* to half of its original duration (Figure 4.10b).

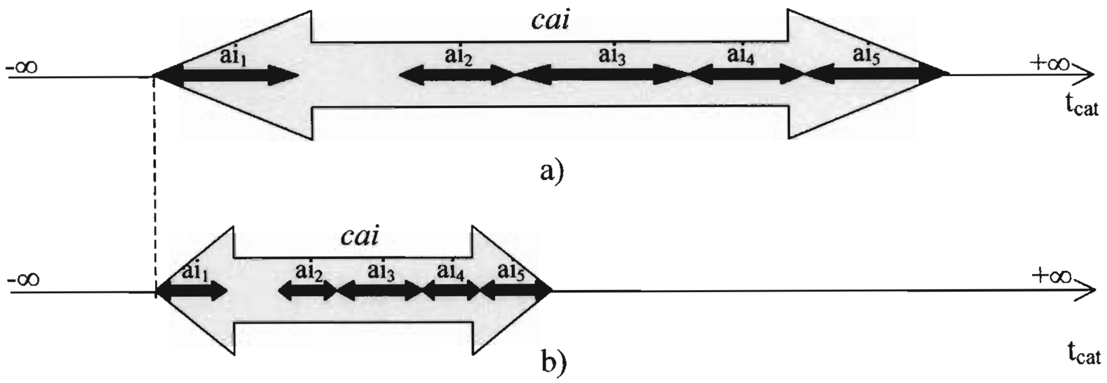


Figure 4.10. Graphical representations of two versions of a car's movement: a) the modeled configuration of the car and b) the effect of changing the duration of the trip.

Changing the duration of an object *Course of Actions* is equivalent to performing a scale operation over *Course of Actions Interval* with respect to the start point of the interval. Thus, the start point of the *Course of Actions Interval* remains the same and its duration is multiplied by a scale factor. The ratio between the previous and the new duration defines the scale factor. A scale operation with the same scale factor and same reference point (i.e., the start point of the *Course of Actions Interval*) is performed over every *Act Interval* in the collection. It means that not only are the durations of *Act Intervals* affected, but also all Act Intervals' start point that do not coincide with the reference point of the scale operation.

The `setOrder` operation allows the user to produce an animation where a certain object is seen performing its behavior in a reverse chronological order while the other objects are seen performing their behaviors in the chronological order. If a user wants to produce such animation with the car of the previous example, a `setOrder` operation with a *reverse* argument performed over the car temporal characteristics can handle the task. Figure 4.11a shows the graphical representation of the modeled behavior of the car while the Figure 4.11b depicts the graphical representation of the car moving backwards.

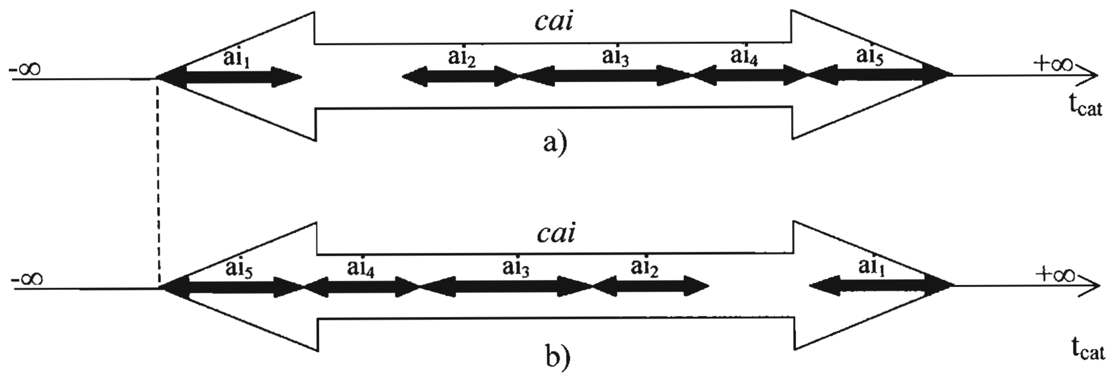


Figure 4.11. Graphical representations of the operation setOrder over the entire behavior of the car: a) the modeled configuration of the car's behavior and b) the effect of inverting the evolution of the entire behavior of the car.

The `setOrder` operation does not change the modeled duration or start point of the *Course of Actions Interval.* At a finer level of granularity, however, the start points of the *Act Intervals* are affected. Graphically, the `setOrder` operation with a *reverse* argument is equivalent to perform a mirror transformation with respect to the midpoint of the *Course of Actions Interval.* A mirror transformation over temporal intervals changes the endpoints of the interval. The new position of these points are the ones in which the distance from the new position to the reference point is the same as from the old position to the reference point. Thus, at a coarse granularity, the *Course of Actions Interval* remains the same (i.e., the new position of the interval end point coincides with the old position of the interval start point). At a finer granularity, however, the start points of all *Act Intervals* in the collection change. The process of updating the start points of *Acts Intervals* is accomplished by mirroring each *Act Interval* with respect to the midpoint of the *Course of Actions Interval* (Figure 4.12b). In addition to that, a `setOrder` operation with a *reverse* argument is performed over every *Act Interval* in the collection.

The `setFlow` operation allows the user to select between a continuous evolution of the movement of the car or an animation where only representative states of the car's position are shown. The `setFlow` operation performed over a *Course of Actions Interval* is equivalent to performing the same version of this operation over all *Act Intervals* in the collection. In this case, the duration and start point at all temporal intervals at both levels of granularity remain the same. Thus, the graphical representation of the outcome of this operation is similar to the original configuration of the *Course of Actions Interval* at both levels of granularity.

4.2.2.2 Cyclic Behaviors

One important requirement of an animation model is that its data abstractions need to capture the temporal structure of the application domain. Failing to satisfy this requirement means that information coming from the *valid time* domain is only approximate or gets lost when mapped onto the *animated time* domain. The conceptualization of different granularities of objects' behaviors position along a timeline and powered with operations to modify their temporal organization is not robust enough to capture the richness of temporal geographic information. These abstractions do not capture, for example, objects' behaviors that repeat themselves in a cyclic fashion.

Current animation models do not properly capture cyclic behaviors. In these models cycles are simulated through a mechanism that repeats portions of the animation a certain number of times or indefinitely. Moreover, in these models the behavior of the object represents the entire period of the cycle. It means that there is no interval of inactivity between two occurrences of the cycle. In order to support more sophisticate kinds of cyclic behaviors, we extend the animation model with a new class called *Cycle Interval.* This class is modeled as a specialization of the class *Course of Action Interval* (Figure 4.12). The temporal characteristics of cyclic behaviors are represented by instances of the class *Cycle Interval.* At this point we support only cyclic behaviors of the entire behavior of an object (i.e., *Course of Actions*). We leave for future extensions of this model the support of cycles at the *Act* level of abstraction.

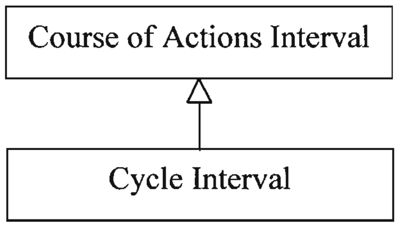


Figure 4.12. Class diagram showing a Cycle Interval as a specialization of the class Course of Actions Interval.

In order to understand the structural elements of objects that represent cycles, consider, for example, the spray of hot water from a geyser. The water is sprayed out from the ground in a cyclic pattern (Figure 4.13). In the action domain, there is no abstraction that specifically represents different occurrences of the cycle. Every occurrence of the cycle is modeled through a single *Course of Actions* that repeats itself accordingly to the temporal characteristics of the *Cycle Interval* object. In this example, the object spillage represents the *Course of Actions* of the geyser.

In the temporal domain each hot spring is represented by a single interval (i.e., the temporal extent of one occurrence of the geyser's activity). We call this interval the *period of activity*. In this cyclic phenomenon, the time necessary to warm up the underground water to a certain temperature and cause the next spray of the geyser is represented by an interval under the relation *meets* with the *period of activity*. We call this interval the *period of inactivity*. The period of inactivity represents the period of time when the associated action is not happening, or that what is happening is not modeled in the action domain. Thus, there is no action associated with the *period of inactivity* of the geyser. The interval of inactivity represents the temporal extent between two consecutive occurrences of the cyclic behavior. The set union of the *interval of activity* and the *interval of inactivity* defines the *period of the cycle*.

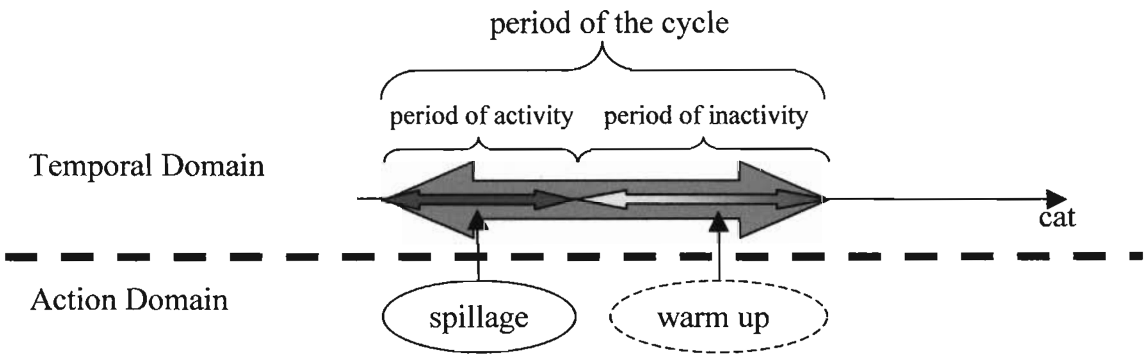


Figure 4.13. An interval of activity followed by an interval of inactivity representing the temporal extent of a single occurrence of the cycle of a geyser.

Similar to the *Course of Actions Interval* representation, a *Cycle Interval* can be depicted at different levels of granularity. At a fine level of granularity, a *Cycle Interval* object is represented by two intervals (i.e., the periods of activity and inactivity). At a coarse level of granularity, a *Cycle Interval* is represented by a single interval (i.e., the period of the cycle). The later representation encompasses the representation of both intervals at the fine level of granularity.

The class *Cycle Interval* inherits from its super-class *Course of Actions Interval* all structural elements related with the cycle's period of activity (i.e., the associated action and the start point and duration of the period of activity). The class *Cycle Interval* complements the structure of the cycle by adding an attribute that defines the duration of the period of inactivity of the cycle (Figure 4.14). Other structural elements of the *Cycle Interval* (e.g., duration of the period of the cycle) can be easily derived from the existing structural elements (Cuckierman and Delgrande 2000; Terenziani 2003).

```
class CycleInterval extends CourseOfActionsInterval {

   Overview: represents the temporal characteristics of cyclic behaviors.
   // Attributes
       int dpi
   ...
   // Constructors
       ...
   // Methods
       int setDurationPeriodInactivity (int dpi)
       ...
```

Figure 4.14. Specification of the class Cycle Interval.

The structural elements of the class *Cycle Interval* capture the temporal characteristic of a single occurrence of the cycle. We call this occurrence the *occurrence of reference*. Other occurrences of the cycle can be retrieved based on *occurrence of reference*.

In order to represent other occurrences of the cyclic behavior we need to define the pattern of the repetition of the cycle. This information is modeled through the operation `setPatternsOfRepetitions`. This operation has four kinds of arguments: *infinite*, *finite*, *until*, and *from*. The semantic associated with each kind of argument determines the temporal characteristics of all occurrences of the cycle and which instance of the cycle the *occurrence of reference* refers to. The semantics of each argument follows:

- The argument *infinite* represents endless cyclic behaviors (Figure 4.15a). This type of cycle implies that the object is always observed performing some activity. The *occurrence of reference* of this cycle can be any occurrence of the cyclic behavior.
- The argument *finite* represents behaviors that repeat themselves a certain number of times after the *occurrence of reference* (Figure 4.15b). The number of cycles needs to be provided in this case. An operation `setNumberOfRepetions` informs the number of times that the cycle repeats.
- The argument *from* models cycles that start with the *occurrence of reference* and repeats the associated action indefinitely (Figure 4.15c).
- The argument *until* models cycles that finish with the *occurrence of reference* but the object is observed performing its behavior at every instant before that (Figure 4.15d).

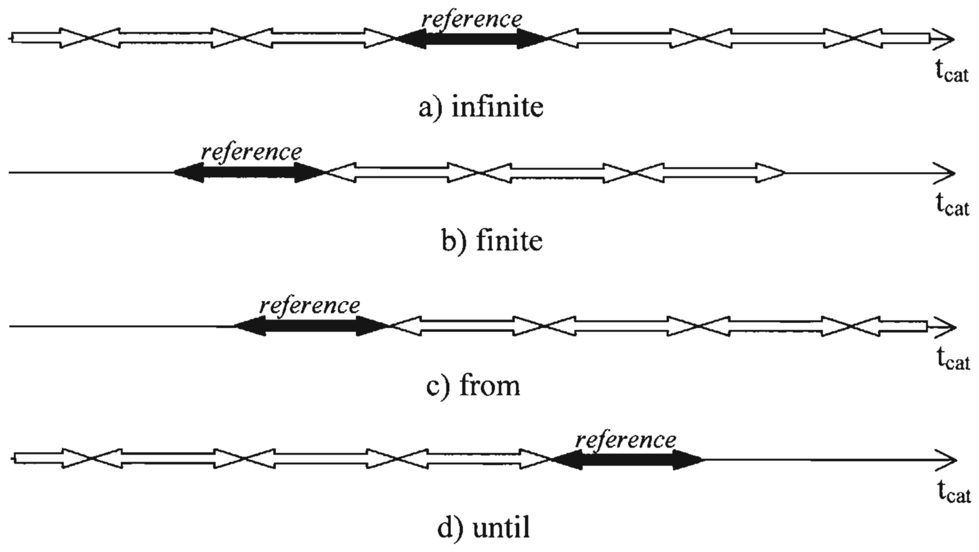


Figure 4.15. Different patterns of repetitions of cyclic behaviors.

It is important to note that the representation of the cycle' occurrences in the Figure 4.16 is depicted using an interval representing the period of the cycle (i.e., an interval that encompasses the periods of activity and inactivity of the cycle).

In this section we treat only the extended functionality and semantics of the class *Cycle Interval*. This class inherits all operations of its super-class (i.e., *Course of Actions Interval*). Some operations, however, are overriden to accommodate specificities of the cycle's representation. Other temporal operations involving cycles are introduced in chapter 5.

4.2.3 Animation Interval

The main purpose of an *Animation* object in the action domain is to serve as a grouping mechanism. *Courses of Actions* are combined to form *Animations* and *Animations* are combined forming complex *Animations*. *Animation Intervals* represents the temporal characteristics of these objects.

Each *Animation Interval* is defined over its own temporal space (i.e., *animation time*). The temporal structure of *animation time* is identical to the temporal structure of *course of actions time* (i.e., linear, discrete, and unbounded at both end).

Animation Intervals are similar to *Course of Actions Intervals* in the sense that they are modeled as a collection of intervals positioned along a temporal axis. Thus, the process of building *Animation Intervals* is accomplished by adding temporal intervals to a collection of intervals. Different from *Course of Actions Intervals*, however, the elements in the collection of intervals can be of different sorts (i.e., the elements can be either *Animations Intervals* or *Course of Actions Intervals*). This characteristic is represented in the specification of the *Animation Interval*, for example, by two versions of the method insert, one for each sort of object allowed in the collection (Figure 4.16).

```
class AnimationsInterval {

    Overview: represents the temporal characteristics of animations objects.

    // Constructors
    // Methods
        void insert (AnimationsInterval ai)
        void insert (CourseOfActionsInterval cai)
```

Figure 4.16. Specification of the class Animation Interval.

From the users perspective an *Animation Interval* is the smallest convex interval that encompasses all temporal intervals in its collection. Figure 4.17 shows a graphical representation of *Animation Interval* composed of one *Animation Interval* and two *Courses of Actions Intervals*.

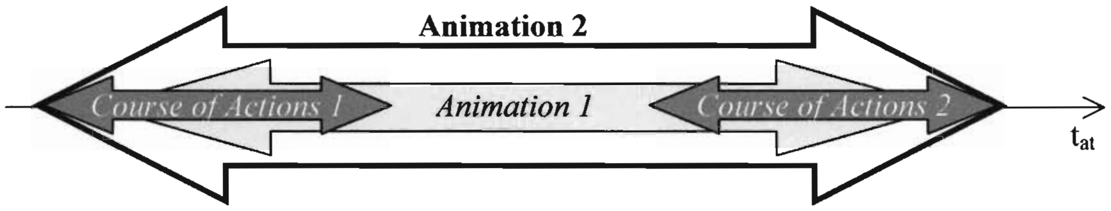


Figure 4.17. Graphical representation of Animation Intervals.

At the *Animation Interval* level of abstraction, a set of transformation operations (i.e., `setDuration`, `setOrder`, and `setFlow`) simulates some kind of transformation of the *animation time* space. These operations allow a user to manipulate the behavior of a group of objects or all objects in the environment. The semantics and syntax of transformation operations over *Animation Intervals* are similar to the same set of operations at the *Course of Actions Interval* level of abstraction and they are not repeated here.

Users are not restricted to manipulate *Animations Intervals* only by simulating transformation of the underlying temporal space. At the *Animation Interval* level of abstraction, a group of operations that resembles operations over sets (i.e., intersection, difference, and union) can also be used to produce a new view of the dynamic environment. We call this group of operations *combinations operations*.

- The `intersection` of two animations generates an animation defined over an interval obtained from the intersection of the intervals used as the arguments. The result is an animation defined over an interval when only the simultaneous occurrences of both animations will be presented. With this operation a user can verify, for instance, mutual interference between the movements of different objects.
- The `difference` of two animations generates an animation defined over an interval when the second animation is not defined (i.e., an interval when only the first animation occurs). This operation permits the user to isolate the behavior of the first object during the period when the second object does not occur.
- The `union` of two animations is a simple combination of two animations while preserving the temporal configuration of individual arguments. This operation is useful for combining animations to form complex animations.

Despite the modeled configuration of the dynamic environment (i.e., the number and configuration of all *Animations*, *Course of Actions*, and *Act* objects), the model for virtual exploration of animation creates a special *Animation* object that combine all action objects in the environment. We named this special object *All Animations*. The temporal representation of the entity *All Animations* is an interval that spans from the instant that the first object starts to perform some action until the last movement in the environment.

Operations performed over the object *All Animations* affect all objects uniformly. In this way, an operation `setDuration`, for example, allows the user to specify the desired duration of the entire animation, while preserving the relative duration of all actions in respect to each other. There are some operations, however, that are applied only over the object *All Animations*. We call this group of operations *presentation* operations.

The group of *presentation* operations is formed by the operations `start`, `play`, and `stop`. These operations direct the flow of information coming from the *animated time* to the *user time* domain. The operation `start` aligns a point in the animation timeline with the user present. This operation has the effect of defining the point of insertion of the user in the animated temporal domain (i.e., this operation defines the start point of the animation). The operation `play` performs a continuous mapping from the *animated time* domain to the *user time* domain, allowing the user to sense the evolution of the modeled dynamic environment (i.e., allowing the user to explore the animation). The operation `stop` suspends the flow of information coming from the *animated time* domain, which allows a user to observe a static version of the environment.

4.3 Summary

This chapter discussed the conceptual model of the temporal part of the framework of virtual exploration of animations. A structure of classes representing the temporal characteristics of actions was presented and attributes and methods of these classes were discussed in an informal way.

The conceptualization of the action and temporal parts of the animation model framework gives a user a cognitively plausible representation of the dynamic environment at different levels of granularity of objects' behaviors. A user can, through operations associated with each level of granularity, create new views of the dynamic environment. This process is accomplished by manipulating the object's underlying temporal space through *transformation operations* or by combining objects *Animations* through *combination operations*. These operations, however, do not allow a user to change the temporal arrangement of the dynamic objects in the environment. This functionality is accomplished through operations called *compositions* operations.

Composition operations allow a user to change the animation by rearranging the temporal characteristics of the object's behaviors at different levels of granularity (e.g., changing the start point of an *Act Interval* or the end point of a *Course of Actions Interval*). Giving a user the possibility of manipulating this kind of information, however, allows a user to create views of the environment that make no sense in the context of the application domain (e.g., a cause happening after the effect). In this way a mechanism to avoid "unrealistic" views of the dynamic environment and keep the modified version of the animation coherent with some requirements of the application domain is necessary.

The next chapter introduces *composition operations* and the mechanism used to represent *temporal constraints* among entities of the temporal part of the animation framework.

CHAPTER 5

TEMPORAL CONSTRAINTS

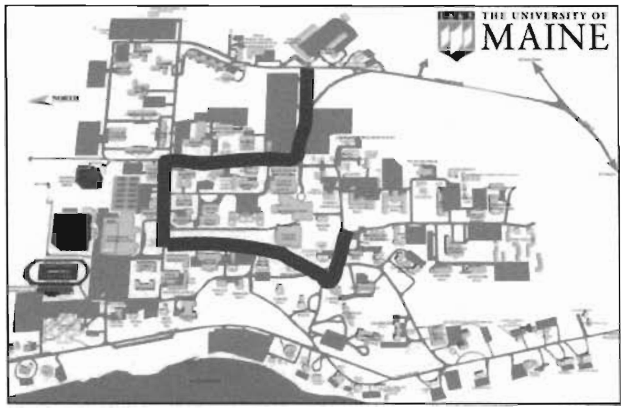
The basic unit of time in the animation model is an interval. Intervals are ubiquitous in the model and used to represent the temporal extent of objects' behaviors at different levels of granularity. Abstractions of the animation model, however, does not capture any qualitative relationship among the intervals, which is a valuable piece of information to represent constraints among temporal entities and treat temporal information that is available only as the relative order between objects' behavior (Frank 1998).

This chapter introduces a mechanism to incorporate into the animation model qualitative information about the organization of the temporal characteristics of objects' behavior. This information comes from the knowledge base of the application domain and it is represented in the model through temporal constraints among intervals. The next section discusses the role of representing such temporal relationships among entities of model. Subsequently, the mechanism to represent temporal constraints is discussed in the context of temporal relationships between two intervals and among a sequence of intervals of cyclic behaviors.

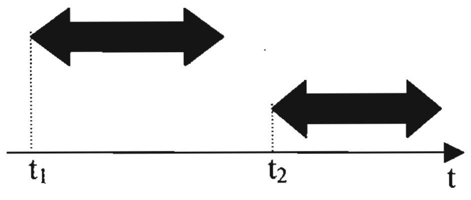
5.1 The Role of Representing Relationships between Temporal Intervals

In existing animation models (Fiume *et al.* 1987; Dollner and Hinrichs 1997), the "knowledge" about the temporal organization of the animation's components is used mainly for performance purposes. Considering the order in which behaviors occur, for example, these models can perform a temporal indexation of objects' behavior and optimize the information sent to the graphic engine. This "knowledge", however, plays a significant role if the application allows the user to manipulate the temporal configuration of the elements that form the animation. In an animation model that supports the manipulation of the temporal characteristics of objects' behavior, it is important to "know" if there exists any temporal relationship among their constituent parts. Thus, the modification of the temporal characteristic of a single interval allow the animation model to modify the temporal characteristics of all related intervals in a way that preserves some "known" characteristics of the modeled animation.

Consider, for example, the animation of two vehicles from the University of Maine's maintenance department. These vehicles travel between two locations in response to two different requests. The blue and red lines in the Figures 5.1a represent the spatial characteristics of such vehicles' behavior (i.e., the path of each vehicle on the campus). The temporal characteristics of the vehicles' behavior (i.e., the instant when they start to move and the duration of the trip) are represented as temporal intervals positioned along a timeline (Figure 5.1b).



a)



b)

Figure 5.1. Graphical representations of the spatial and temporal characteristics of the movement of two vehicles on the campus: a) the paths of the vehicles and b) the temporal characteristics of each movement.

Analyzing the spatial and temporal characteristics of the vehicles' behavior, an observer can infer that each trip corresponds to a request from different locations on the campus and that these requests are responded in a timely manner (i.e., they are responded at different periods). If the animation model provides the means to an observer to change the temporal characteristics of the vehicles' behavior, an observer can produce, for example, an animation in which the trip of the second vehicle occurs before the trip of the first vehicle or an animation in which both vehicles are traveling at the same time. Since there is no information in the model relating the behaviors of the vehicles, the progression of the animation in both cases is obvious. If there exists any relationships between these behaviors, however, the progression of the animation may vary.

Consider, for example, that is known from the application domain that the trip of the first vehicle (the red interval) is due to a request from the President's office and the trip of the second vehicle (the blue interval) is due to a request from the library. Due to the nature of the problem of each request (i.e., the library has a problem with the roof and the President's office has a problem in the sewer), the vehicle used in each response is different. Consider yet that requests from the President's office have a higher priority than requests from other departments of the University and that the same employee responds to both requests. This information can be represented in the animation model through a relationship that captures that the red interval always occurs before the blue interval (Figure 5.2). This relationship models the fact that a high priority is given to the President office's requests as well as the fact that a person cannot drive the same vehicle at the same time.

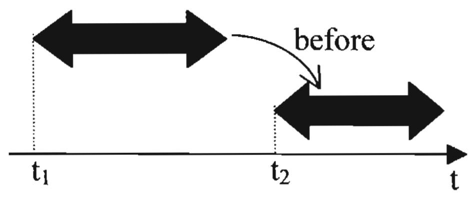


Figure 5.2. A relationship between two intervals representing some knowledge from the application domain.

In this scenario, an observer can change the temporal characteristic of the interval that represents the response to the President's office (the red interval) and make it start at the instant when the blue interval starts. Since there is a temporal relationship between these intervals, two possible temporal configurations of the animation are possible. First, the application can redefine the start point of the blue interval in a way that the original temporal relationship between the intervals still holds (Figure 5.3a). In this case, the modified version of the animation is similar to the original version and differs only by the fact that the maintenance department starts to respond the requests later. The rationale used to update the temporal characteristics of the animation when a temporal relationship is in place is discussed in chapter 6 together with the operations that allow an observer to manipulate the animation. Second, the application can ignore the temporal relationship between the intervals and keep the original configuration of the blue interval (Figure 5.3b). In this case, the modified version of the animation shows both vehicles traveling at the same time. Since the modified version of the animation violates some knowledge from the application domain, the animation model responds to this fact by changing the way in which the object interacts with other objects in the environment (i.e., the animation reflects the fact that the same person is driving two vehicles at the same time). The rationale used to modify the way that objects interact with other objects is discussed in chapter 7 together with the mechanism used to represent the semantics of the objects in VR environments.

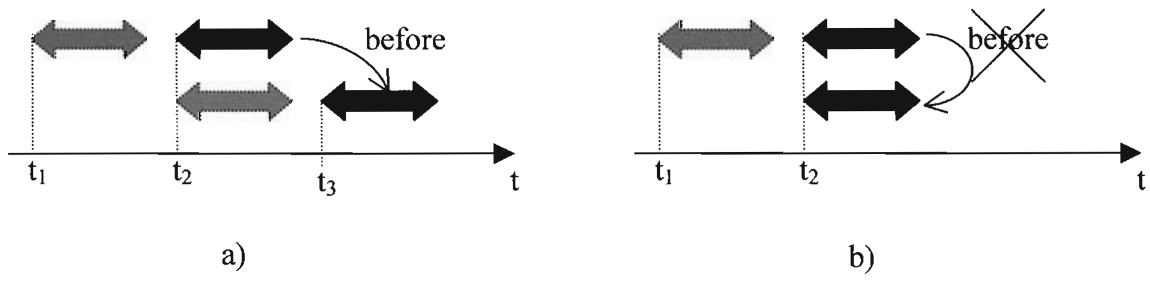
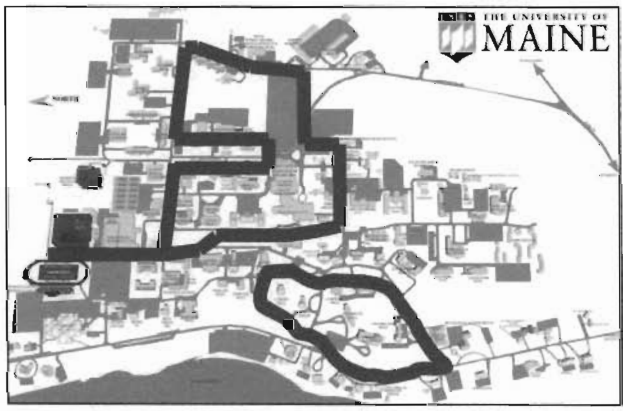
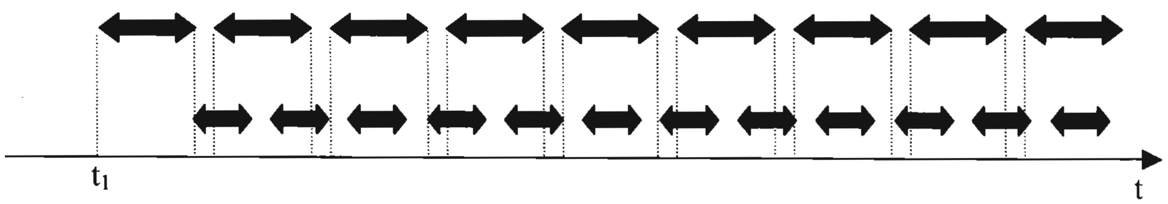


Figure 5.3. Different configurations of an animation where the start point of the red interval is redefined: a) a configuration in which the relative temporal relationship is preserved and b) a configuration in which the temporal relationship is violated.

Using a certain temporal relation between two intervals to capture some known relationship between objects' behavior does not cover some cases in which the behavior of the objects have a cyclic pattern of repetition. The temporal relation between such behaviors requires a richer representation (i.e., a representation that considers the set of relations among elements in a sequence of intervals). Consider, for example, the behavior of two buses running on the campus. Each bus has a different route on the campus (i.e., they serve different locations). The blue and red lines in the Figures 5.4a represent the spatial characteristics of the buses behavior (i.e., the buses route). Due to the difference in the length of each route, the amount of time for each bus to perform each run is different, but between each run the buses rest for the same period of time. Thus, the temporal characteristics of the buses' behavior are represented by a sequence of intervals intercalated with gaps of the same duration (Figure 5.4b). Each interval in the sequence corresponds to a run of the bus and the gaps represent periods when the buses are not running.



a)



b)

Figure 5.4. Graphical representations of the spatial and temporal characteristics of the movement of two buses with a cyclic behavior: a) the routes of the buses and b) the temporal characteristics of each run.

The graphical representation of the temporal characteristics of the buses' behavior (Figure 5.4b) allows an observer to reason about the buses' schedule. An observer can infer, for example, when the buses start to run, the duration of each run, and so forth. This representation, however, does not tell the observer the reason that the bus of the blue route starts its first run when the bus of the red route finishes its first run. It can be either a coincidence or an imposition between the buses' schedule.

Consider that is known from the application domain that the schedule of the buses is set up in a way that enforces a policy of the University that states that "whenever possible there is at least one bus running on the campus". Due to the duration of the periods of activity and inactivity of the buses, the temporal configuration shown in Figure 5.4b guarantees that there is always one bus running on the campus. This information can be registered in the model through a temporal relationship between the first run of each bus (e.g., the interval representing the first run of the bus on blue route is "met by" the interval representing the first run of the bus on the red route). The temporal relationship between the first run of each bus, however, is not robust enough not capture the knowledge from the application domain (i.e., the university policy concerned the schedule of the buses). This relationship reflects only a solution found by a person that sets up the schedule of the buses based on the duration of each run.

In this scenario, the animation model is not always capable of preserving the knowledge from the application domain. Depending on the manipulation made by an observer, the modified version of the animation may not reflect a "known" or "desired" temporal relationship between entities of the "real" world. Consider, for example, that an observer wants to create an animation where each run the bus on the blue route takes more time to complete each run. This situation can be motivated, for example, by the fact that the bus running on the blue route has a new driver and for security reasons the speed limit of the bus is be reduced by 5 mph during the driver's training period. Thus, changing the duration of each run of the bus on the blue route and preserving the temporal relationship between the first run of the buses generates a temporal configuration where sometimes both buses are not running (Figure 5.5).

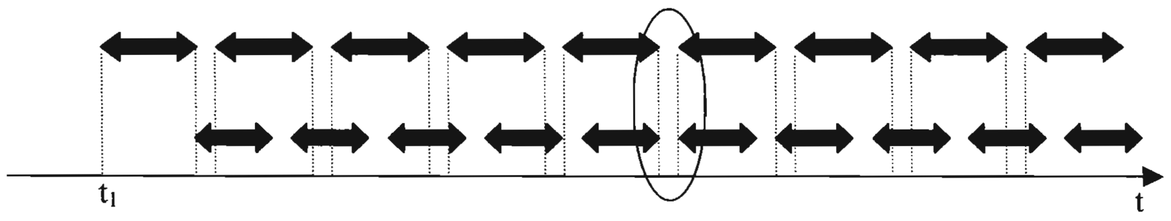


Figure 5.5. Graphical representations of the temporal characteristics of the buses' behavior in which the bus of the blue route has a new driver. The oval highlights the period in which neither bus is running.

When dealing with cyclic behaviors, a new set of temporal relationships is needed to relate these more complex structures. These relationships have to be less restrictive and able to address temporal relations that must hold between all intervals of the cycle. A temporal relationship of the kind "*maximize the occurrence of periods of non-concurrent activities*", for example, gives the animation model the information needed to find a new temporal configuration with always at least one bus running on the campus. There is no guarantee, however, that it is possible to find such a temporal configuration, but the application can find a configuration in which simultaneous periods of inactivity are distributed evenly during the day, minimizing the occurrence of long periods with no bus running on the campus.

The next section discusses the mechanism used to represent relationships among the temporal characteristics of objects' behavior and to update such characteristics when an observer changes the temporal configuration of the modeled animation.

5.2 Representing Temporal Constraints

In the model for *virtual exploration of animations*, *Course of Actions Interval* and *Animations Interval* are abstractions that deal with collections of time periods. These classes have an attribute to represent some known temporal relationship among elements in their collections of intervals. This attribute is called *temporal constraints*.

The attribute *temporal constraints* is implemented as a list of triples of the form (*temporal object*, *temporal object*, *constraint*). *Temporal object* is any instance of the classes of the temporal model introduced in chapter 4 (i.e., *Act*, *Course of Actions*, *Cycles*, and *Animations Intervals*). The element *constraint* is any information that represents a dependency between *temporal object*s. The types of *temporal objects* in the triple vary accordingly the class that the attribute belongs. For example, a *Course of Actions Interval* is composed of *Act Intervals*. Thus, the *temporal objects* in the triple are always instances of *Act Intervals* and the value of the element *constraint* is some information that relates two temporal intervals, for example, a temporal relation *metBy* between two *Act Intervals* informs that one act of the object follows the other.

The class *Animation Interval* is more complicated. *Animation Intervals* handles a multi-sort collection of intervals. Objects in this collection can be of the sort *Course of Actions Interval, Cycle Interval*, and *Animation Interval*. Thus, the range of values of the element *constraint* varies from values that relate two intervals to values that relate sequences of intervals of cyclic behaviors.

The implementation of attribute *temporal constraint* is straightforward. An operation called `InsertTemporalConstraints` adds elements in the list of the attribute *temporal constraint.* This operation registers all known dependencies among entities of the model. The operation `InsertTemporalConstraints` is polymorphic and available to all classes that deal with collections of intervals (i.e., it can be used to relate any sort of temporal objects). Thus, two major issues concerned the temporal constraint mechanism are the definition of possible values associated with the element *constraint* and the definition of the rationale used to keep modified versions of the animation coherent with the temporal constraints introduced in the model. The next sections discuss these issues in the context of temporal constraints between two intervals and between two cycles.

5.2.1 Temporal Constraints between Intervals

The first task to handle in the *temporal constraints* mechanism is the characterization of the elements in the list of triples of the attribute *temporal constraints*. Firstly, its need to characterize which kinds of temporal structure we are relating. Secondly, we need to identify which kind of constraint we intend to capture. Thirdly, we need to define the rationale used to update the temporal characteristics of all related temporal structures when an observer changes the temporal characteristics of an object's behavior.

The kinds of temporal structure that we dealing in this section are intervals and the constraint that we intend to capture are some temporal relation between two intervals. In this way, the major issue concerning the functionality of the temporal constraints mechanism becomes the characterization of the object *constraint*.

The range of values of the element *constraint* is strongly dependent on the kind of temporal relation that the model is able to express (i.e., we cannot define values of *constraint* without knowing the type of temporal relations that can be represented in the model). In this thesis we adopted the Allen's set of relationships between intervals (Allen 1983). This set is composed of thirteen temporal relations (i.e., *before*, *meets*, *overlaps*, *starts*, *contains*, *finishes*, *equals*, and their converse) and captures any relationship that may hold between two intervals.

Based in the Allen's set of temporal relations, we propose a set of fifteen values of constraints to relate two intervals (Table 5.1). Although it is natural to think about instances the element *constraint* representing only the basic set of temporal relations between intervals, we decide to represent the constraint where two intervals start together or finish together without having to constraint the duration of the related interval. The semantic of the constraints `makeStartTogether` and `makeFinishTogether` are discussed later in this section.

Based on the rationale used to update the temporal characteristics of related intervals, instances of the *constraint* element can be divided in two sub-groups: *tight* and *loose* (Table 5.1). The *tight* group of instances constrains at least one point of the interval involved in the relation. The constraint `makeEquals` represents the strongest constraint in the set since it defines the position of the start and end points of the related interval. The *loose* group of instances of *constraint* limits the range of possible values of the *related* interval end points, but does not unambiguously define the position of any end points.

	Values of the element Constraints	Possible Temporal Relations
Tight	`makeMeet`	*Meets*
	`makeMetBy`	*MetBy*
	`makeStartTogether`	*Starts* or *StartedBy*
	`makeFinishTogether`	*Finishes* or *FinishedBy*
	`makeEquals`	*Equals*
Loose	`makeBefore`	*Before*
	`makeAfter`	*After*
	`makeStart`	*Starts*
	`makeStartBy`	*StartedBy*
	`makeContain`	*Contains*
	`makeContainedBy`	*ContainedBy*
	`makeFinish`	*Finishes*
	`makeFinishedBy`	*FinishedBy*
	`makeOverlap`	*Overlaps*
	`makeOverlappedBy`	*OverlappedBy*

Table 5.1. Temporal constraints between a pair of intervals.

Once populated the list of triples of the attribute *temporal constraints*, the model has incorporated all knowledge from the application domain concerned temporal relationships among entities of the animation (i.e., intervals representing the temporal extend of object's behavior). Thus, any modification made by an observer in the temporal characteristic of an interval present in the list of the attribute *temporal constraint* causes the modification of all related intervals in a way that preserves known relationships stored in the model. Intervals that are not related in the list of the attribute *temporal constraint* keep their original temporal characteristics.

Consider, for example, an animation composed of the behavior of four objects. The temporal characteristics of these objects' behavior are represented by four *Course of Actions Intervals* (i.e., c1, c2, c3, and c4). Triples in the attribute *temporal constraints* captures known relationships among the temporal characteristics of these behaviors (i.e., [(c1,c2, `startTogether`), (c2,c3, `starts`), (c3,c4, `overlaps`)]). Given certain duration for the intervals, figure 5.6a shows a graphical representation of the *Course of Actions Intervals* and the temporal constrains register in the model.

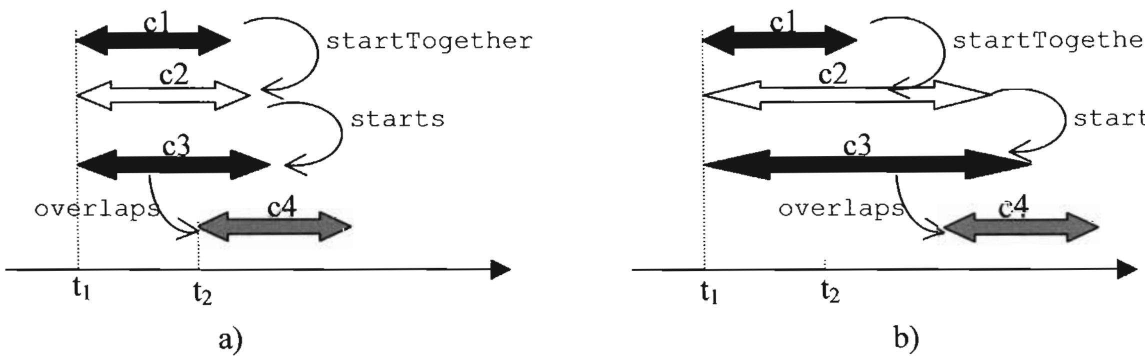


Figure 5.6. Temporal constraints between a Course of Actions Interval and different kinds of cycles.

Consider that an observer wants to simulate a situation where the behavior of the second object takes longer (i.e., the observer changes the duration of the yellow interval). If the yellow interval represents the behavior of a car moving between two locations, for example, an observer wants to slow down the movement of the car. Figure 5.6b shows a graphical representation of the modified version of the animation. In this version of the animation, the duration of the yellow interval is the one specified by the observer. Since there exist temporal constraints among the intervals, however, the application redefines the temporal characteristics of other intervals in a way that preserves all information stored in the list of the attribute *temporal constraints*. The first triple of the attribute *temporal constraints* captures the fact that blue and yellow intervals start together. In the modified version of the animation the temporal characteristic of the blue interval remains the same. The new duration of the yellow interval does not change this fact. The second *constraint* in the list captures the fact that the yellow interval starts the red interval. It means that the intervals have the same start point but the duration of the yellow interval is shorter. The new duration of yellow interval violates such a constraints (i.e., the temporal relation becomes *startBy*). Thus, in the modified version of the animation, the duration of the red interval is redefined in order to preserve the temporal constraint. The third *constraint* informs that red interval overlaps the green interval. Considering the new configuration of the red interval, the application changes the start point of the green interval and preserves the temporal relations *overlaps* between these entities.

The rationale used to update the temporal characteristics of the intervals related by a *constraint* from the group of tight *constraints* (e.g., `makeStartTogether`) is straightforward. The temporal characteristics of the interval are completely defined in terms of the temporal characteristics of the related interval and the intended *constraint*. The group of loose temporal constraints is more flexible in the sense that it gives implementers of the model more room to specify the best configuration of the related interval. For example, the decision about the new duration of the red interval and the new position of the green interval is left to implementers. In this chapter we abstract the implementations details.

The model of *virtual exploration of animation* supports other temporal structures that cannot be represented as a single interval. In the class *Animation Interval*, for example, some *temporal objects* can be of the sort *Cycle Interval*. It means that *Cycle Intervals* objects can be also elements in the triple of the attribute *temporal constraints*. In this way, it is possible, for example, to relate a *Course of Actions Interval* with a *Cycle Interval* or to relate two *Cycles Intervals* with *constraints* that relates two intervals (Table 5.1).

Consider, for example, the behavior of three objects. The first object has a non-cyclic behavior (i.e., the temporal characteristic of the object's behavior is represented by a single interval). The second and the third objects have a cyclic behavior with an *infinite* and *finite* pattern of repetition, respectively. The temporal characteristic of the cyclic object's behavior is represented by a sequence of intervals. Consider also that is known from the application domain that a certain occurrence of the infinite cyclic behavior starts when the non-cyclic behavior stops and that this same occurrence of the infinite cycle starts together with the first occurrence of the finite cyclic behavior. Figure 5.7 shows the graphical representations of the temporal characteristics of these objects' behavior and the temporal *constraints* introduced in the model. The yellow interval represents the temporal characteristics of the non-cyclic behavior and the red and blue intervals represent the related occurrences of the *Cycle Intervals*. In this scenario, the modification of any characteristic of an object's behavior makes the application to redefine the temporal characteristics of all related intervals using the same rationale discussed early in this section. In the case of the cyclic behaviors, the application considers only the colored occurrences of the cycles. Other occurrences of the cycles are redefined based on the structural elements of the *Cycle Intervals* (i.e., the periods of activity and inactivity, and the pattern of repetition), but they are not taken into account in the temporal constraints mechanism.

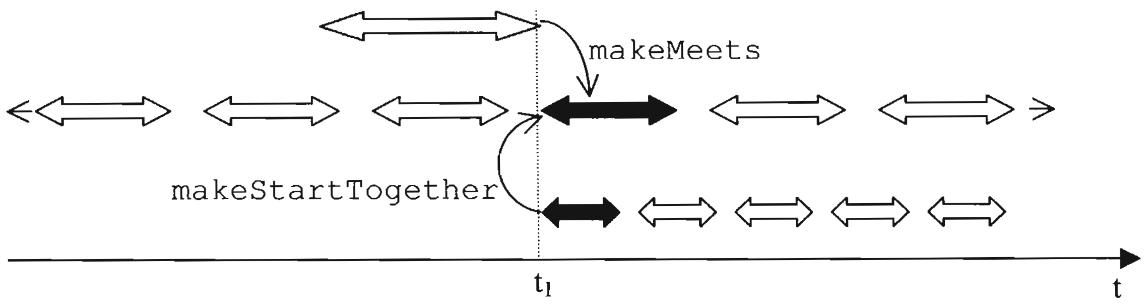


Figure 5.7. Temporal constraints between a non-cyclic behavior and cyclic behavior and between two cyclic behaviors.

Imposing a temporal constraint between a *Course of Actions Interval* and an occurrence of a *Cycle Interval* or between two occurrences of *Cycle Intervals* is meaningful only because a single temporal relation is in place. When two cycles are involved, however, a temporal relation between two *occurrences* is not always the most representative. Other occurrences of the cycles will have different temporal relations and with the possibility that the imposed relation is the less relevant (i.e., is the relation that occurs less frequently). The next section discusses an extension to values of *constraints* to be used when the *temporal objects* are cycles and when the knowledge that we want to capture involves temporal relations between all occurrences of the cycles.

5.2.2 Extending Temporal Constraints between Cycles

A sequence of intervals that do not overlap is the temporal representation of cyclic behaviors. In this way, the mechanism used to constraints pairs of intervals is no longer sufficient to be used by an application that deals with cycles.

A critical issue in the extension of the temporal constraint mechanism is the representation of the set of temporal relations among the basic units of time that comprise a cyclic behavior. Previous studies have addressed the issue of temporal relations between collections of intervals. These studies can be divided into three main groups. The first group (Ladkin 1986; Leban *et al.* 1986; Balbiani *et al.* 1988; Morris and Khatib 1997) considers relations among generalized sequences of recurring events, that is, these studies do not consider any constraints among elements of the sequence. The second group (Frank 1998; Hornsby *et al.* 1999; Osmani 1999) takes into account the cyclic pattern of the sequence of intervals but limit their scope to the particular case in which the cycles have the same period. The third group considers cycles with different periods (Cuckierman and Delgrande 2000; Terenziani 2003) but limit the representation of temporal relations to a disjunction of Allen's set of temporal relations (Allen 1983). This thesis is more related with the second and third groups of study but our reasoning process depends on a more detailed representation of temporal relations that is not addressed in any previous studies. This more detailed representation considers cycles with different periods and includes the frequency in which each temporal relation between occurrences of the cycles occurs. The next section discusses the representation of temporal relations between cycles with different periods.

5.2.2.1 Temporal Relations between Cycles

A temporal relation between occurrences of cycles is called *correlation* (Morris *et al.* 1996). The number of correlations, depending on the temporal characteristics of the cycles, can be infinite. After, a certain amount of time, however, the pattern of correlations repeats in a cyclic fashion. Thus, a finite subset of correlations can be chosen to represent all possible correlations that may hold between occurrences of these cycles. In this thesis we are interested in the smallest subset of correlations that captures also the frequency in which each correlation occurs. Consider, for example, two cycles in which pairs composed of one occurrence of each cycle are either under a temporal relation *startBy* or *contains*. Consider yet that the temporal relation *contains* occurs two times more frequently than the temporal relation *startBy*. Thus, the smallest subset of correlations must reflect this fact. In this case, an instance of the smallest subset of correlation between these cycles would be: {*contains*, *contains*, *startBy*}.

In order to compute the smallest subset of correlation between two cycles we need to take into account some quantitative information about the cycles (i.e., the duration of each cycle period). Consider, for example, animated objects that have a cyclic behavior with an infinite pattern of repetition. The first cycle has a period of twelve time units (nine units for its period of activity and three units for its period of inactivity). The second cycle has a period of eight time units (three units for its period of activity and five units for its period of inactivity). Figure 5.8 shows a small portion of a timeline with some occurrences of these cycles. A set of correlations can be retrieved from this sample. This set, however, do not necessarily represent the smallest subset of correlations between these cycles.

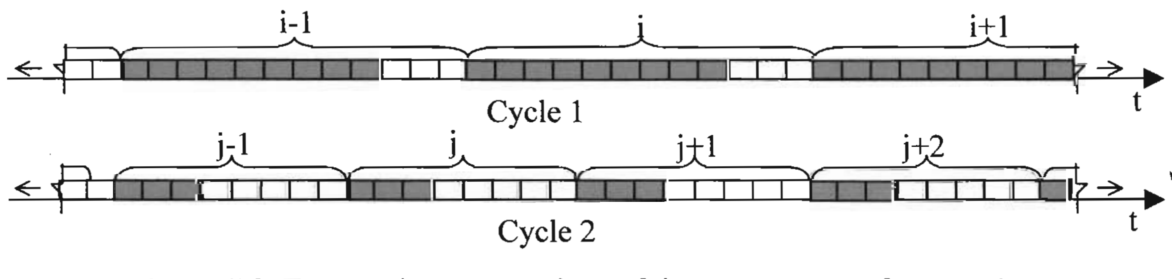


Figure 5.8. Temporal representations of the occurrences of two cycles.

The minimum amount of time required to capture the frequency of all correlations that may hold between occurrences of two cycles is determined by the duration of the periods of the cycles involved in the process. This amount of time is called *extended period of the cycle* (E). The duration of the *extended* period equals the least common multiplier between the duration of the cycles. Thus, the duration of the first cycle's period (D_1) and the duration of the second cycle's period (D_2) are used in the computation of the duration of the cycles' *extended* period (D_E) as follows: D_E=LCM(D_1, D_2). LCM is an operation that returns the least common multiplier of two integers. The duration of the *extended* period determines the number of occurrences of each cycle needed to cover the entire *extended* period. In the cycles depicted in the Figure 5.8, for example, the number of occurrences of cycles 1 and 2 needed to compute the smallest subset of correlations are two and three, respectively.

Once the number of occurrences for each cycle has been determined, an application can select two subsets of occurrences of each cycle and compute all temporal relations that hold between pairs of these occurrences (Figure 5.9a). Since these subsets of occurrences do not represent any specific set of occurrence of the cycles, they can be also depicted using a cyclic representation (Figure 5.9b). In the cyclic representation the circumference of the circle equals the extended period of the cycles.

The smallest subset of correlations is obtained by comparing the period of activity of the set of occurrences of the cycles 1 and cycle 2. Elements of the subset of correlations are one out of twelve possible temporal relations between intervals in a cyclic representation, that is, relations *before* and *after* are collapsed in a single relation called *disconnected* (Frank 1998; Hornsby *et al.* 1999). Thus, the smallest subset of correlation between the cycle 1 and 2 are composed of the temporal relations *startedBy*, *overlaps*, and *contains*.

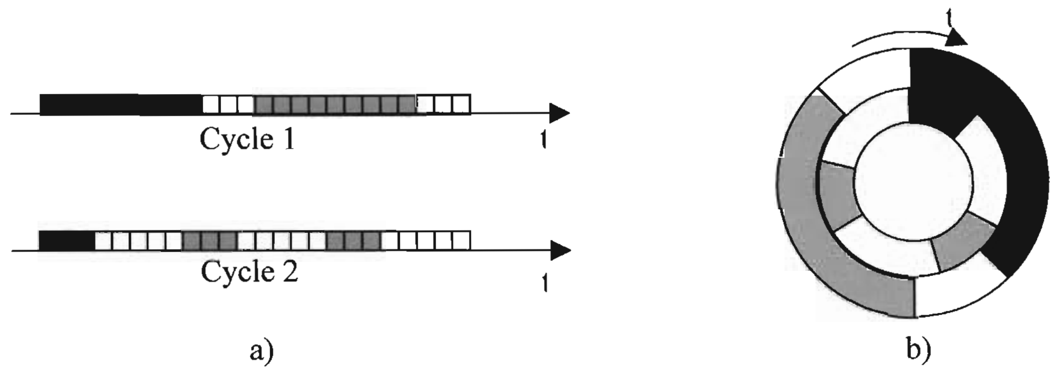


Figure 5.9. Temporal representations of subsets of occurrences of two cycles covering the extended period of the cycle: a) a linear representation and b) a cyclic representation.

Based on the representation of temporal relations between two cycles (i.e., the smallest subset of correlations), we propose an extension of the range of values of the element *constraint*. These values of *constraints* can only be used when *temporal objects* in the triple of the attribute temporal constraints are of the sort *Cycle Interval*. We call this set of values *cyclic constraints*.

5.2.2.2 Constraints between Occurrences of Cycles

The role of temporal constraints between cycle intervals is the same as the temporal constraints between intervals (i.e., to represent some known relationships among entities of the model and keep these characteristics in modified versions of the animation). *Cyclic constraints*, however, are less rigid than the set of *constraints* between two intervals and the rationale used to maintain the modified version of the animation coherent with the constraint introduced in the model is more complicated. In order to represent some known relationship between cycles we propose a set of eighteen values for the element *constraint*. Instances of constrains are values of the form `maximize`*`Relation`* and `minimize`*`Relation`*, where *`Relation`* assumes one of the following values: *`Meets`*, *`MetBy`*, *`StartTogether`*, *`FinishTogether`*, *`Equals`*, *`Disconnected`*, *`Contain`*, *`Overlaps`*, or *`OverlappedBy`*.

The set of values of cyclic constraints does not refer to all kinds of temporal relations individually. We collapse certain temporal relations into a single temporal relation that does not depend on the duration of the intervals. In this sense, the pairs of temporal relations *start* and *startedBy*, *finish* and *finishedBy*, and *contains* and *containedBy* are represented by the constraints `maximize` or `minimize` `startTogether`, `finishTogether`, and `contain`, respectively. This is motivated by the fact that the mechanism used to modify the temporal characteristics of the cycles while keeping the modified version of the animation coherent with *cyclic constraints* does not change the duration of the cycles involved in the process.

Consider, for example, the case of the buses introduced earlier in this chapter. The original configuration of the buses was defined based on the knowledge from the application domain that requires, whenever possible, that buses run at the same time. This constraint between the behaviors of the buses can be represented in the model through an instance of *cyclic constraints* of the type `maximizeDisconnections`. The temporal relation *disconnected* means that when one object is performing its associated behavior the other object is at rest. In this way, when an observer changes the temporal characteristics of one cycle, the application can look for a temporal arrangement of the cycle intervals in which the smallest subset of correlations has the largest number of relation *disconnected*.

The rationale used to redefine the temporal arrangement of *Cycle Intervals* and enforce a *cyclic constrain*t is very different from the rationale used for non-cyclic temporal structures. First, the redefinition of the temporal characteristics of the cycles is limited to redefinition of the of the start point of the cycles' occurrences. Thus, this mechanism never changes the duration of the cycles' occurrences. Second, *cyclic constraints* do not impose a certain temporal relationship or a set of temporal relationships between the *Cycle Intervals*, but requires that a certain temporal relation hold more or less frequently between occurrences of the cycles. Moreover, the temporal relation that an instance of *cyclic constraints* refers to does not necessarily holds between any pair of occurrences. In the case of the buses, for example, it is impossible to achieve a temporal configuration for the schedule of the bus in which a temporal relation *disconnected* holds. The duration of both periods of activity of the buses is larger than their periods of inactivity.

The algorithm used to find the best temporal configuration of the cycles (i.e., the start points of occurrences of the cycles) that reflects a *cyclic constraints* is the same as the algorithm used by operations that allow an observer to modify the temporal configurations of cyclic behaviors. This algorithm is discussed in next chapter.

5.3 Summary

This chapter discussed a mechanism to represent temporal constraints among entities of the animation model. This mechanism captures all known temporal relations among the elements in a collection of intervals maintained by entities at different levels of abstractions. Thus, any modification in the temporal characteristics of an interval in the list of *temporal constraints*, cause the redefinition of the intervals in the list in a way that preserves all *constraints* introduced in the model. These modifications occur during the manipulation of the temporal characteristics of the elements of an animation performed by a user in order to produce new views of the environment.

Temporal constraints is a robust mechanism to incorporate knowledge from the application domain in a form of temporal relationships between intervals. The complexity and computational effort to maintain such a mechanism, however, are useless if the temporal configuration of the animation is fixed (i.e., cannot be modified by a user). In order to allow an observer to interfere with the temporal configuration of the animation, classes of the temporal model have a set of operations that allow a user to modify the temporal organization of the components of the animation. These operations are called *composition operations*.

CHAPTER 6

TEMPORAL COMPOSITION

Classes representing high-level abstractions of objects' behavior deal with collections of time periods. In these abstractions intervals are positioned along a temporal axis by specifying their start points. In an environment tailored for users to manipulate animations, however, this mechanism is time consuming, susceptible to errors, and hard to maintain.

Using temporal relations between time periods (Little and Ghafoor 1993; Weiss *et al.* 1995) is a more natural way to position temporal intervals. Existing animation models have explored the use of temporal relations between intervals (Fiume *et al.* 1987; Dollner and Hinrichs 1997). These models, however, are primarily concerned with the production of the animation. The set of operations of such models acts over low-level abstractions of the objects' behavior. Thus, they are not suitable for manipulation by users. Moreover, a critical problem of these models is that they do not consider cyclic behaviors, a recurrent type of behavior in GIS applications.

This chapter introduces a set of operations to redefine the arrangement of temporal characteristics of objects' behavior taking into account the temporal relationship between their arguments. This set of operations is called *composition* operations.

6.1 Composition Operations

Composition operations are used by an observer to manipulate the configuration of the animation at different levels of abstractions creating new views of the dynamic environment. These operations have two *temporal objects* as arguments: a *reference* and a *target* object. The observer is responsible for identifying the *reference* and the *target* objects. The main functionality of these operations is that the temporal characteristics of the *target* object are redefined to satisfy a certain temporal relation between the *reference* and *target* objects. The resulting temporal configuration is given by the semantics of each operation.

The sorts of the *reference* and *target* objects are any classes of the temporal model. Thus, it is possible for an observer to redefine a new temporal configuration of the animation manipulating the temporal characteristics of objects' behavior from different levels of granularity. For example, an observer can impose a temporal relation between an interval representing the behavior of a group of objects (i.e., an *Animation Interval*) and an interval representing a single act of an object (i.e., an *Act Interval*). Some compositions operations, however, are specific for cycles and do not accept other sort of *temporal objects* as arguments than *Cycles Intervals*. Thus, the set of *composition* operations can be divided in two main groups. The first group of operations is used to compose animations by redefining the temporal characteristic of a single interval. Cycles can be used as arguments for this group of operations but only the *occurrence of reference* (a single interval) of the cycle is taken into account. The second group of operations is specific for cycles. Next sections discuss these groups of operations.

6.2 Composition Operations between Two Intervals

The goal of *composition operations* is similar to the goal of the *temporal constraint* mechanism (i.e., to impose temporal relationships among temporal characteristics of objects behavior). In the context of the *temporal constraints* mechanism, the imposed temporal relationship informs a known dependency between entities of the animation. In the context of *composition operations*, the imposed temporal relationship reflects an observer intention to manipulate the temporal configuration of the animation to gain insights and discover relationships among geographic phenomena. Despite the similarity of their goals, these mechanisms have different requirements. The *temporal constraint* mechanism requires a sophisticated functionality to keep the modified temporal configuration of the animation coherent with constraints introduced in the model. It includes the redefinition of temporal characteristics of all related intervals or the modification of the semantics of associated objects. The latter requirement is discussed in chapter 7. Major requirements of *composition operations* are that this set of operations needs to be small, intuitive, and have the same functionality across different levels of granularities. These requirements facilitate the assimilation of the functionalities of these operations by a user.

Composition operations are part of the user interface. These operations represent the bridge between the user and instances of the model's abstractions. Thus, these operations are strongly dependent on the way that the user perceives such abstractions. In this thesis we propose a graphical interface (chapter 8) in which the representation of the temporal characteristics of objects' behaviors are depicted as intervals positioned along a timeline. In order to make the set of *composition operations* intuitive to the user, it is essential to associate the operations' functionality with the graphical representation of their arguments (i.e., operations used to modify the position of intervals along a timeline need to have a graphical appeal). Thus, we borrowed from the drawing packages that deal with geometric objects some of operations used to align graphics objects (Table 6.1). These operations have a well-known functionality in the graphical domain that can be easily extrapolate to the temporal domain. In the context of an animation interface, these operations impose a temporal relationship between two intervals selected as arguments.

Composition Operations	Possible Temporal Relations
`makeTouch`	*meets* or *metBy*
`makeAlignLeft`	*starts, startedBy, or equal*
`makeAlignRight`	*finishes, finishedBy, or equal*
`makeAlignCenter`	*contains, containedBy, or equal*
`makeCross`	*overlaps* or *overlappedBy*
`makeEqual`	*equal*
`makeMirror`	*Any*

Table 6.1. Composition operations between intervals and possible temporal relations holding between the intervals after the execution of the operation.

The temporal relationship that holds between the intervals after a *composition operation* depends on the order in which the intervals are selected by the user and on the configuration of the intervals prior to the execution of the operation. The operation `makeTouch`, for example, makes the start point of the *target* interval coincide with the end point of the *reference* interval or vice-versa, depending on which is closer. Thus, possible temporal relations between the *reference* and *target* intervals would be the temporal relations *meets* or *metBy*. The operation `makeMirror` changes the position of the midpoint of the *target* interval to a new position where the absolute value of the distance from the midpoint of the *reference* interval is preserved. This operation acts as a converse for the other operations in the set. If the temporal configuration resulting from the operation makeTouch is the relation *meets*, for example, the observer can turn the temporal configuration to a relation *metBy* via the operation makeMirror. We believe that the semantics of the other operations are self-explanatory. The amount of overlapping in the operation makeCross is the only issue that deserves further consideration. We define this value as half of the smallest duration of the *reference* and *target* intervals. For example, if the *target* interval is shorter than the *reference* interval, the start point of the target interval coincides with the mid point of the reference interval (Figure 6.1a). If the *target* interval is longer than the *reference* interval, the mid point of the *target* interval coincides with the end point of the reference interval (Figure 6.1b).

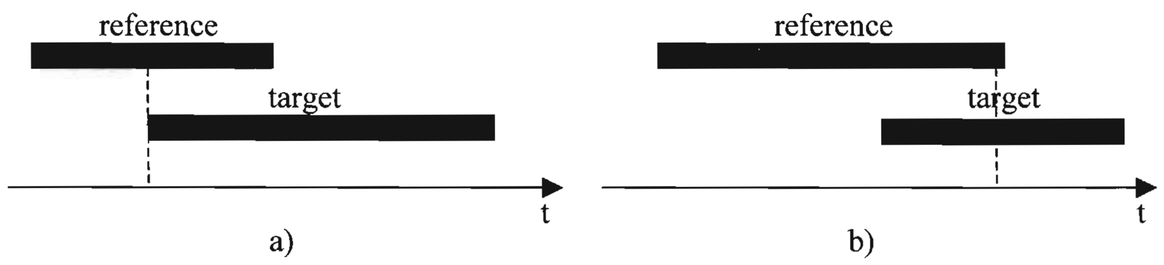


Figure 6.1. Two different configurations of the animation for the operation makeCross: a) a configuration in which the reference interval is shorter than the target interval and b) a configuration in which the reference is longer than the target interval.

These set *composition operations* takes two *temporal objects* as arguments. It means that *Cycle Intervals* can also be used as arguments of these operations. In this case, the *composition operation* considers only a single occurrence of the cycle (i.e., the cycle's occurrence of reference). Changing the start point of a cycle's occurrence of reference changes the start point of all occurrences of the cycle, but only the occurrence of reference is under the temporal relation specified by the observer through a certain composition operation.

Similar to the *temporal constraints* mechanism, there are some *composition operations* specific for cycles and do not accept other sort of objects as arguments. These operations are discussed in next section.

6.3 Composition Operations Between Intervals

Cyclic compositions are operations that change the temporal characteristics of cyclic behaviors taking into account the temporal relation that must hold between all occurrences of the cycles used as arguments (i.e., the *reference* and *target* cycles). These operations are divided into two groups. The first group changes the durations of periods of activity and inactivity of the *target* cycle in a way that a particular temporal relation holds between all occurrences of the cycles. The second group of operation preserves the duration of *target* cycle but changes the start point of its occurrences in a way that a certain temporal relation among occurrences of the *reference* and *target* cycles occurs more or less frequently. These groups of operations are discussed in next sections.

6.3.1 Changing the Structural Elements of the Cycle

The first group of *composition operations* between cycles is composed of the operations `makeConcurrent` and `makeAlternate`. These operations change the values of the structural elements of the *target* cycle (i.e., the periods of activity and inactivity of the cycle) based on the values of the structural elements of the *reference* cycle.

The operation `makeConcurrent` is the cyclic version of the operation `makeEqual` between two intervals. This operation redefines the start point and the duration of the *target* cycle in a way that their occurrences and occurrences of the *reference* cycle are concurrent (i.e., they start at the same time and have the same duration). The operation `makeAlternate` redefines also the start point and the duration of occurrences of the *target* cycle but in a different way. For this operation the start point of the *target* cycle's occurrences coincide with the end point of the *reference* cycle's occurrences and the duration of the *target* cycle coincides with the period of inactivity of the *reference* interval. In this way, the behavior associated with the *reference* cycle is seen only when the behavior associated with the *target* cycle is not happening, and vice-versa. Consider, for example, the temporal configuration of the cycles depicted in Figure 6.2a. Blue intervals represent occurrences of the *reference* cycle and red intervals represent occurrences of the *target* cycle. The temporal configurations of the cycles after the operations `makeConcurrent` and `makeAlternate` are shown in Figures 6.2b and 6.2c, respectively.

The operations `makeConcurrent` and `makeAlternate` produce a temporal configuration in which the cycles have the same period (the sum of the durations of the period of activity and period of inactivity is the same). Thus, there exists a single temporal relation between all occurrences of the cycles. The first case is the temporal relation *equals* and the second case is the temporal relation *meetsTwice* (Hornsby *et al.* 1999), that is, a conjunction of the temporal relations *meets* and *metBy*.

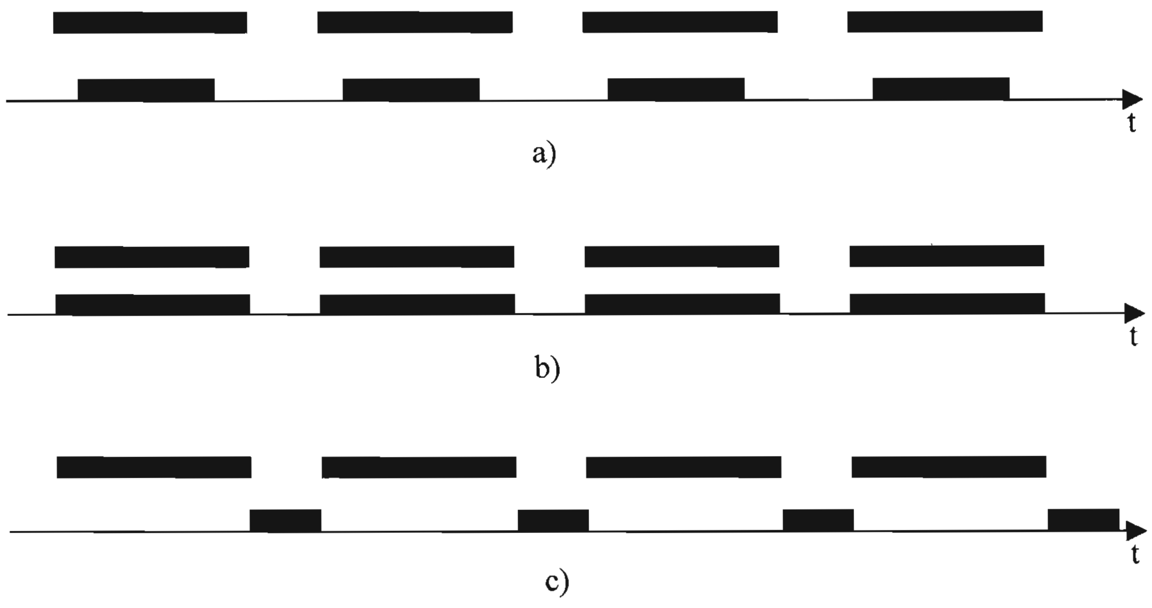


Figure 6.2. Cycles compositions: a) the original temporal configuration, b) the temporal configuration after the operation makeConcurrent, and c) the temporal configuration after the operation makeAlternate.

6.3.2 Changing the Start Point of Cycle's Occurrences

The second group of *composition operations* between cycles changes the position of the start point of the occurences of the *target* cycle in a way that maximizes or minimizes a certain correlation between their arguments. These operations preserve the duraton of periods of activity and inactivity of the cycles used as arguments.

The algorithm used to find the position of the *target* cycle that best represents the intended temporal configuration between the cycles is identical to the algorithm used by the *temporal constraint* mechanism cycles to enforce a *cyclic constraint*.

The first task of this algorithm is to identify all possible temporal configurations between two cycles obtained by changing the start point of all occurrences of one cycle. The number of possible configurations depends on the number of cycles' occurrences needed to cover the *period of equivalence* (i.e., the amount of time needed to capture all correlations between two cycles). Consider, for example, the occurrences of the *reference* and *target* cycles during the period of equivalence (Figure 6.3)

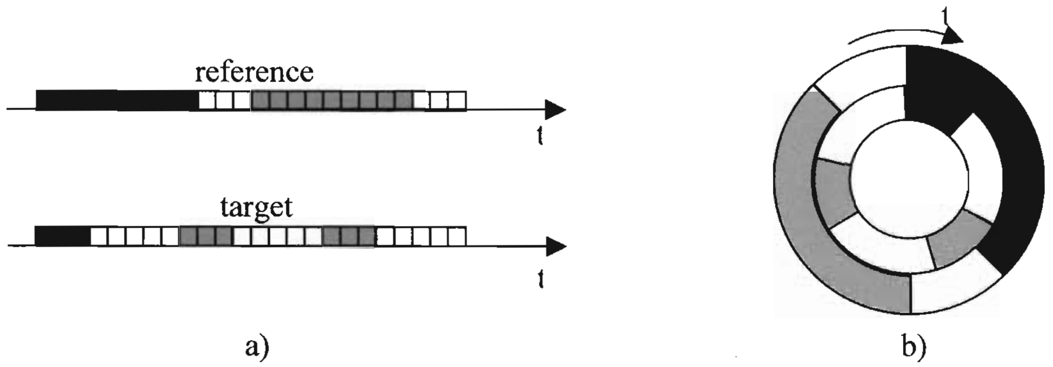


Figure 6.3. The linear and cyclic representations of occurrences of the reference (outer) and target (inner) cycles needed to capture the smallest subset of correlations.

In order to obtain all possible temporal configurations between these cycles, the algorithm increments and decrements by one unit of time the start point of *target* cycle. For each increment or decrement the algorithm computes the smallest subset of correlations. Figure 6.4 depicts the four possible temporal configurations between the *reference* and *target* cycles and their respective smallest set of correlations.

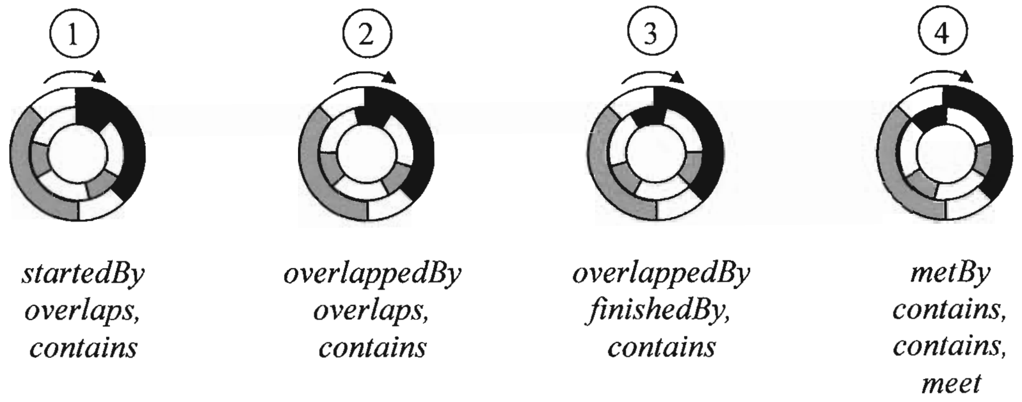


Figure 6.4. Possible temporal configuration between the reference (outer) and target (inner) intervals and the smallest set of correlations for each configuration.

Once computed all possible configurations between the cycles, the algorithm can select the configuration that maximizes or minimizes the occurrence of a certain temporal relation in the smallest set of correlations. Consider, for example, that the temporal relation to be maximized is *meets*, *metBy*, *contain*, *StartTogether*, or *FinishTogether*. In these cases, the algorithm can unambiguously select a certain temporal configuration from the set of all possible configurations. If the intention is to maximize the relation *meet*, *metBy*, or *contains*, for example, the fourth configuration is selected. If the intention is to maximize the relations *startTogether* or *finishTogether*, the first and the third configurations is selected, respectively.

If the intention is to maximize the relation *equal*, *disconnected*, *overlap*, or *overlappedBy*, however, the criterion of frequency of occurrences is not enough to select a unique configuration. The criterion of maximizing the temporal relation *overlap* or *overlappedBy*, for example, is satisfied by two different configurations each (i.e., the criterion of maximizing *overlap* is satisfied by the first and second configurations and the criterion of maximizing *overlappedBy* is satisfied by the second and third configuration). The criteria of maximizing *equal* or *disconnected* are satisfied by every configuration. Since the relations *equal* and *disconnected* do not hold between occurrences of the *reference* and *target* cycles, all configurations have the same frequency of these temporal relations (i.e., no occurrence of such relations).

This scenario is worst when the intention is to minimize a certain temporal relation. In this case, the algorithm can identify only the configuration that satisfies the criterion to minimize the temporal relations *overlap* and *overlappedBy* (i.e., the fourth configuration).

The limitation of the frequency of relation criterion requires the definition of additional criteria that can be used to break the tie when more than one configuration satisfy the intended relation. The second criterion is qualitative and considers the frequency of closest temporal relations (in a topological sense) that occur in the set of correlations. The third criterion is quantitative and considers the topological distance of the temporal relation.

In order to compute frequencies of closest relations we used the idea of conceptual neighborhood introduced by Freksa (Freksa 1992). A conceptual neighborhood of a temporal relation between two intervals is a different temporal relation obtained by atomic deformations of one of the intervals. Such a deformation can be 1) the redefinition of the interval's start or end point, 2) the redefinition of the interval's start and end points in a way that preserves the duration of the interval, or 3) the redefinition of the interval's start and end points in a way that preserves the location of the midpoint of the interval. Depending on the type of deformation, the structure of conceptual neighborhood changes. Freksa called these structures A-, B-, and C-neighborhood, respectively.

The second type of deformation is the only method that preserves the duration of the intervals. This deformation is equivalent of sliding an interval along the temporal axis. Since it is the same mechanism used to obtain different configurations of the smallest set of correlation between two cycles, we adopt the structure of b-neighborhood.

Figure 6.5 depicts the structure of the conceptual neighborhood of the set of relations between occurrences of cycles. This structure differs from the structure proposed by Freksa (Freksa 1992). In our case we are dealing with a representative set of intervals that captures temporal relations between two cycles. These occurrences do not represent any specific occurrence of the cycle. Thus, in the conceptual neighborhood diagram the temporal relations *before* and *after* are collapsed in a single relation *disconnected.*

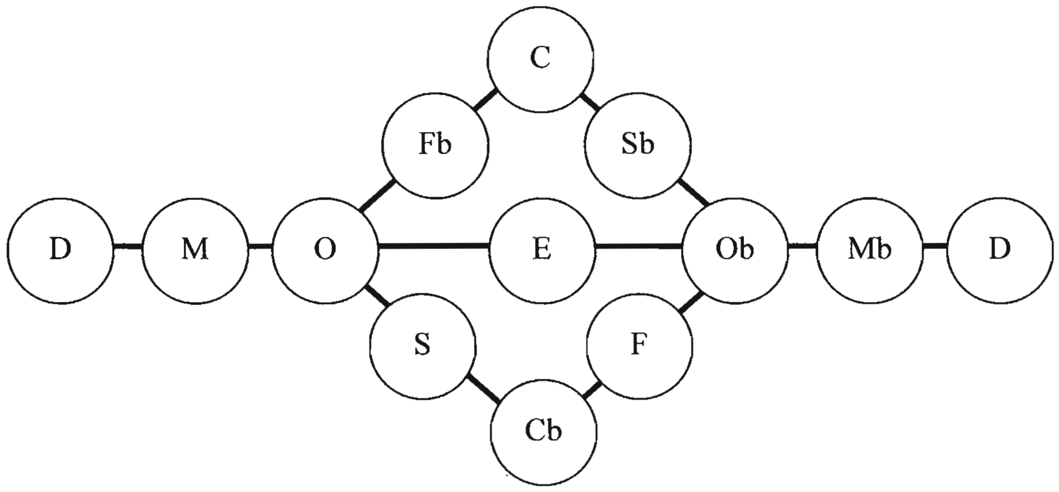


Figure 6.5. Conceptual b-neighborhood structure for temporal relations between occurrences of cycles.

Based on the structure of the b-conceptual neighborhood, we define as the second criterion to be used by the algorithm the configuration of the smallest set of correlations with the largest or smallest frequency of conceptual neighborhoods of the intended relation.

Consider, for example, the intention of maximizing the temporal relation *disconnected* between the cycles of the previous example. Since the temporal relation *disconnected* does not occur in the set of correlation, the intention of maximize *disconnected* is satisfied by all different configurations. Considering the frequency of conceptual neighborhoods of the relation *disconnected* (i.e., *met* and *metBy*), however, the fourth configuration is the "closest" one that satisfies the intended relation (Figure 6.4). Therefore the fourth configuration is selected.

The process of breaking a tie with conceptual neighborhoods is recursive in the sense that it can be extend for different degree of conceptual neighborhoods. Thus, if the immediate conceptual neighborhood is not sufficient to distinguish among different configuration, the second-degree conceptual neighborhood can be used. Consider, for example, the intention of maximizing the temporal relation *overlaps*. The first and second configurations satisfy the first criterion (i.e., they have the same frequency of the relation *overlaps*). Considering the conceptual neighborhood of the relation *overlaps* (i.e., the relations *meet*, *starts*, *finishedBy*, and *equal*), the first and the second configurations are still undistinguishable. These configurations have no occurrences of conceptual neighborhood relations. At the second degree of conceptual neighborhood, however, the second configuration is chosen since it has an occurrence of a conceptual neighborhood in second degree (i.e., the relation *overlappedBy*).

Considering only the criterion of frequencies of relations at different levels of conceptual neighborhood, however, does not give the "best" configuration when the relations under consideration are *disconnected*, *contains*, *containedBy*, *overlapps*, and *overlappedBy*.

Consider, for example, two cycles with the same period and different periods of activity (i.e., the smallest set of correlation has a single temporal relation). If the intention is to maximize the relation *disconnected*, for example, three temporal configurations are possible (Figure 6.6). Using only frequency of temporal relations and conceptual neighborhood, however, the algorithm is unable to select a configuration among the set of possible configurations. Because the temporal relation to maximize is disconnected, the second configuration should be chosen. It can be interpreted as that the occurrence of the *target* cycle in the second configuration is "more diconnected" than it is in the first and in the third configuration.

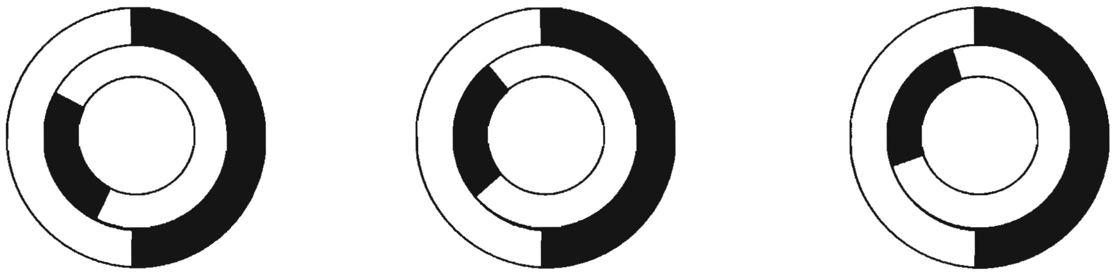


Figure 6.6. Different configuration of two cycles with a temporal relation disconnected.

In order to distinguish a certain configuration among set of correlations with the same frequency of temporal relations, we introduce a weight for each temporal relation. A measure for such a weight takes into account the topological distances between the relation and its closest neighborhoods. We define as a topological distances the number of unitary increments and decrements in the target interval needed to change the underlying relation. Figure 6.7 shows the representations of the topological distances in a conceptual b-neighborhood structure for the relations *disconnected*, *contains*, *containedBy*, *overlapps*, and *overlappedBy*. The arrows in the figure represent the number of atomic increments or decrements in the *target* interval needed to change the relations (i.e. their topological distances). We do not consider the topological distances of the temporal relations *meets*, *starts*, *finish*, *equal*, and their converses. For these relations, any unitary increment or decrement changes the underlying temporal relation. Thus, they are not useful in our reasoning.

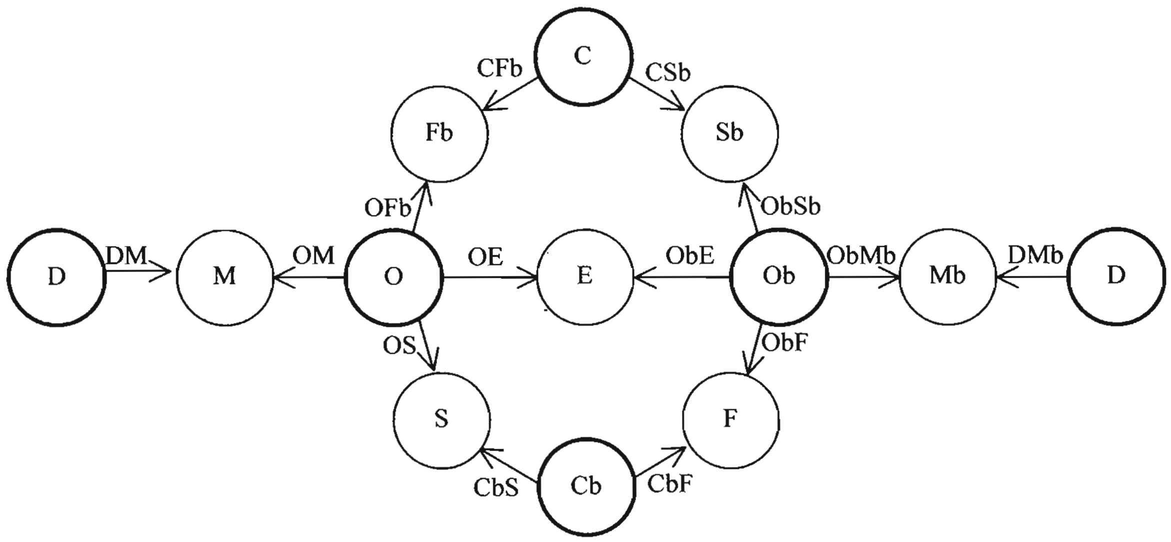


Figure 6.7. Topological distances associated with relations of the loose group of temporal relations in the conceptual b-neighborhood structure.

Each relation has two topological distances: the distance obtained by incrementing the target interval (D^+) the distance obtained by decrementing the target interval (D^-). For the relation *contains,* for example, the topological distance D^+ is represented by the arrow CFb and the topological distance D^- is represented by the arrow CSb. In the case of the relation overlaps, the topological distance D^+ is represented by the arrow OM and the topological distance D^- is represented by the arrows OFb, OE, or OS. The exact relation obtained by the decrementing the *target* interval under the relation *overlaps* depends on the duration of the *reference* and the *target* intervals, but the topological distances OFb, OE, or OS are the same.

Based on the pair of topological distances of each relation we define the *index of topological distance* (I_{TD}). The I_{TD} is computed as follows:

$I_{TD} = | D^+ - D^- | / (D^+ + D^-)$

A characteristic of the topological distances D^+ and D^- is that the sum of these distances is constant for a given *reference* and *target* interval. In this way, the index of topological distance ranges between 0 and 1, inclusive. For relations of the *loose* group of temporal relations, however, the I_{TD}'s capture the number of increments needed to change the underlying relation. I_{TD}'s close to 1 mean that the relation is about to change due to an increment or decrement of the *target* interval. I_{TD}'s close to 0 represent a configuration where the relation will remain the same considering the largest number of increments and decrements of the *target* interval.

The index of topological distance is used as a quantitative criterion wherever the qualitative criterion of frequency of temporal relations fails to distinguish identical configurations (i.e., sets with the same of number of correlations).

The criterion of topological distances is different if the intention is to maximize or minimize the occurrence of a temporal relation. If the intention is to maximize a certain temporal relation the criterion is to select the set that has the smallest I_{TD} for the intended relation. If the intention is to minimize a certain temporal relation the criterion is to select the set that has the greatest I_{TD} for the intended relation. If we reach the highest level of conceptual neighborhood without discerning a unique configuration, we consider that these configurations are undistinguishable with respect to b-neighborhood and topological distances, and everyone satisfies the intended relation.

The algorithm used to find a configuration of the *target* interval considering the frequency of each temporal relation is adequate only for the temporal constraint mechanism. For the user of the animation, this mechanism generates a large number of operations in the graphical interface, which can become very confused. In order to keep the set of *composition* operation between cycles small and intuitive, we collapse some temporal relations in a single *composition* operation. Thus, the types of correlation that can be enforced are to maximize or minimize the temporal relations *disconnected*, *touching*, *overlappings* and *containement*. The temporal relation *touching* encompasses the relations *met* and *metBy*, the temporal relation *overlapping* encompasses the relations *overlaps* and *overlappedBy*, and the temporal relation *containment* encompasses the relations *start*, *finish*, *contains*, *equals*, and their converses. The functionalities of these compositions operations are discussed with a prototype implementation of the model in chapter 8.

6.4 Summary

This chapter introduced the set of *composition operations*. This set of operations gives a user the opportunity of changing the temporal arrangement of the elements of the animation at different levels of granularity. These operations are divided in two groups: a group that deals with two intervals and a group that deals with cycle intervals.

This chapter also introduced an algorithm to redefine the temporal configuration of cycle intervals in a way that maximize or minimize occurrences of a certain temporal relation. This algorithm is used in the context of temporal constraints between cyclic behaviors and in the context of composition operations between cycle intervals.

The next chapter introduces the semantics part of the framework of virtual exploration of animations. The semantics model extends the traditional animation model by introducing entities that explicitly represent the semantics of VR objects. The semantics direct the interaction of the objects with the observer and other objects in the environment.

CHAPTER 7

SEMANTICS OF VR OBJECTS

The conceptualization of data model with high-level abstractions of objects' behaviors and a rich set of operations to manipulate such behaviors provides the means for an observer to produce and investigate new views of the environment. During the production phase, an observer uses operations over temporal intervals positioned along a timeline as a framework to create new temporal configurations of objects' behavior. During the investigatory phase, an observer has a wide variety of devices to support the exploration of the environment. The most usual device is a computer screen presenting projections of the four-dimensional world (i.e., a sequence of projections of the three-dimensional space). An increasing number of GIS applications (Kraak *et al.* 1999; Neves *et al.* 1999; Reddy *et al.* 1999; Verbree *et al.* 1999; Raper 2000; Zlatanova 2000), however, are extending their representational capability to present their information in virtual reality environments.

The ability of the user to manipulate objects' behavior and the fact that the exploration of the information takes place in an immersive environment imposes new requirements on the data model that supports the presentation application. First, the manipulation of the temporal characteristics of the behavior of an object may produce a situation where two objects originally dissociated, start to interact. For example, two objects that were at different positions at a certain instant in time may compete for the same location in the modified version of the animation. Second, in an immersive environment there is a presence of a new actor (i.e., the observer) interacting with other objects in the environment. Different from non-immersive environments, interactions between the observer and the objects become an important issue in VR environments (Kraak 2002). The key point in both cases is interactions that involve constraints imposed by an object on other objects or an observer's actions (e.g., a building that blocks the path of the observer, an obstacle that hides another object, or a heavy object that cannot be moved).

The type of constraints imposed by the objects depends on their associated semantics. In current data models for VR environments, the semantics of the objects are almost always assumed by the observer and derived from the context of the application. These semantics, however, play a significant role in the interaction between VR Objects, as well as in the interaction between observers and the environment.

This chapter introduces a new classification of VR Objects and describes the associated semantics of each element in the taxonomy(Campos *et al.* 2002; Campos *et al.* 2003b). This chapter introduces also a model to represent the semantics of VR objects over time and the rationale for modifying such semantics under the effect of a manipulation by the observer of the object behavior.

7.1 Characteristics of VR Objects' Semantics

The semantics of VR objects directs the way that these objects interact among themselves and with an observer. These semantics are based on three salient characteristics of all objects in the environment: *activity*, *existence*, and *visibility*.

Activity is a characteristic of an object that describes periods when the object is performing an associated action. Based on its activity an object can be considered *active* or *inactive*. An *active* object has the value of at least one of its attributes different from the value perceived in the previous time of observation (e.g., an object that changes continuously its position or shape at different instants in time). An *inactive* object, on the other hand, refers to the case where all the values of an object's attributes equal the values of the attributes from the previous time of observation.

Data abstractions of the animation model describe the activity of a special kind of object in a VR environment (i.e., a *performer*). *Performer* was defined as an abstraction that represents the geometry and appearance of dynamic objects. *Performer* objects, however, do not have an active state all the time. Instead, the behavior of these objects is quite distinct. Some objects are always performing some kind of activity, others have a very short period of activity, and others perform the same activity in a repetitive fashion.

The evolution of a *performer*'s activity is depicted as time intervals representing periods when the object has an active state. This representation allows an observer to carry out qualitative temporal reasoning about the patterns of VR Objects behaviors, identifying frequency, durations, and synchronization between objects' activities (Blok *et al.* 1999). A model that captures only the activity-related characteristics of an object, however, is not rich enough to model all the semantics of such objects in a VR environment. In such an environment it is also important to be aware of the visibility and existence of the object. In current VR data models, the evolution of an object's visibility and existence states are not modeled. The existence of the object is almost always assumed (Luttermann and Grauer 1999) and is strongly related with its visibility.

Visibility is the characteristic of an object that determines if an observer can see the object. Based on its visibility, an object can be classified as *visible*, *invisible*, or *non-visible*. A *visible* object is an object that the user can see. An *invisible* object is an object inside the field of view of the observer that the observer cannot see, although there are no obstacles between the object and the observer. *Invisible* objects reflect either an intrinsic characteristic of the object or a characteristic that can be manipulated by observers for analysis purposes. A *non-visible* object, on the other hand, is an object outside the field of view of the observer, behind a visible object, or so distant that it cannot be seen. In this thesis we deal only with *visible* and *invisible* objects. The *non-visible* state of an object can only be verified at run-time considering both the observer and the object's positions, therefore it is not something that is worth modeling.

Existence refers to the physical presence or occurrence of an object or, for conceptual objects, the belief in or perception of an object (Hornsby and Egenhofer 2000). Based on its existence state, an object can be classified as *non-existent* or *existent*. The *non-existent* state describes the case where an object does not exist at the instant of observation. The object has been destroyed, will be created, or simply does not exist in the physical world at any time. The existence of an object is almost always associated with the notion of its appearance (i.e., the graphical realization of the object). However, the existence of an object does not imply a particular graphical realization and vice-versa (Egenhofer and Hornsby 1998). Some objects do not have an associated visual representation. Alternatively, an observer can manipulate object visibility, turning the object into an invisible object, for instance for analysis purposes. Other objects do not exist at the instant of observation (e.g., a building that will be constructed in the future). The visualization of non-existing objects is likely to occur as the result of a temporal manipulation of the object existence or activity, discussed later in this chapter.

7.2 Taxonomy of VR Objects

Existence, activity, and visibility are orthogonal characteristics of a VR Object. The combination of these three characteristics gives rise to eight possible statuses for an object at a certain time. Objects with different combinations of characteristics carry different semantics in the model. We classify each object in a VR environment based on the combination of these characteristics (Figure 7.1).

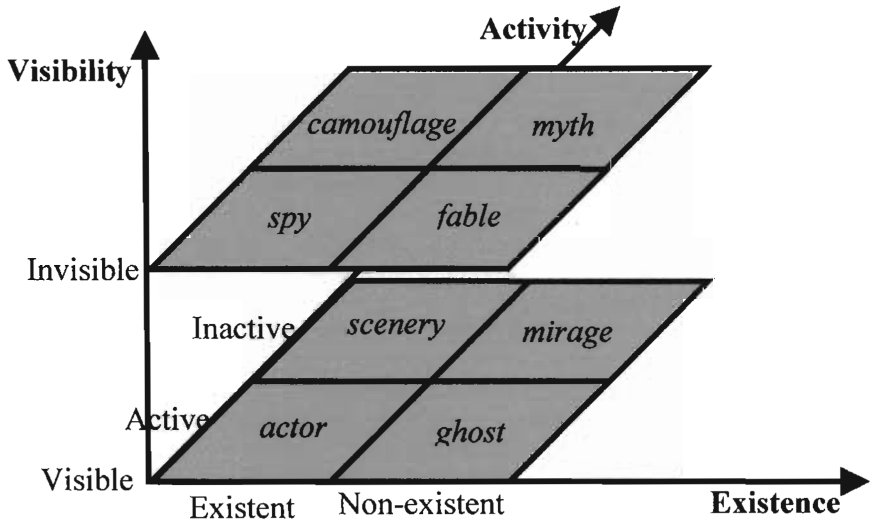


Figure 7.1. Classification of VR objects accordingly to their existence, activity and visibility characteristics.

7.2.1 Existent VR Objects

Actor, *scenery*, *spy*, and *camouflage* objects are elements of the subset of existing VR Objects. These objects exist and are performing the activity, if any, that they are supposed to be performing at the instant of the observation.

Actor is an object with an existent, active, and visible state. An *actor* represents an existent and visible object performing its associated activity, for example, a car traveling between two cities in a non-stop trip. *Scenery* refers to an object that is existent, inactive, and visible. A car that is stopped at a gas station, or a building that always maintains the same size and appearance are examples of *scenery* objects. The semantics associated with *actor* and *scenery* require their visual realizations and that both the observer and other objects be sensitive to the presence of these objects (e.g., an *actor* or a *scenery* object can block the path and the sight of the observer).

Spy and *camouflage* refer to an existent and invisible object. A *spy* represents an object during the performance of its associated activity, while a *camouflage* represents an object during periods of inactivity. The semantics associated with *spy* and *camouflage* objects are similar to the semantics associated with *actor* and *scenery* objects, except that *spy* and *camouflage* objects are invisible. *Spy* and *camouflage* represent objects that do not have a graphical representation or objects that are intentionally hidden by an observer to facilitate the analysis of the environment.

Consider, for instance, a scenario where an urban traffic analyst explores a virtual environment representing a city. Streets, buildings, and vehicles compose the virtual environment. The analyst knows ahead of time that the traffic on some roads will be significantly affected due the construction of a new business complex. The goal of the analyst is to analyze the projected flow of vehicles in some critical intersections of the city. The analyst can walk or fly through the multi-dimensional representation of the information, observing the dynamic behavior of vehicles via animations. The evolution of the state of the objects reveals the semantics of each object over time. Some objects are always *scenery* (e.g., roads and buildings), others are always *actors* (e.g., a bus that runs continuously in the environment), and others alternate their state between *actor* and *scenery* (e.g., a car that parks along the road for a while and then drives off away). As far as the analyst is concerned, vehicles and roads are essential objects. Buildings, on the other hand, have a secondary role in the environment, at least for the purpose of a traffic analysis. The presentation of the buildings provides a more realistic representation of the city and can be eventually used as landmark, helping the navigation of the observer in the environment. During the exploration of the dynamic environment, however, some buildings can steal the attention of the analyst or block the visualization of an interesting configuration of the traffic's flow. Thus, the analyst can hide a group of buildings to clean up his/her field of view facilitating the observation of phenomena of interest. These buildings, originally *scenery* objects, are transformed into *camouflage* objects when hidden by the analyst. A *camouflage* building permits the visualization of every object behind it, while the environment remains sensitive to the physical presence of this object (i.e., a *camouflage* building still blocks the path of the analyst). One can conceive also a situation where the analyst decides to hide certain class of vehicles (e.g. buses or trucks). These hidden vehicles become *camouflage* or *spy* objects, depending on their current state, *scenery* or *actor*, respectively. The semantics associate with a *spy* vehicle, for instance, is that the vehicle continues to change its position despite the fact that it cannot be seen, and, eventually, can collide with or block the passage of another vehicle in an intersection of the roads.

A model that has only existent objects is not yet rich enough to represent the semantics of all objects in the VR environment. The buildings of the business complex, for instance, do not yet exist as well as some of the vehicles representing the projected flow of vehicles. Including additional semantics that treat non-existent versions of objects gives us an additional group of VR Objects.

7.2.2 Non-Existent VR Objects

Ghost, *mirage*, *fable*, and *myth* compose the non-existent subset of VR Object statuses. *Ghost* refers to an object with a non-existent, active, and visible state, while *mirage* is an object with a non-existent, inactive, and visible state. *Ghost* and *mirage* share some semantic characteristics with their existent versions (i.e., *actor* and *scenery*, respectively); the only difference is that the observer and other objects are not sensitive to the physical presence of *ghost* and *mirage* objects. *Ghost* and *mirage* are useful for visually comparing existent and non-existent objects while avoiding the interference of the non-existent ones in the environment.

Consider, for instance, the example of the traffic simulation in the neighborhoods of the new business complex. The existent buildings are modeled as *scenery* objects, while the status of the buildings of the business complex are modeled as a *mirage* objects. The semantics associated with *mirage* buildings requires their visual realizations, but does not impose physical constraints on the environment (e.g., the observer can walk through *mirage* buildings). The vehicles representing the projected flow of vehicles become a *mirage* or *ghost* objects, depending on their activity's state. The semantics of a *ghost* object also does not impose physical constraints on the environment, but requires the visualization of the object while it performs its associated activity.

Fable and *myth* represent the non-existent versions of *spy* and *camouflage* objects or the hidden version of *ghost* and *mirage* objects. *Fable* is a non-existent, active, and invisible object, while *myth* refers to a non-existent, inactive, and invisible object. The semantics associated with *fable* and *myth* objects are that the environment and the observer are not sensitive to the presence of these objects and the observer cannot see them. For example, if the traffic analyst decides to explore the actual configuration of the environment, he or she can hide the *mirage* and *ghost* objects, transforming them into *myth* and *fable* objects, respectively. This kind of manipulation generates an environment where only existent objects are visible.

7.3 Semantics Operations

The status of VR Objects can change over time. Some objects constantly change their status, others change it only a few times, and others have a particular status during their entire lifetime. The change from one status to another is accomplished through a set of semantic operations (i.e., `appear`, `disappear`, `activate`, `deactivate`, `kill`, and `resuscitate`). These operations can be arranged in three different groups (Figure 7.2). Operations of each group act only over a specific characteristic of the object (i.e., *visibility*, *activity*, or *existence*). Each group has exactly two operations where one operation is the inverse of the other. The domain of each operation is a subset of the set of VR Objects' statuses (e.g., actor, scenery, spy, and camouflage). The range of each operation is the complimentary subset of its respective semantic domain (e.g., ghost, mirage, fable, and myth).

The visibility-related operations are `appear` and its inverse, `disappear`. `Appear` and `disappear` model the transition of the visibility state of an object to visible or invisible, respectively. These operations direct the application to start or to stop rendering the graphical representation of the object (Figure 7.2a).

`Activate` and `deactivate` form the group of activity-related operations. `Activate` indicates that the object starts to perform its associated activity. After the `activate` operation, the object becomes an active object, implying that the value of at least one of its attributes for which the observer can sense the variation will change at the next instant. The `deactivate` operation indicates that the object stops performing its associated activity. Figure 7.2b shows the result of these operations over their respective semantic domains.

The existence-related operations are `kill` and `resuscitate` (Figure 7.2c). `Kill` models the case where the object ceases to exist. `Resuscitate` indicates that an object comes to an existent state.

The domain and range of each operation limit possible transitions between different statuses. Thus, it is not possible to model the transition from a myth to an actor using a single operation. The combinations of the semantics operations, however, can model all possible transitions among elements of the set of VR Objects statuses. For instance, a composition of operations (denoted by •) that turn a myth into an actor could be:

activate • appear • resuscitate(*myth*) = *actor*

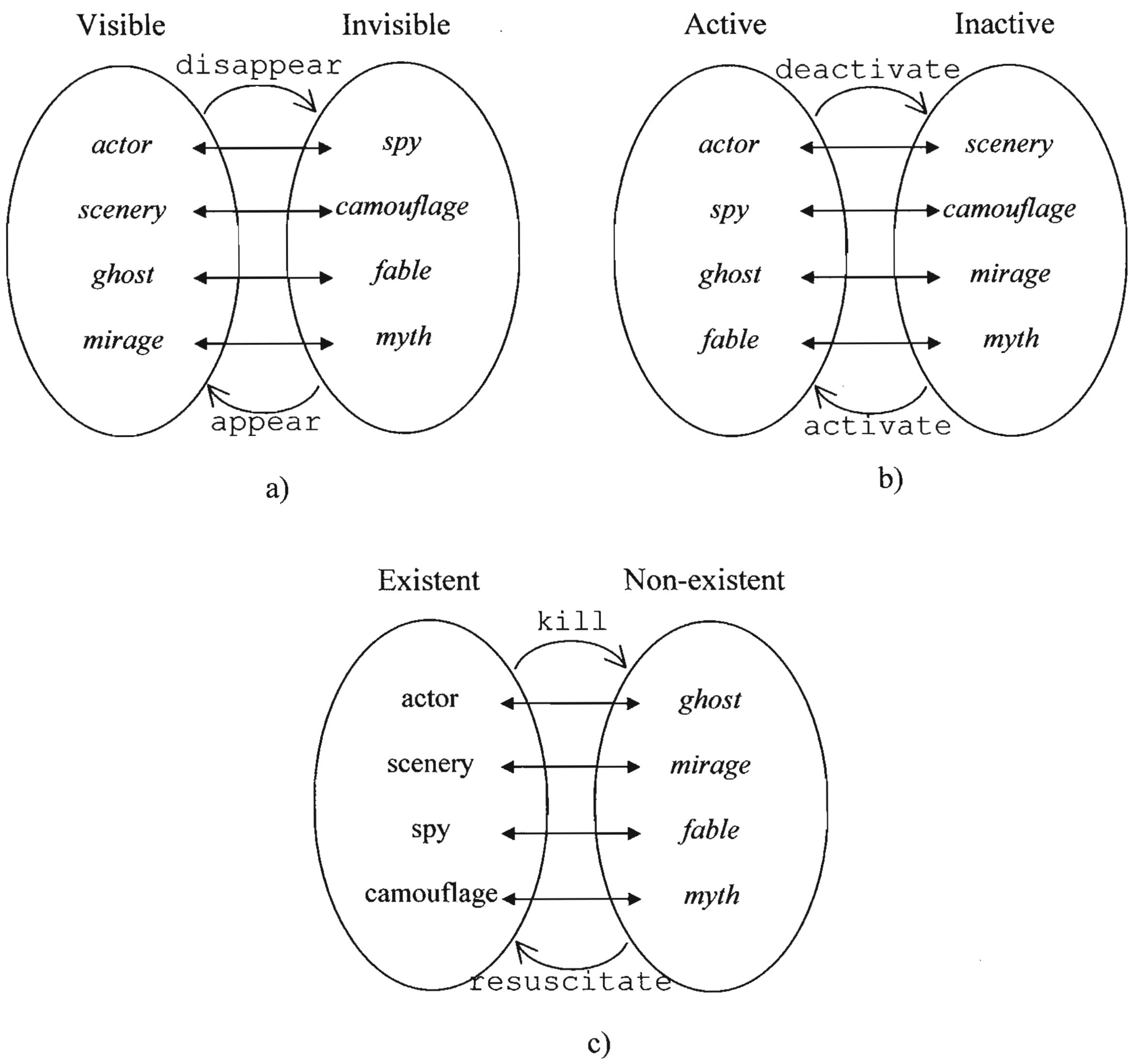


Figure 7.2. Mappings of semantics operations: (a) visibility-related operations, (b) activity-related operations, and (c) existence-related operations.

7.4 Evolution of VR Objects' Semantics

In order to keep track of a VR Object's semantics throughout its lifetime, we introduce *History*. *History* is a data abstraction that models the evolution of the status of every *VR Object* with respect to existence, visibility, and activity characteristics.

Figure 7.3 shows the relationships of the class *History* with other data abstractions of the animation model. Each *VR Object* has its own *History*. *VR Object* is an abstraction that represents the geometry and appearance of all objects in the environment. The class *Performer* represents a special type of *VR Object*. Object *performer* has associated *Course of Actions* and *Course of Actions Interval* objects modeling the spatial and temporal characteristics of its behavior. There exists a dependency between the temporal characteristics of an object *performer* and its *History*. This dependency is due to the fact that the periods of activity of the *performer* define the activity-related status of the object.

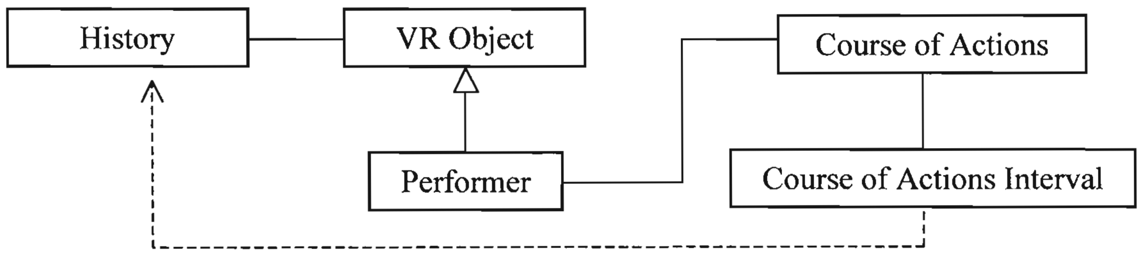


Figure 7.3. Class diagram with the History and other related data abstractions of the animation model.

The functionality of the class *History* is completely defined by its constructor method and the set of semantics operations (i.e., `appear`, `disappear`, `activate`, `deactivate`, `kill`, and `resuscitate`). The constructor method instantiates a *History* object a *myth* status. If it is not the case, the initial status of the object can be changed through an appropriate combination of semantic operations. Semantic operations model the evolution of an object's status over time. Thus, each operation has as argument the instant in time when it is introduced in the object's history. If the time attribute is null, it means that the semantic operation is changing the initial status of the *VR Object*; otherwise, the operation is changing the status of the VR Object from that instant on.

The representation of the history of an object is accomplished by an attribute that stores a list of all semantics operations and the instant when these operations occur in the model. Thus, a status of a VR object can be retrieved at any instant based of the elements of this list. Figure 7.4 shows the graphical representation of *history* of a *VR Object*. The *VR Object* is of the type *performer*, which means that it has an associated behavior. The graphical representation of such a behavior is also shown in Figure 7.4 to highlight the dependency between these two representations.

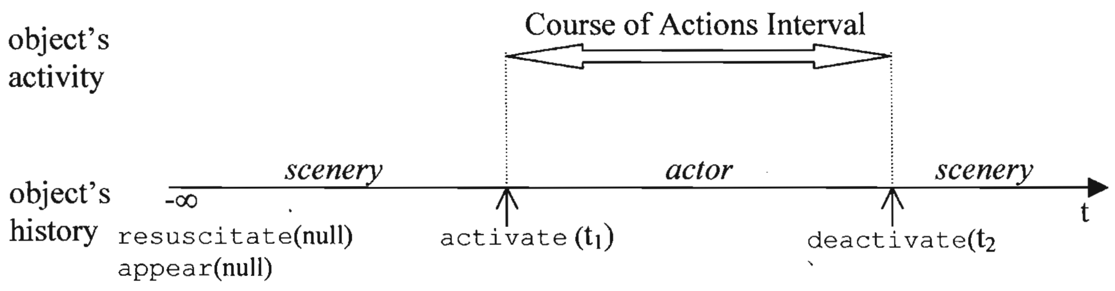


Figure 7.4. Graphical representations of the History and Course of Action Intervals of a performer object.

The graphical representation of the object's *history* shows the evolution of the objects' statuses over time. The object's *history* is instantiated as *myth*, but the operations `resuscitate` and `appear` with a null argument changes the initial status of the object to a *scenery* object. This object keeps the *scenery* status until the first occurrence of a semantic operation. At the instant t_1, an `activate` operation changes the status of the object to an *actor*. This operation reflects the fact that the object has an associated activity that starts at the instant t_1 (e.g., if the performer object is a car, the car starts to move). At the instant t_2, a `deactivate` operation transforms the object back into a *scenery* object, reflecting the end point of the *course of action* (e.g., the car stops its movement). The object remains as *scenery* indefinitely since there are no other operations to be performed in the model. From the instant t_2 on, the car can be seen parked somewhere in the virtual environment.

7.5 Modification of VR Objects' Semantics

Some important manipulations of VR objects involve the redefinition of the time when the object performs its associated activity. Phenomena that have occurred at different times can be manipulated and observed at the same time, facilitating the comparison of their behavior. Allowing observers to interfere with the original flow of the objects' dynamics, however, requires the modification of an objects' *history* in an automatic and consistent way.

Consider, for instance, a scenario where an analyst explores a virtual environment involving a storm and a ship traveling from Boston, Massachusetts to Portland, Maine. The storm system developed a path somewhere between Boston and Portland. Figure 7.5 depicts the graphical representation of the storm and ship activities (i.e., their associated *course of actions intervals*) and the evolution of their statuses (i.e., their *histories*). The exact path and size of the storm and the path of the ship are modeled by their respective *courses of action.*

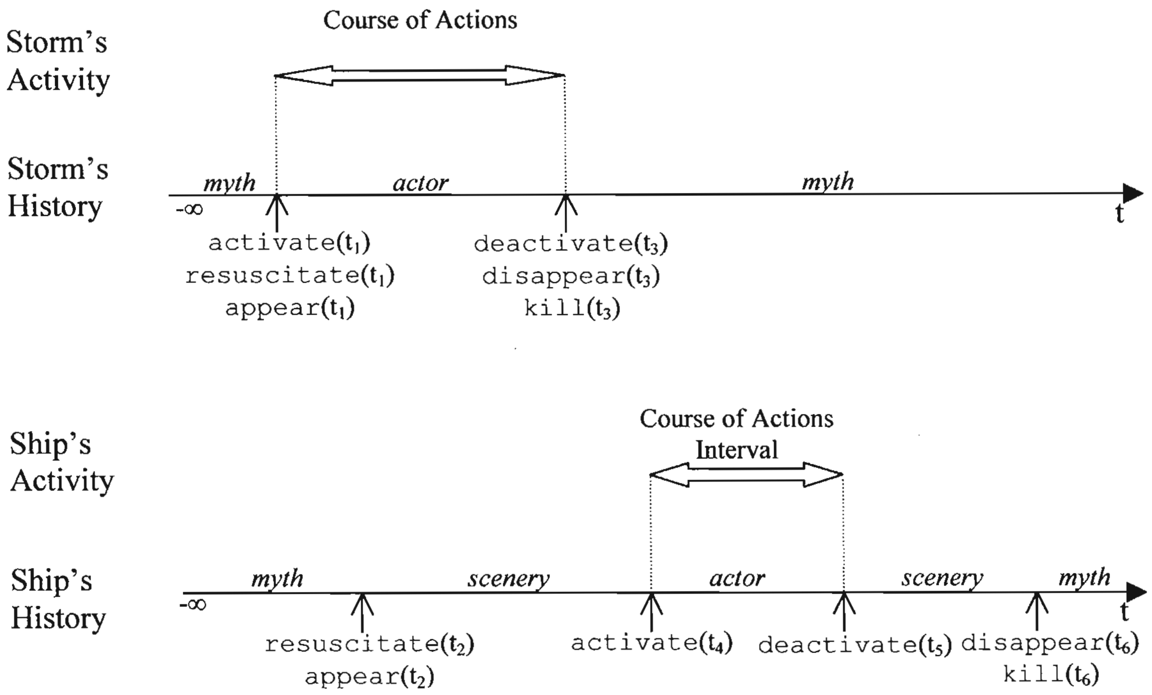


Figure 7.5. Graphical representation of the original configuration of the course of action and history of the storm and ship.

The *history* of the storm reveals that the object starts as a *myth* and remains with this status until the instant t_1, when a sequence of operations (i.e., `activate`, `resuscitate`, and `appear`) changes its status to an *actor*. The semantics associated with an *actor* object indicates that the object exists, is visible and is performing the activity modeled by its course of action (e.g., the storm is changing its position and size). The storm remains as an *actor* until the instant t_3, when another sequence of operations (i.e., `deactivate`, `kill`, and `disappear`) transforms it back into a *myth,* modeling the end of the storm. The simultaneous occurrence of one operation of each group in the *history* of the storm is not a coincidence. A storm is a phenomenon in which the visibility, existence and activity states are strongly related. Thus, is reasonable to think that a storm object has a *history* with only *myth* or *actor* statuses.

The *history* of the ship shows that the object starts as a *myth*, implying that the environment and the observer are not sensitive to the presence of the ship and the observer cannot see it. The ship retains this status until the instant t_2, when a sequence of operations (i.e., `resuscitate` and `appear`) transforms it into a *scenery* object. The semantics associated with a *scenery* object is that the object can be seen and the environment is sensitive to the presence of the object. The position of the ship is a dock at the port of Boston. This information is available as the initial value of position's attribute of the ship. The ship remains docked at Boston until the instant t_4, when the occurrence of an `activate` operation transforms its status into an *actor*, indicating that the ship starts its trip to Portland. A `deactivate` operation at the instant t_5 denotes that the ship finishes its trip and becomes a *scenery* object at the Portland's port (i.e., the ship no longer is performing an associated activity). At the instant t_6, another sequence of operations (i.e., `kill` and `disappear`) turns the ship's status into a *myth*. The ship keeps the *myth* status indefinitely. The status of the ship as a *myth* in the start and end of its *history* can be interpreted as a lack of knowledge about the ship during these periods.

The analysis of the storm and ship *history* discloses that the ship was safely anchored during the occurrence of the storm. However, an observer can manipulate the original flow of the object behavior and explore the virtual environment with hypothetical configurations. Consider, for instance, a situation where the observer makes the ship start its trip at the same instant as the formation of the storm. This configuration can be accomplished with an operation that changes the start point of the ship *Course of Actions*

Interval (e.g., the operation `makeStartTogether`). In such a manipulation the history model of the storm is not affected but the evolution of the semantics of the ship needs to be updated accordingly to reflect the new spatio-temporal configuration of the object. Figure 7.6 shows the process of manipulating the ship's activity and the resulting history of such an object. For the sake of clarity semantic operations are represents by the first three initials in the figure.

The modified *history* of the ship shows that the ship starts as a *myth* and keeps its status until the instant of the formation of the storm. At this instant the ship appears at the port of Boston and immediately starts its trip to Portland as a *ghost* object. The *ghost* semantic implies that the ship can be seen performing its associated activity, but its presence does not interfere with the environment. The ship finishes its trip at the instant t_n, becoming a *mirage* at the port of Portland. The *mirage* semantics implies that the ship still can be seen and its presence does not interfere with the environment, but there is no activity associated with the ship. The *ghost* and *mirage* semantics associated with the ship reflect the fact that the ship is not supposed to be doing its trip or to be docked at the Portland's port during these periods. The ship remains as a *mirage* until the instant t_5 when it becomes a scenery object at the Portland's port. After the instant t_5, the ship continues with its original history.

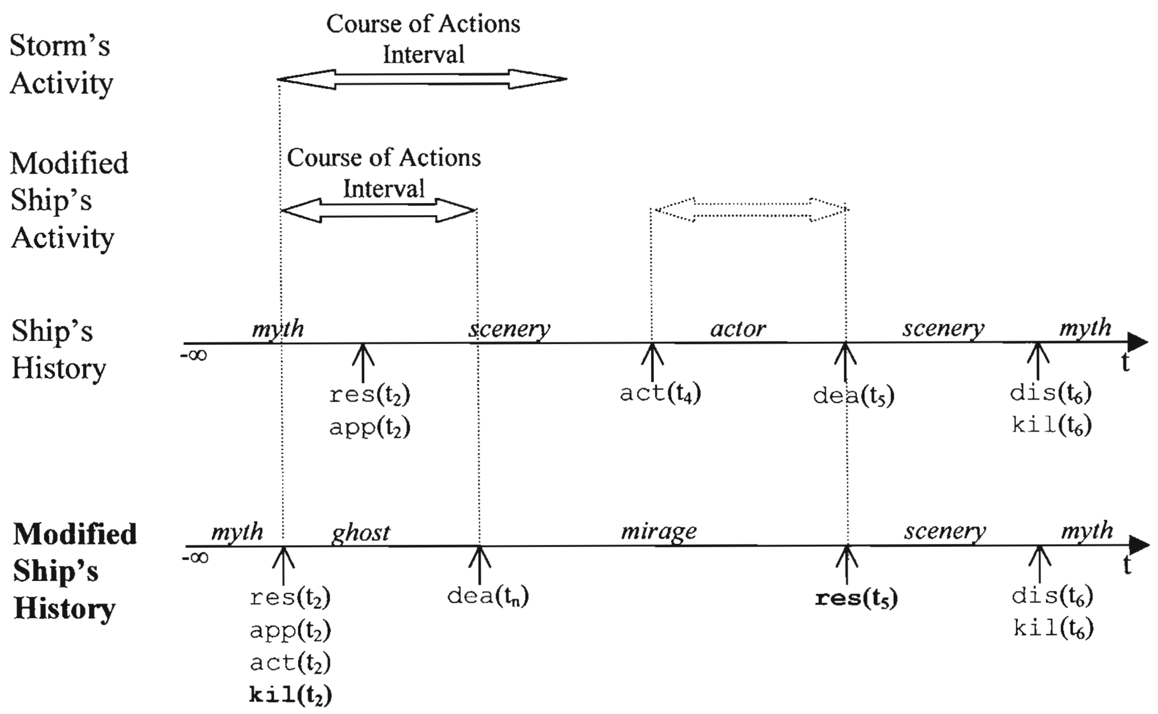


Figure 7.6. Manipulation of the temporal configuration of the ship's activity and the modified version of the ship's history.

The modified history of the ship illustrates the rationale used to update the evolution of VR objects' statuses when its activity is manipulated. In this case, the course of action of the ship is translated to a new position in the temporal domain. The translation of the *course of action* implies the reposition of the `activate` and `deactivate` operations in the object's *history*. The temporal displacement between these two operations is kept the same as the original configuration (i.e., t_n-t_1=t_5-t_4). This constraint guarantees the original pace of the ship's trip. All operations occurring in the ship's *history* between the original and the new position of the `activate` operation are positioned at the same time (i.e., at the new position of the `activate` operation) forming a long sequence of operations. The order of the operations in the sequence reflects the order that they occur in the original model. A `kill` operation is added at the end of this sequence to represent the fact that the object's activity was manipulated. Finally, a `resuscitate` operation is introduced at the original position of the `deactivate` operation, restoring the original semantics of the object.

A graphical representation of the *history* the *VR Objects* can be presented in a graphical user interface together with commands representing semantic operations. Such an interface allows the user to manipulate the status of the objects by changing one of its semantics characteristics. Giving the observer the ability to manipulate such characteristics is not treated in this thesis. We used the modeled semantics of the objects to capture types of interactions that may occur among objects and between an observer and the objects in the environment and to modify the status of the objects due to a manipulation of the temporal configuration of the animation.

7.6 Summary

This chapter introduced a model to represent VR Objects' semantics. Based on this model, a new classification of VR Objects was presented, and the semantics associated with each class of object was described. VR Objects were categorized with respect to existence, activity, and visibility. A set of operations, acting upon individual characteristics of the object, models the object's semantics. In order to capture an object's changing semantics, we introduced *History*, an extension of the animation model that represents the evolution of the semantics of VR Objects over time. Finally, this chapter outlines the rationale used to modify the evolution of an object's semantics during the manipulation of the object behavior.

The next chapter introduces a graphical user interface for the virtual exploration of the animation model and discusses some interface's operations.

CHAPTER 8

PROTOTYPE IMPLEMENTATION AND ASSESSMENTS

The *Virtual exploration of animations* is a framework composed of abstract data types and a user interface that allow non-expert users to control, manipulate, analyze, and present objects' behaviors. Previous chapters discussed the conceptualization and functionalities of a model for *virtual exploration of animations*. This chapter introduces a graphical user interface for the model and also assesses the model's expressive power.

The next section introduces a simple example. Subsequently, we present a prototype implementation and discuss the functionality of a representative subset of the interface's operations. The prototype was written in the programming language Java (Horton 1999) and relies on the support of the Java 3D application programming interface (Java-3D 2004) to render four-dimensional information. Finally, we use the example introduced in the first section as a reference in the computation of the number of different versions of the animation that can be accomplished by a user with the proposed model and with a model that extends the set of operations of a VCR control to each object's behavior in the environment. The analysis of these results is used to test the hypothesis that is:

The model of virtual exploration of animations produces views of an animation that cannot be accomplished by any combination of operations of the individual VCR model.

8.1 An Example

The example discussed here was conceived to highlight all features of the model for *virtual exploration of animations* and does not intend to represent any real world scenario. In an abstract way, however, this example can be compared with a simulation of the movement of equipment on a factory floor.

Our virtual world is composed of the floor of a square room and five moving objects (Figure 8.1). The moving objects are a *cone*, a *cylinder*, a *box*, and two *balls*. The *cone* follows a triangular path in the room. The object starts to move at the instant t=0s and takes 12 seconds to complete its entire behavior. The *cone* returns to its original position at the end of the movement and rests for 3 seconds before starting to move again. The *cone* repeats the same behavior in a cyclic fashion indefinitely.

The *cylinder* object also moves in a triangular path in the room. The spatial and temporal characteristics of the cylinder's movement, however, are different from those of the *cone*. The trajectory of the *cylinder*, for example, has only some spatial locations that coincide with the trajectory of the *cone*. These locations are potential points of contact between the *cone* and the *cylinder*. The temporal characteristic of the *cylinder*'s movement also follows a cyclic behavior. The *cylinder* starts to move at the instant t=0s but takes only 8 seconds to complete its movement. The *cylinder* rests for 2 seconds before starting its movement again.

The *red* and *yellow balls* move from the center to the border of the room in a straight line. The balls start their movement at the instant t=0s, and since the duration of their movements is the same, they finish their behavior together at the instant t=3s. There exists a temporal constraint in the model between the *red* and *yellow balls* (i.e., the *red ball* always starts when the *yellow ball* starts to move). Thus, changing the instant when the yellow starts to move causes the changing of the instant when the *red ball* starts to move as well.

The *box* also moves in a straight line, but its trajectory is perpendicular to the trajectories of the balls. The temporal characteristic of the movement of the *box* is that the *box* starts its movement when the balls stop moving. There is no temporal constraint in the model representing this fact. The box takes 2 seconds to complete its movement and stops somewhere along the path of the balls.

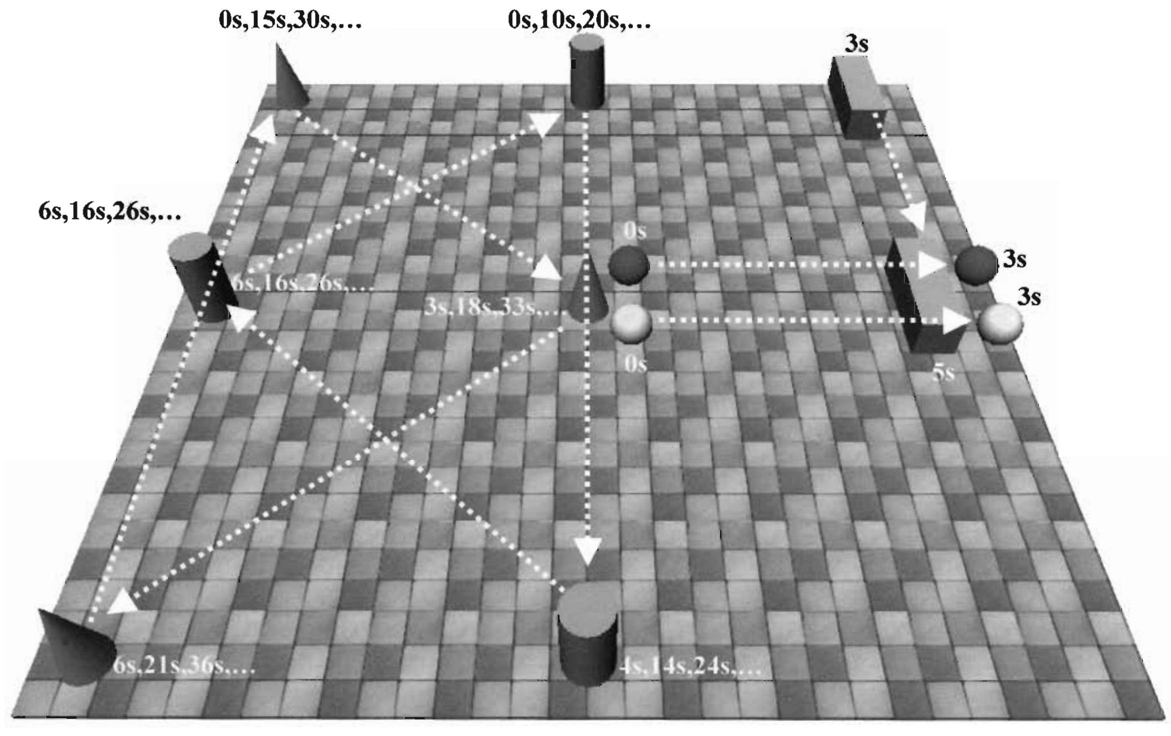


Figure 8.1. Trajectories and key states of the objects moving in the room.

8.2 A Possible Graphical User Interface

The model for *virtual exploration of animations* requires two graphical user interfaces: an *animation editor* and an *animation browser*. The *animation editor* presents the modeled configuration of objects' behavior and their temporal representations. This interface allows the user to manipulate the temporal characteristics of object's actions and some parameters of the animation's presentation. The *animation browser* allows the user to explore the three-dimensional representation of the environment in which objects can be seen performing their behaviors. The next sections present two possible graphical user interfaces for the *editor* and *browser* of the model for *virtual explorations of animations*.

8.2.1 The Virtual Exploration of Animations Editor

The *virtual exploration of animation editor* is comprised of two main sections. The first section presents the organization of action objects and their temporal characteristics. The second section presents the operations used to manipulate the structure and the temporal characteristics of objects' behaviors (Figure 8.2).

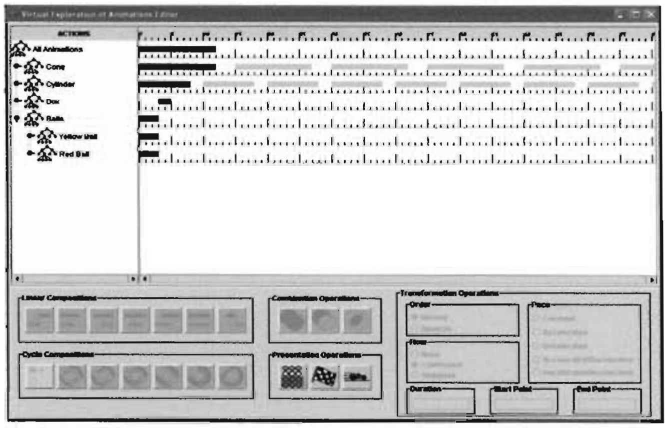


Figure 8.2. An editor for virtual exploration of animations.

In the first section of the *animation editor*, action objects are depicted in a tree-like structure and their temporal characteristics are depicted as intervals positioned along the animation timeline (Figure 8.3). The structure of the tree is defined by the application that converts information from the application domain to the *virtual exploration of animations* domain. The tree in the Figure 8.3 represents the structure of action objects of the example introduced early in this chapter.

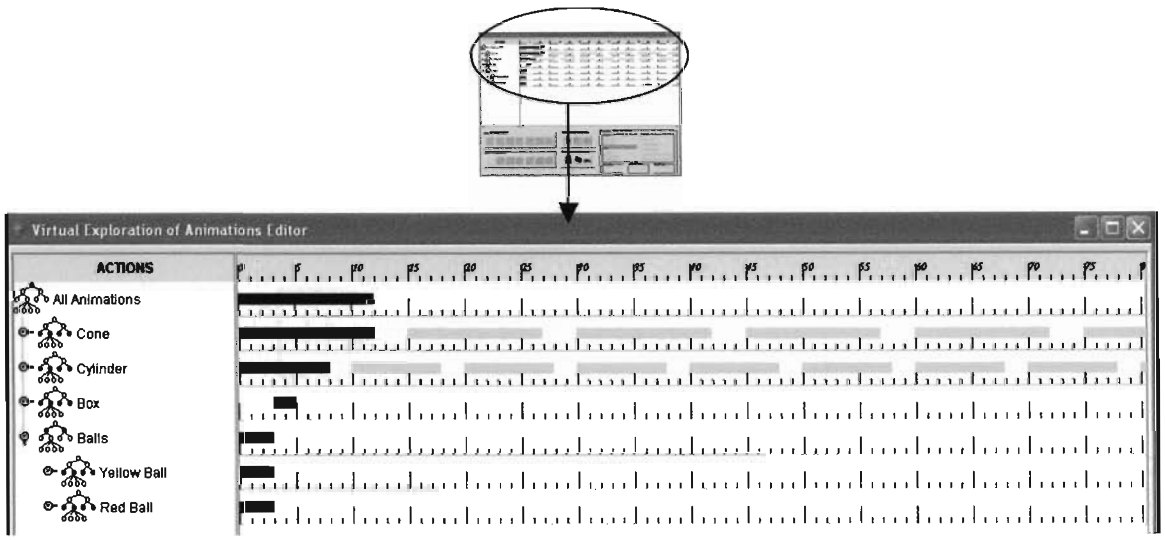


Figure 8.3. The graphical representation of the structure of action objects and their temporal characteristics.

Elements of the tree are objects of the action part of the framework for *virtual exploration of animations* (i.e., *Animations*, *Course of Actions*, and *Acts*). The root of the tree is an object of the sort *Animation* and represents the collection of all objects' behaviors in the environment. Branches of the tree are *Animations* or *Course of Actions* objects. *Animation* represents a collection of *Course of Actions* or *Animations* objects. *Course of Actions* represents the behavior of a single object. The leaves of the tree are always *Acts* objects. *Acts* objects represent the finest granularity of an object's behavior.

Since all *Course of Actions* nodes in the tree are collapsed, the *Acts* objects are not depicted in Figure 8.3.

The temporal characteristics of action objects are represented as black rectangles on the right side of the interface. Each action has an associated convex temporal interval representing the start point and duration of the behavior. If an object has a cyclic behavior, however, the representation of its temporal characteristics is slightly different. The temporal characteristics of cycles are depicted with one black rectangle and some gray rectangles. The black rectangle represents the *occurrence of reference* of the cycle. The gray rectangles represent other occurrences of the cyclic behavior. The number and positions of gray rectangles depend on the *pattern of repetition*, the *period of inactivity*, and for finite cycles, the *number of repetitions*. In the example shown in the Figure 8.3, the *cone* and the *cylinder* have a cyclic behavior of the type *from*. It means that the objects start their behavior with the *occurrence of reference* and repeat the same behavior indefinitely.

The hierarchical structure of the model imposes the constraint that any modification in the temporal characteristic of an action object be consistently propagated up and down in the tree structure (i.e., changing the start point of a *Course of Action Interval* changes the start point of all *Act Intervals* associated with the *Course of Actions* and may change the temporal characteristics of the *Animations Interval* located higher up in the hierarchy). This process is repeated until the application reaches the root of the tree (i.e., the node *all animations*). It was a design decision to not incorporate the duration of all occurrences of cyclic behaviors in the temporal characteristics of their parents in the tree structure. In this way, only the *occurrence of reference* is used in the computation of the start point and duration of the *Animation Interval* that has *Cycle Intervals* as children. This decision allows the model to extend the operations performed over convex intervals to *Cycle Intervals*.

The second section of the animation editor shows a graphical interface for operations used by an observer to manipulate the content of the animation (Figure 8.4). These operations are divided in five main groups (i.e., *linear compositions*, *cycle compositions*, *combination operations*, *transformation operations* and *presentation operations*).

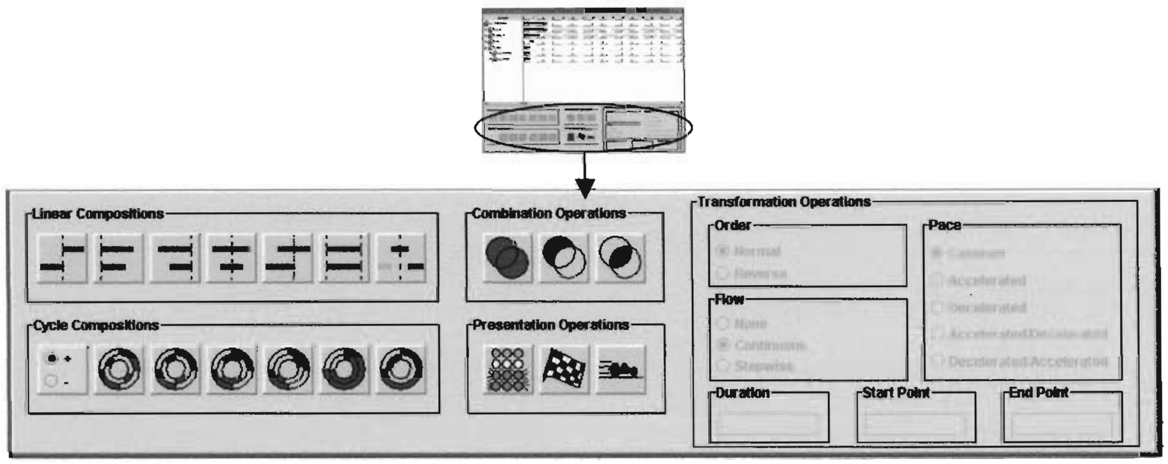


Figure 8.4. User interface representing operations to manipulate the temporal characteristics of animations.

Linear compositions, *cycle compositions*, *combinations operations*, *transformation operations*, and their effects in the outcome of the animation are discussed latter in this chapter. *Presentation operations* allow an observer to specify some parameters of the animation's presentation and link the information of the *animation editor* with the *animation browser*.

Presentation operations are `start point`, `end point`, and `play`. The operation `start point` defines an instant in the animation temporal space when the application starts to map animation times to the user time domain. This operation has the effect of defining the instant of insertion of the observer in the virtual world. The operation `end point` defines the instant in the animation temporal space when the application stops mapping *animation times* to the *user time* domain. This operation has the effect of stopping the movement of all objects in the environment from the specified instant on. In this chapter we assume that the *start point* of the animation is the instant that the first object starts to move and the *end point* is not defined, that is, the mapping from animation time to user time domain never stops.

The operation `play` aligns the present of an observer with the *start point* of the animation and directs the application to start mapping animation times to user times. This operation allows an observer to perceive objects' behaviors. The operation `play` opens a window to the virtual world. This window is presented as a browser that allows the observer to navigate in the virtual environment and explore objects' behavior. The next section introduces the browser used for the *virtual exploration of animations*.

8.2.2 The Virtual Exploration of Animations Browser

The graphical realization of the objects and the outcome of the manipulations of the animation are explored in a virtual reality environment. In such an environment an observer can walk through the three-dimensional representation of the information, interact with the objects, and examine the objects performing their modeled behavior. In our prototype we use an application that renders such pieces of information on the screen of a desktop computer. In such a computer environment, the user does not experience full immersion in the virtual world.

In order to control the position and orientation of the observer's vantage point in the environment, the browser offers a set of buttons (Figure 8.5). These buttons allow the observer to move along the principal directions (i.e., *up*, *down*, *right*, *left*, *forward*, and *backward*) and to change the direction of his or her vantage point by rotating around the x axis (*turn up* and *turn down*) and by rotating around the y axis (*turn right* and *turn left*). In this application, the xy plane always coincides with the plane of the screen and the positive direction of the z axis is pointing toward the observer.

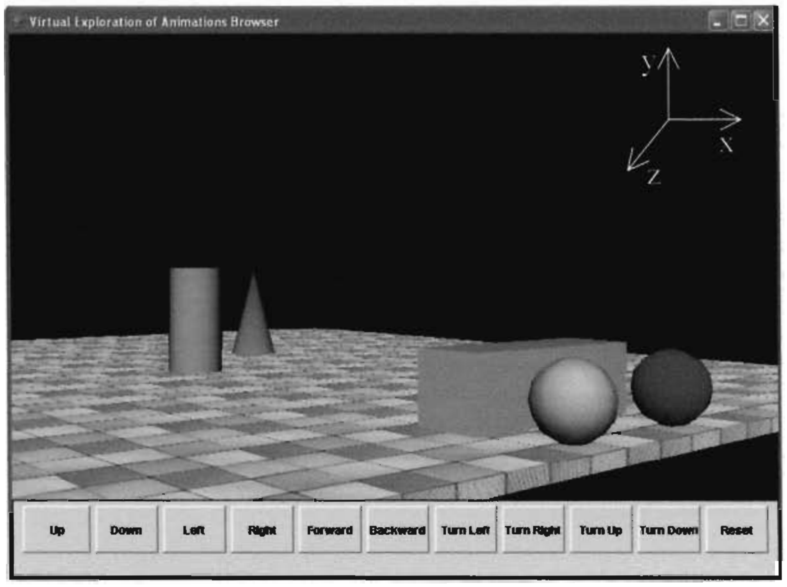


Figure 8.5. A browser for virtual exploration of animations.

One salient characteristic of the animations' browser is that all objects carry semantic information (i.e., the status of the object's visibility, activity and existence). This information is used by the application to direct the way that an object is rendered and the way it interacts with other objects in the environment and with the observer. An object with an existent status, for example, blocks the path of the observer or other objects in the environment with a similar status.

The navigational functionality of the animation browser is straightforward and does not need further explanation. The way that the application processes semantic information, however, is discussed in the next section together with operations of the *virtual exploration of animations* editor.

8.3 Animation Operations

Animation operations allow an observer to manipulate the modeled animation (i.e., information of objects' behavior converted from the application domain to the animations domain). The animation model carries information about the spatial characteristics of the behavior (e.g., the key states of an object), the temporal characteristics of the behavior (e.g., the instant when the behavior starts), temporal constraints (e.g., the fact that an object always starts its behavior when other objects are performing their activity), and semantic information (e.g., whether the object exists at the instant of observation). Among all pieces of information captured by the animation model, only the temporal characteristics of behaviors can be directly manipulated by an observer. At this stage of development, the prototype does not provide the means to introduce temporal constraints or to manipulate the semantics of the objects and the spatial characteristics of object's behavior. The application, however, uses all these pieces of information to keep the modified version of the animation coherent with the original configuration. The application, for example, does not limit the observer to generating versions of the animation where all temporal constraints are satisfied, but the application does change the semantics of the object to reflect the fact that the modified version violated a temporal constraint from the application domain. The next section discusses some animation operations and their effects on the outcome of the animation.

8.3.1 Compositions Operations

Composition operations are used to change the temporal arrangement of objects' behavior. These operations have two temporal intervals as arguments: a *reference* interval and a *target* interval.

Depending on the sort of their arguments, *composition operations* can be divided into two main groups: *linear compositions* and *cycle compositions. Linear composition* operations accept any sort of temporal interval as arguments. *Cycle composition* operations are more specific and accept only *Cycles Intervals* as arguments. The next sections introduce the representations of these operations on the graphical user interface and discuss some relevant details.

8.3.1.1 Compositions Operations Between Intervals

Linear compositions are used to position an interval (i.e., a *target* interval) with respect to the position of another interval (i.e., a *reference* interval). Depending on the operation, the start point and/or duration of the *target* interval are modified to satisfy a certain temporal relation between the *target* and *reference* intervals.

Linear compositions operations are `makeTouch`, `makeAlignLeft`, `makeAlignRight`, `makeAlignCenter`, `makeCross`, `makeEqual`, and `makeMirror`. Each operation is associated with a button in the graphical user interface (Figure 8.6). The semantic of each operation was discussed in chapter 6.

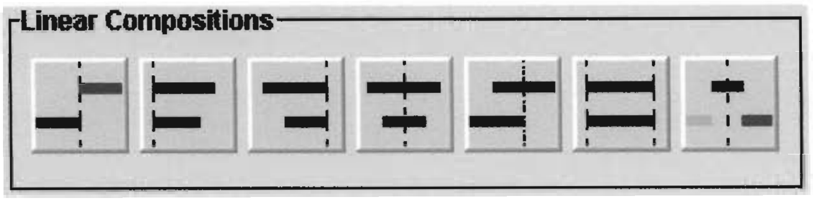


Figure 8.6. The seven linear composition operations between convex intervals.

In order to illustrate the use of *linear composition* operations, consider the movement of the *balls* and the *box* in the room. In the modeled configuration, the *box* starts to move at the same instant when the *balls* finish their movement. Since the *box* crosses the path of the *balls*, there is a potential interference between these objects.

An observer may be interested in analyzing an animation where the *balls* and the box start to move at the same time. Such a configuration can be accomplished, for instance, by changing the start point of the *balls*. One possible strategy is to select the *box*'s temporal interval as the *reference* interval and the *yellow ball*'s interval as the *target* interval and apply the *linear composition* operation `makeStartTogether`. From the knowledge base of the application domain, it is known that the *red ball* always starts its movement when the *yellow ball* starts to move. Thus, the start point of the *red ball* is also redefined by the application to satisfy the temporal constraint. Figure 8.7 shows the temporal configuration of the intervals before and after the execution of this operation.

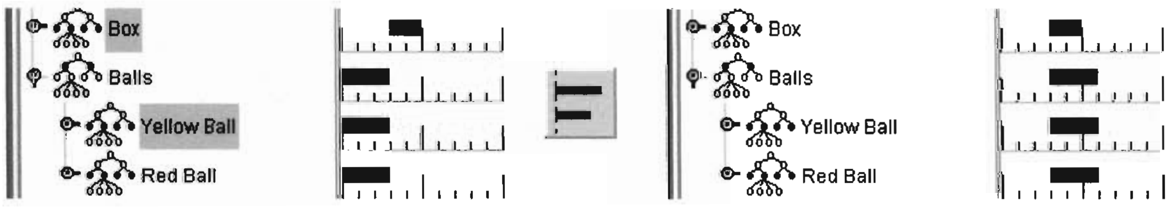


Figure 8.7. The effect in the temporal configuration of the animation by applying a makeStartTogether operation with the box and the yellow ball used as arguments.

The analyses of the graphical representation of the periods when the *box* and the *balls* are performing their activities do not allow an observer to reason about interactions among these objects. The observer does not have any spatial information about the objects' behaviors but only information about the qualitative state of the object (i.e., the object is moving or not). In this way, the observer cannot infer if the object interacts with other object simply by analyzing this kind of information. Although the animation model does have all necessary information that allows an application to anticipate any kind of interaction, we decide to leave to the observer the task of exploring the environment and verifying all interactions among the objects. For example, by exploring the dynamic environment with the *balls* and the *box*, the observer can verify that the *box* blocks the path of the *balls*; that is, due to a collision with the *box*, the *yellow* and the *red balls* stop their modeled behavior sooner than expected. Figure 8.8 shows some snapshots of representative instants of the animation. The vantage point of the observer was positioned to facilitate the analysis of the behavior of the objects of interest. During the first 2 seconds of the animation only the *cone* and the *cylinder* objects are moving in the environment. At the instant t=3s, the *balls* and the *box* start to move. By the instant t=5s the *box* finishes its movement but the *balls* are still on their way to the border of the room. Between the instants t=5s and t=6s the *balls* collide with the *box* and finish their movement without reaching their final destination.

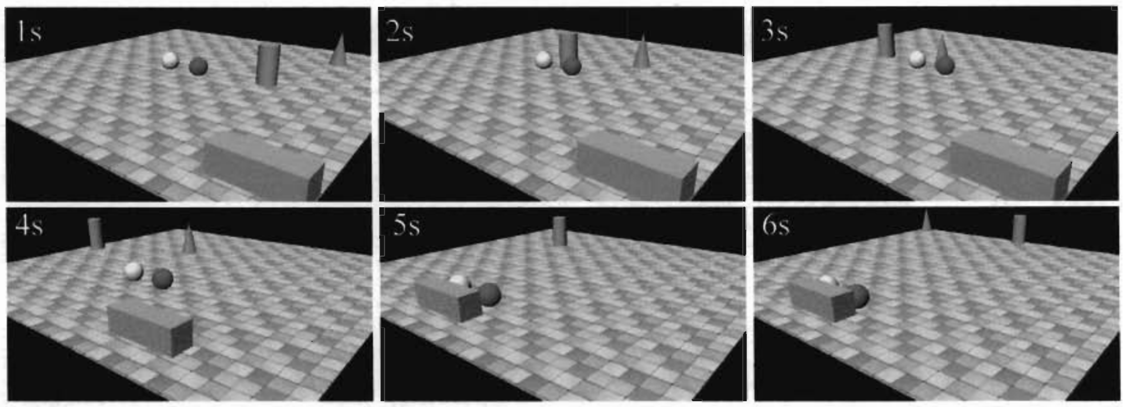


Figure 8.8. Snapshots of an animation where the balls and the box start to move at the same time. The balls collide with the box and stop their modeled behaviors.

The outcome of the animation in the previous example would be different if an observer chose the *red ball* as the target *interval* instead. By doing that, we assume that the observer is intentionally violating a temporal constraint introduced in the model. Thus, the start point of the movement of the *yellow ball* remains the same and only the start point of *red ball* is redefined (Figure 8.9).

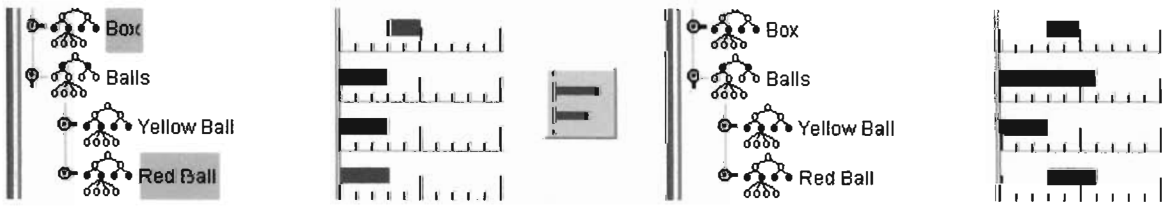


Figure 8.9. The effect in the temporal configuration of the animation by applying a makeStartTogether operation with the box and the red ball used as arguments. This configuration violates a temporal constraint introduced in the model.

In order to capture the fact that a temporal constraint was violated, the application change the status of the *red ball* during the time period when the object is violating the temporal constraint. Thus, the status of the *red ball* changes from an existent to a non-existent object. Figure 8.10 shows representative snapshots of this animation with the modified status of the *red ball*. In this version of the animation, the *yellow ball* starts to move in the beginning of the animation and the *red ball* becomes a *mirage* in the environment (i.e., an object with a *inactive*, *visible*, and *non-existent* state). At the instant t=3s the *red ball* and the *box* start to move. Since the red ball still has a *non-existent* status, it moves as a *ghost* object. At the instant t=5s the *yellow ball* and the *box* had finished their behaviors but the *red ball* is still moving as a *ghost* object. Between the instants t=5s and t=6s the *red ball* passes through the box and reaches its destination. Although the *box* has an existent status, the *box* is not able to block the passage of non-existent objects. From the instant t=6s on, the red ball recovers its existent status and becomes a *scenery* object in the environment (i.e., an existent, inactive, and visible object). We have decided to present non-existent objects with a semi-transparent graphical representation. This mechanism gives the observer visual feedback about the existent status of the object.

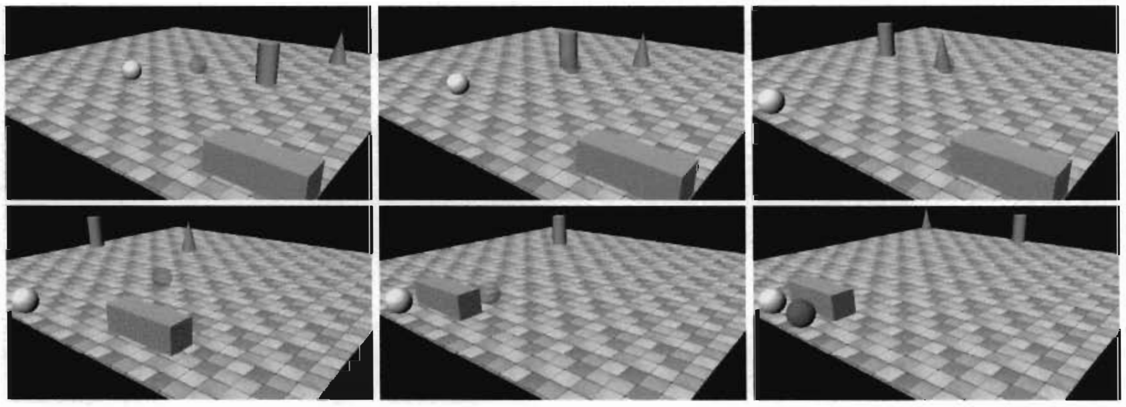


Figure 8.10. Snapshots of an animation where only the red ball start to move with the box. The red ball has a non-existent status during the period when the temporal constraint is violated.

All composition operations but `makeEqual` preserve the duration of the target interval. For these operations, the application needs only to deal with the existent status of objects when changing the position of *reference* interval violates a temporal constraint introduced in the model. If the duration of the *target* interval changes, however, the existent status of the object always changes. In the model for *virtual exploration of animations* the duration of the interval is considered a implicit temporal constraint. The rationale used to modify the status of an object when the duration of its behavior is discussed later in this chapter.

Since composition operations accept any sort of temporal intervals, these operations can also be used to modify the temporal characteristics of *Cycles Intervals*. When at least one of the arguments is of the sort *Cycle Interval*, the *reference* and/or the *target* intervals of the operation are always the *occurrence of reference* of the cyclic behavior.

8.3.1.2 Compositions Operations Between Cycles Intervals

Cycle compositions are operations that modify the temporal characteristics of *Cycle Intervals*. Operations of this kind are performed only when both of the intervals used as arguments are cycles. Depending on the semantics of the operation, the start point, duration of the period of activity, and/or the duration of the period of inactivity of the *target* cycle is modified to satisfy a certain temporal configuration among all occurrences of the cycles.

The group of *cycle compositions* operations is composed of ten operations: `makeConcurrent`, `makeAlternate`, `maximizeRelation`, and `minimizeRelation`, where `Relation` can assume one of the following values:

disconnections, touchings, crossings, or containments. The graphical user interface provides a set of buttons representing such operations (Figure 8.11). The first button in the group allows an observer to select between operations that maximize or minimize the number of incidences of certain temporal configurations between occurrences of the cycle. The following four buttons in the group represent two operations each. These buttons can either represent a maximizeRelation operation or a minimizeRelation operation. The relations are in the order: disconnections, touchings, crossings and containments. The last two buttons represent the operations makeConcurrent and makeAlternate, respectively.

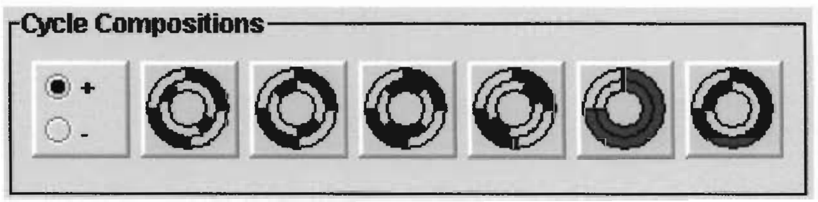


Figure 8.11. The group of composition operations between cycle intervals.

The main difference among operations that maximize or minimize the incidence of a certain temporal configuration and the operation makeConcurrent and makeAlternate is that in the former case the *target* cycle preserves the duration of its periods of activity and inactivity and in the later case both durations are redefined.

In order to illustrate the use of operations of the group of cycle compositions, consider the original configuration of the animation of the *cone* and the *cylinder* object in the room. By exploring this version of the animation, an observer can verify that the movements of the *cone* and the *cylinder* have two spatial locations where they almost collide. The closest distance between these objects occurs at the instants t=4.7s and t=16.7s. At these instants the distance from center to center of each object is only 1.5 units of distance. To give an idea of the magnitude of such a distance, the radius of the *cylinder* and the *cone* have 0.5 unit of distance.

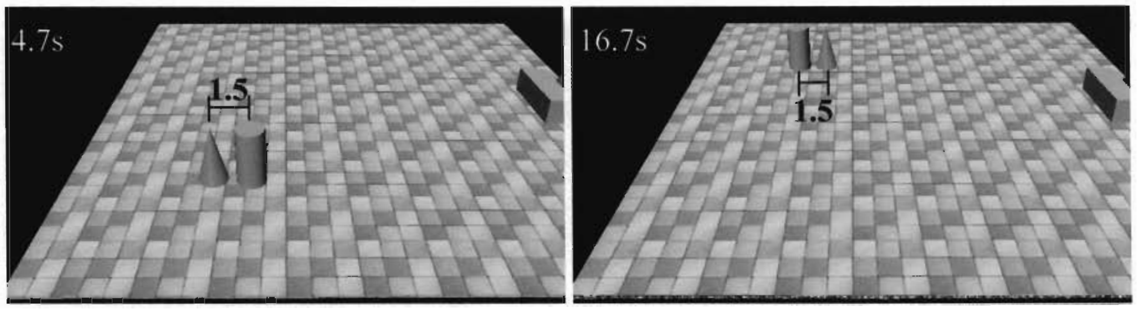


Figure 8.12. Snapshots of the instants in which the cone and the cylinder are at a closest distance.

By analyzing the original configuration of the occurrences of the *cone* and *cylinder* behaviors (Figure 8.13), an observer can verify that there exist some periods when the *cone* is moving and the *cylinder* is during its period of inactivity, and *vice-versa.* Since the spatial location in which one object stops is not part of the path of the other object, it is reasonable to think that increasing the periods when both objects are moving will increase the possibility of collision between the objects.

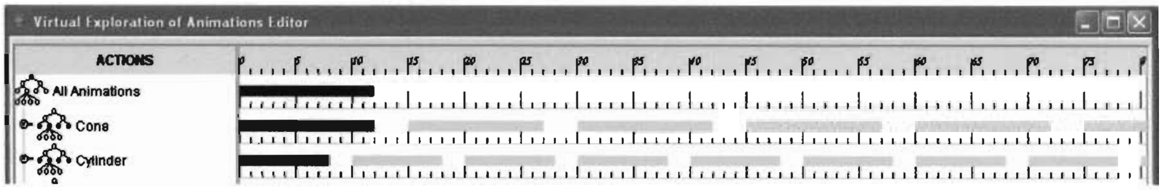
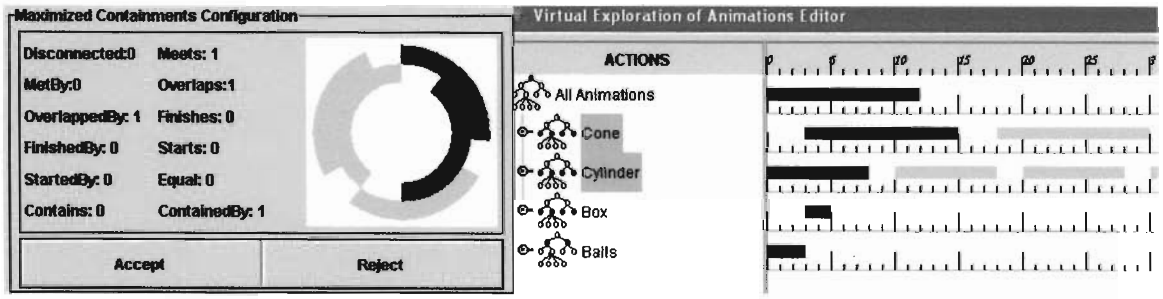


Figure 8.13. Linear representation of occurrences of cone and cylinder's cycles interval in the modeled animation.

In order to maximize the periods when both objects are performing their activities, an observer can select the temporal characteristics of the *cone* and the *cylinder* and perform a `maximizeContainements` operation. The observer makes the decision of which object will be the *reference* interval and which object will be the *target* interval. This decision is based mainly on which interval the observer wants to preserve the original temporal characteristics. This interval must be the *reference* interval.

Based on the temporal characteristics of the *reference* and *target* cycles, the application computes all possible different configurations and selects the best configuration that satisfies the intended temporal relation. The selection of the best configuration is based on the number and kinds of temporal relations that hold between occurrences of the cycles and on the topological distance of these relations as detailed in Chapter 5. Before changing the position of the target interval, the application provides a summary of all temporal relations and a cyclic graphical representation of the cycles' occurrences (Figure 8.14). In this way, an observer has the opportunity to verify the proposed temporal configuration before applying it to the *target* interval. Figure 8.14 shows also the temporal configuration of behavior of the cycles after the observer had accepted the proposed configuration.


Maximized Containments Configuration

Disconnected:0	Meets: 1
MetBy:0	Overlaps:1
OverlappedBy: 1	Finishes: 0
FinishedBy: 0	Starts: 0
StartedBy: 0	Equal: 0
Contains: 0	ContainedBy: 1

Figure 8.14. Summary of temporal relations, linear and cyclic representation of a temporal configuration that maximizes containments between the cone and the cylinder.

An observer can analyze the animation of the modified version the *cone* behavior and verify if his or her assumption about maximizing containments cause the collision of the objects. The outcome of the animation, however, shows that the effect is opposite the one expected by the observer. In the modified version of the animation, the closest distance between the objects occur at instants t=21s and the t=26s. These distances are 4.5 and 1.5 units of distances, respectively (Figure 8.15). In this way, by maximizing containments between the behavior of the *cone* and the *cylinder*, the observer actually minimized the possibility of collisions between these objects.

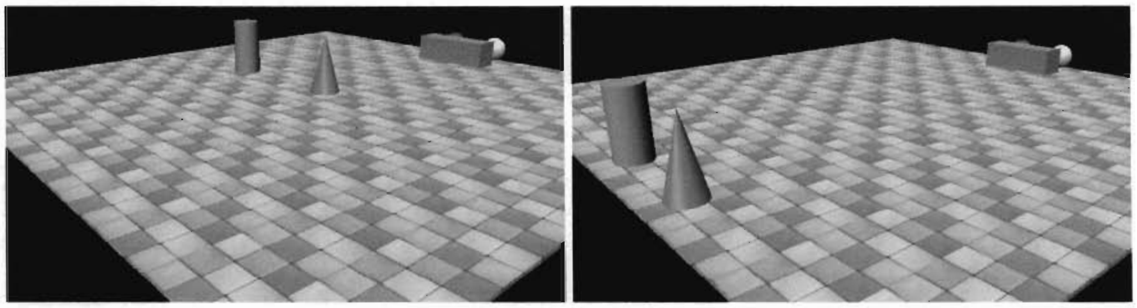


Figure 8.15. Snapshots of the instants in which the cone and the cylinder are at a closest distance in a modified version of the animation that maximizes containments.

Since the operation `maximizeContainements` did not violate any temporal constraint, the application does not have to deal with the existent status of the objects involved in the operation. This fact, however, is not true when the operations are `makeConcurrent` and `makeAlternate`. These operations may change the duration of both the period of activity and the period of inactivity of the *target* cycle. If it occurs, the application changes the status of the object associated with the *target* cycle from existent to non-existent.

Consider, for example, that an observer wants to analyze an animation where the *cylinder* and the *cone* have the same period and start together (i.e., they are concurrent). By selecting the *cylinder* as the *target* interval and performing the `makeConcurrent` operation (Figure 8.16), the temporal configuration of these cycle intervals change to a configuration where a single temporal relation holds between all occurrences of the cycles (i.e., the temporal relation *equal*).

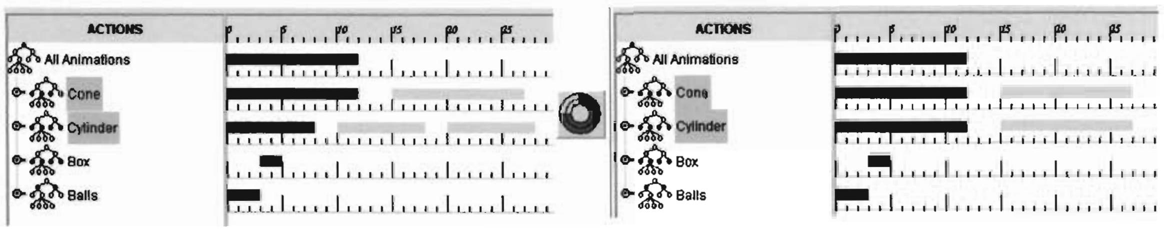


Figure 8.16. The effect in the temporal configuration of the animation by applying a makeConcurrent operation with the cylinder as a target interval.

By analyzing the animation with the concurrent behaviors of the *cylinder* and *cone*, an observer can see the *cylinder* moving in the environment as a *ghost* object or resting in its initial position as a *mirage* object during. Making the behavior of the *cylinder* concurrent with the behavior of the *cone*, however, produces an animation where these objects collide many times. Since the *cylinder* has a non-existent status, these objects never stop performing their activities. Figure 8.17 shows a snapshot of the first instant when the *cylinder* and the *cone* touch each other. The application does not provide any mechanism to modify the semantics of the object. Thus, there is no way to make these objects stop due to a collision detection.

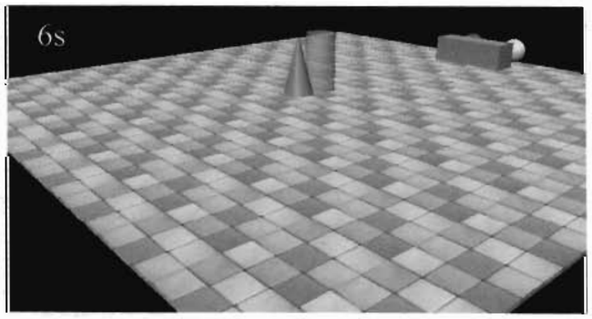
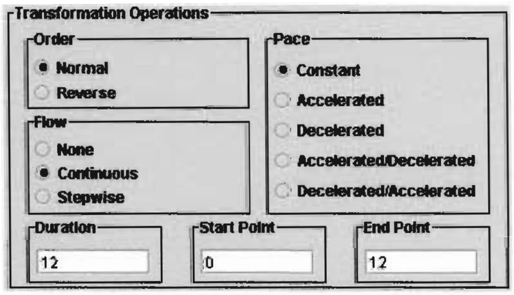


Figure 8.17. Snapshot of the first instant when the cylinder and the cone touch each other in the modified version of the animation where the cycle intervals are concurrent.

8.3.2 Transformation Operations

Transformation operations simulate some kind of transformation of the objects' underlying temporal space. These operations have a single temporal interval as argument. *Transformation operations* modify the way that an observer perceives the evolution of an object's behavior.

The set of *transformation operations* is composed of the operations `Order`, `Flow`, `Pace` and `Duration` (Figure 8.18). The operation `Start Point` and `End Point` appear in interface in the group of *transformation operations* for a convenience in the interface's design. Rigorously, these operations do not produce any transformation of the object's temporal space. These operations are used by an observer for a finer adjustment of the temporal configuration of the animation or to accomplish a configuration that cannot be produced by the set of *combination operations*.


Transformation Operations

Order: [✓ Normal] Reverse

Flow: None [✓ Continuous] Stepwise

Pace: [✓ Constant] Accelerated Decelerated Accelerated/Decelerated Decelerated/Accelerated

Duration: 12

Start Point: 0

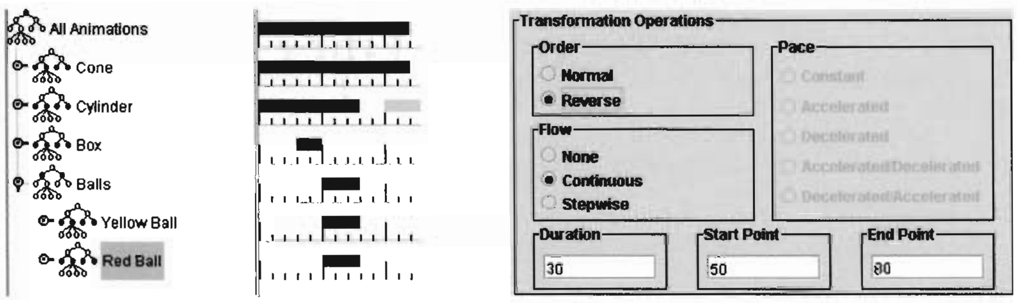
End Point: 12

Figure 8.18. The group of transformation operations.

The operation `Pace` allows an observer to adjust the evolution of an object's behavior to a more realistic presentation. This operation is available only the *Act Interval* level of abstraction. In the example of the objects in the room, all objects have a pace of the kind *constant*. Modifying the pace of an object's act does not alter the existent status of the object.

The operation `Order` allows an observer to change the chronological order of the presentation of object's behavior. The operation `Flow` allows observer to suspend the execution or to present only representative states of an object's behavior. The operation `Duration` allows an observer to change the duration of an object's behavior, slowing down or speeding up the presentation an object's behavior. The operations `Order`, `Flow`, `Duration` always alter the semantics of the associated object from a existent to a non-existent status. In the case of the `Order` operation, for example, the change in the object status reflects the fact that the object is performing its behavior in a reverse chronological order. For the `Flow` operation, the non-existent status of the object reflects the fact that the object was supposed to be performing some kind of activity. In the case of the `Duration` operation the change in the object's status is due to a violation of an intrinsic temporal constraint (i.e., the duration of an object's behavior).

Consider, for example, a more elaborate manipulation of the temporal characteristics of the animation. An observer applies a `makeMirror` operation between the temporal characteristics of the box (the *reference* interval) and the temporal characteristic of the *yellow ball* (the *target* interval). Thus, the movement of the *balls* starts after the completion of the movement of the *box*. Due to a temporal constraint between the *red* and the *yellow* ball, the start point of the *red ball* is also redefined. In addition, the observer changes the order of the movement of the *red ball* (i.e., the *red ball* performs its movement in a reverse chronological order). Figure 8.19 shows the temporal configuration of the animation and a *transformation operation* performed over the *red ball* interval in the interface of the *animation editor*.



Transformation Operations

Order: Normal [✓ Reverse]

Flow: None [✓ Continuous] Stepwise

Pace: Constant, Accelerated, Decelerated, AcceleratedDecelerated, DeceleratedAccelerated

Duration: 30

Start Point: 50

End Point: 80

Figure 8.19. The effect in the temporal configuration of the animation by changing the instant when the balls start to move and inverting the order of the red ball.

By analyzing the temporal configuration of the animation, an observer can verify that the rearrangement of the balls' temporal characteristics does not violate any temporal constraint in the model (i.e., the *red ball* start its movement as the yellow ball starts to move). The fact that the *red ball* is not supposed to be performing a reverse movement, however, forces the application to change the status of the *red ball* from an existent to a non-existent status.

Figure 8.20 shows snapshots of the animation of the *red ball* performing its movement in the reverse chronological order. Between the instants t=3s and t=5s only the *box*, the *cone*, and the *cylinder* are moving in the environment. The *red ball*, however, occupies its start location (i.e., its end location in the modeled animation) with a non-existent status. At the instant t=5s, the *box* has already finished its movement and it is positioned somewhere along the path of the *balls*. Between the instant t=5s and t=8s, the *balls* can be seen performing their behaviors (i.e., the *yellow ball* as an *actor* object and the *red ball* as a *ghost* object). Since the *red ball* has a non-existent status, it is capable to pass through the *box*. This fact is not true for the *yellow ball*. Thus, the *yellow ball* stops its modeled behavior earlier due to a collision with the *box*. After the instant t=8s on, the *red ball* keeps its non-existent status and remains in the environment as a *mirage* object.

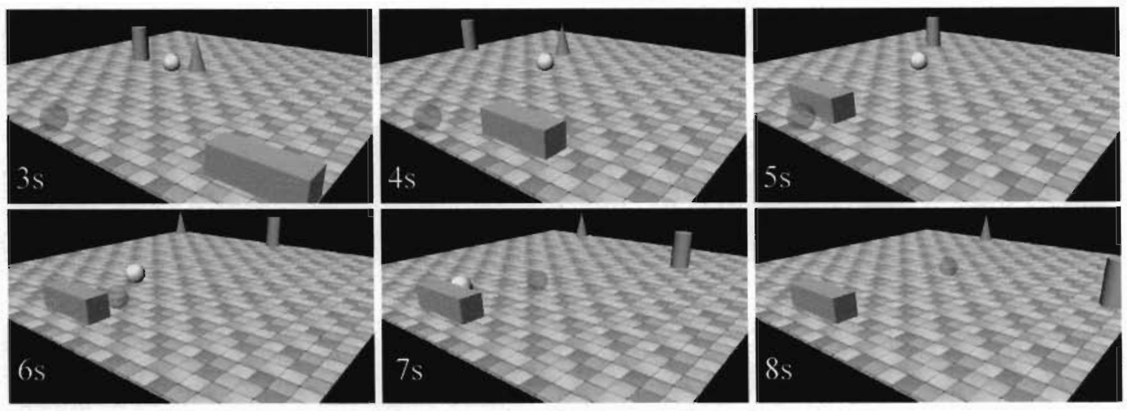


Figure 8.20. Snapshots of an animation where only the red ball performs its movement in a reverse chronological order.

8.3.3 Combination Operations

Combinations operations are similar to operations over mathematical sets. These operations are `union`, `difference`, and `intersection` (Figure 8.21).

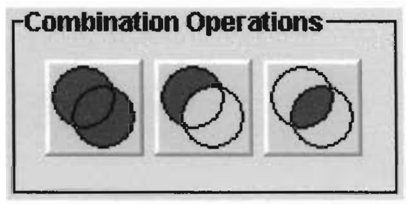


Figure 8.21. The group of composition operations between cycle intervals.

Combinations operations allow an observer to manipulate the structure of the tree and the content of the animation. In this implementation, *Combinations operations* are implemented only for objects of the sort *Animation Interval* (i.e., these operations are applied only at the coarsest level of granularity of objects' behavior).

The `union` of two *Animation Intervals* produces a new node in the tree that has both arguments as children. Thus, this operation does not change the outcome of the animation and it is used only as a grouping mechanism. The `difference` and `intersection` have the effect of performing a `Flow` operation with the argument *none* in pieces of an object's behavior. Thus, the application changes the existent status of the objects during periods in which its behavior is disabled. Consider, for example, the `difference` between the animation of the *box* and the *cone*. The *cone* performs its modeled behavior until the *box* starts to move and then the *cone* stops its movement during the period of activity of the *box*. The *cone* becomes a *mirage* object during this period. This change reflects the fact that the *cone* is not supposed to be at its actual position in the environment. After the completion of the behavior of the *box*, the *cone* recovers its existent status, "jumps" to its modeled position, and continues to move as an *actor* object (Figure 8.22).

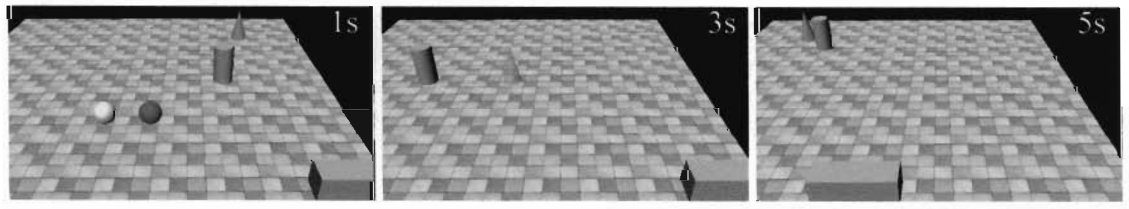


Figure 8.22. Snapshots of an animation with the difference between the behaviors of the box and the cone. The cylinder does not move during the period when the box is moving.

8.4 Hypothesis Evaluation

The use of the VCR metaphor to direct the way in which a sequence of images is presented to an observer has been the dominant paradigm for the analysis of time-varying information in the GIS field. Clear advantages of the VCR metaphor are that the number of operations is small and the functionality of each operation is widely known. A major drawback of such a mechanism, however, is that the information is treated as whole, which makes it impossible to control individual pieces of the information. Thus, the number of different views that can be created by a user is limited.

Extending the set of VCR-like operations to each object's behavior in the environment enhances the user's ability to produce more views of the information. In a scenario where all objects in the application have an associated VCR control, the number of different views of the dynamic environment is large. We call this model *Individual VCR*. Despite the large number of views that can be produced using the *Individual VCR* approach, this thesis postulates the hypothesis that the model of *virtual exploration* of *animations* allows an observer to produce views that cannot be produced by any combination of operations of an animation model in which each dynamic object has an associated VCR control. The next sections highlight limitations of the *Individual VCR model* when compared with views that can be produced by a user with the model of *virtual exploration* of *animations*.

8.4.1 What Can Be Done with Operations of the Individual VCR Model

In order to illustrate the number of different versions of animation that can be produced with operations of the *individual VCR* model, we use the example of objects moving in a room introduced earlier in this chapter. In the example of the room there are five objects with associated behaviors.

In the *individual VCR* model, each object's behavior in the environment has an extended VCR control attached to it and each control has exactly one of the operations selected. The number of controls in a VCR-like interface varies according to the application. There is no standard specifying the number and kind of controls. These characteristics depend on the requirements of the application domain. In our hypothetical model of *individual VCR* we propose an interface with a large number of buttons (Figure 8.23). It is very unusual for an application to require such a number of operations. We use it here only for comparison purposes.

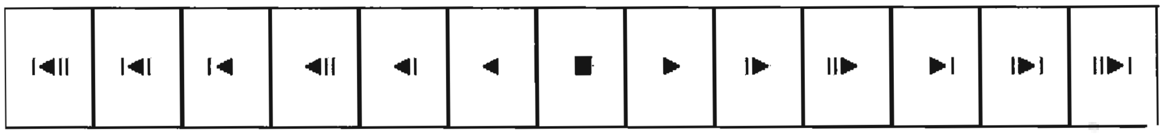


Figure 8.23. Operations of the extended VCR control.

The extended set of VCR controls has 13 buttons The functionality of each button is in the order: *frame by frame fast backward*, *frame by frame slow backward*, *frame by frame backward*, *fast backward*, *slow backward*, *backward*, *stop*, *forward* or *play*, *slow forward*, *fast forward*, *frame by frame forward*, *frame by frame slow forward*, and *frame by frame fast forward*. Operations with the modifier *frame by frame* show only key frames of objects' behavior. Thus, if the object is moving between two locations and only the initial and final position of the object is stored in the model (i.e., the object's key states), the animation shows the object in its initial position during the duration of the movement and at its final position at the end. These operations have the effect of showing the object "jumping" from one location to another without passing through intermediate locations. Operations with the modifier *fast* and *slow* show objects performing their behavior twice as fast or slow.

Consider, for example, that a user of the *individual VCR* model wants to produce an animation in which different operations of the extended VCR control are selected for each object's behavior (Figure 8.24). The analysis of the progression of this version of the animation shows that the *cone* is moving backward, the *cylinder* is moving forward, the *box* is not moving, the *yellow ball* is moving forward twice as fast and the *red ball* is seen only at its initial or final position (i.e., the animation does not show intermediate positions of the red ball). This animation represents only one possible configuration of the animation that can be accomplished by a user with the individual VCR model. Selecting a different operation for any object's behavior produces a new version of the animation.

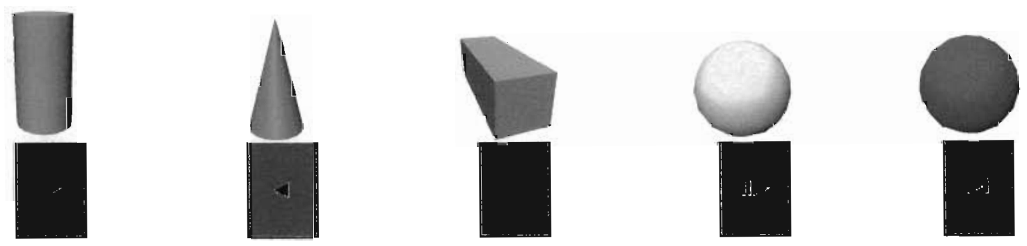


Figure 8.24. A view of the animation produced by different operations of the VCR control for each object's behavior.

Animations produced by operations of the *individual VCR* model can be simulated by the combination of three operations of the model for *virtual exploration of animations* (i.e., `Duration`, `Order`, and `Flow`). These operations are part of the group of *transformation* operations. This group of operations affects the way that an observer perceives the evolution of objects' states with the passage of time and can be used over abstractions of the model that represent the entire behavior of an object.

Transformation operations take two arguments: the first argument is an object's behavior and the second argument is a value for the temporal characteristic of the object behavior that the operation modifies. The number and kinds of values of the second argument varies according to the *transformation* operation. The operation `Order`, for example, has two kinds of arguments: *reverse* and *normal*. The *reverse* argument causes the behavior of the object to be perceived in the reverse chronological order, while the argument *normal* preserves the modeled evolution of the object's behavior. The operation `Flow` has three kinds of arguments: *none*, *continuous*, and *stepwise*. The argument *none* indicates that the object is not performing its associate behavior. When the `Flow` operation has the argument *none*, the values of the arguments of other operations are irrelevant. This operation is equivalent to the operation *stop* of a VCR control. The arguments *continuous* or *stepwise* indicates that the object is performing its associated behavior but in a different way. The argument *continuous* causes the illusion of the object moving continuously between two locations, while the argument *stepwise* shows only key states of an object movement. The operation `Duration` does not have a fixed range of values for the second argument of the operation (i.e., the duration of an object's behavior). The user can specify any value for this argument. It makes the number of views produced by this operation infinite. In order to be compatible with the operations available in a VCR control we limit the range of the second argument to the following values: (*1x*, *0.5x*, and *2x*). The first value keeps the modeled duration of the object's behavior, the second argument reduces the modeled duration by half, and the third argument increases the modeled duration by a factor of 2.

In order to produce the same versions of the animation created by operations of the *individual VCR* model it is necessary to select appropriate values for the arguments of the operations `Order`, `Flow` and `Duration`. Figure 8.25 shows the previous version of the animation produced with the *individual VCR* model using a combination of *transformation* operations of the model for *virtual exploration of animations*.

Order(*normal*) Flow(continuous) Duration(*1x*)
Order(*reverse*) Flow(continuous) Duration(*1x*)
Order(*any*) Flow(none) Duration(*any*)
Order(*normal*) Flow(continuous) Duration(*2x*)
Order(*normal*) Flow(stepwise) Duration(*1x*)

Figure 8.25. A view of the animation produced by the combination of the operations order, flow and duration.

For each operation of the extended VCR control there is a unique combination of the operations `Order`, `Flow` and `Duration` that produces the same result. Table 8.1 shows the values of the arguments of *transformation* operations for every operation of the *individual VCR* model.

Operations of Individual VCR Model	Operations of the Virtual Exploration of Animation Model
frame by frame fast backward	Order(*reverse*) Flow(*stepwise*) Duration(*0.5x*)
frame by frame slow backward	Order(*reverse*) Flow(stepwise) Duration(*2x*)
frame by frame backward	Order(*reverse*) Flow(stepwise) Duration(*1x*)
fast backward	Order(*reverse*) Flow(*continuous*) Duration(*0.5x*)
slow backward	Order(*reverse*) Flow(*continuous*) Duration(*2x*)
backward	Order(*reverse*) Flow(*continuous*) Duration(*1x*)
stop	Order(*any*) Flow(*none*) Duration(*any*)
forward or *play*	Order(*normal*) Flow(*continuous*) Duration(*1x*)
slow forward	Order(*normal*) Flow(*continuous*) Duration(*2x*)
fast forward	Order(*normal*) Flow(*continuous*) Duration(*2x*)
frame by frame forward	Order(*normal*) Flow(stepwise) Duration(*1x*)
frame by frame slow forward	Order(*normal*) Flow(stepwise) Duration(*2x*)
frame by frame fast forward	Order(*normal*) Flow(stepwise) Duration(*0.5x*)

Table 8.1. Operations of the extended VCR control and the equivalent set of operation in the virtual exploration of animation model.

The set of operations of the model for *virtual exploration of animation* is richer than the set of operations composed of `Order`, `Flow`, and `Duration`. Other operations of the model produce different versions of the animation that cannot be accomplished by any operation of the *individual VCR* model. The next section discusses such operations.

8.4.2 What Cannot Be Done with Operations of the Individual VCR Model

The number of different versions of the animation produced by *transformation* operations `Order`, `Flow`, and `Duration` offer the same capabilities (measured in terms of views) as the *Individual VCR* model. What is more, *in the model for virtual exploration of animations* these capabilities are provided by a more compact interface. The group of *transformation* operation, however, has an additional operation (i.e., `Pace`) that changes the way that an observer perceives the evolution of an object's behavior. Thus, this operation also produces different views of the animation. The operation `Pace` has five kinds of arguments. This operation allows the representation of object's behavior at variable speeds (e.g., accelerating or decelerating). This effect cannot be accomplished by any operation of the *individual VCR* model. The operation `Pace` is used to give a more realistic presentation of an object's behavior. For a large number of applications, however, the effect of the `Pace` operation does not give any additional insight into the analysis of the information. In this way, this operation is irrelevant for most applications and does not constitute a significant advantage of the *virtual exploration of animations* model over the *individual VCR* model. The main difference between the *virtual exploration of animations* and *individual VCR* models is defined by two groups of operations: *composition* and *combination* operations.

C*omposition operations* give a user the opportunity of changing the temporal arrangement of the elements of the animation in order to gain insights and discover relationships among geographic phenomena. These operations take two behavior objects as arguments. Depending on the order of the arguments, the operation produces a different view of the animation.

The group of *composition* operations is composed of two groups of operations (i.e., *linear* and *cyclic* compositions). *Linear* compositions are operations that change the temporal characteristics of object's behavior by imposing a temporal relationship between their arguments. *Cyclic* compositions are operations that change the temporal characteristics of cyclic behaviors taking into account the temporal relation that must hold between all occurrences of the cycles used as arguments.

The group of *combination* operations is composed of three operations: `union`, `difference`, and `intersection`. The `union` operation allows an observer to rearrange the structure of the tree in a more suitable structure based on the task at hand. This operation is useful in a real-world scenario where the number of animated objects is large. The `difference` and `intersection` of two animations change the outcome of the animation by limiting the periods of time when the objects are seen performing their activities. The `difference` of two animations, for example, defines an animation where the objects associated with the second argument have an inactive state during the periods in which the objects associated with the first arguments are moving. This operation reduces the number of active objects in the environment that are not of interest to an observer, thus, facilitating the analysis of phenomena of interest and reducing the computational cost of rendering the animation. The `intersection` operation generates an animation where the objects associated with its arguments are seen performing their activities only when both objects have an active state. This operation allows an observer to isolate pieces of animations where all objects of interest are moving simultaneously.

The unique views created by *composition* and *combination* operations supports the hypothesis of this thesis. These operations create views of the animation that cannot be accomplished by operations of the *individual VCR* model. The simple accounting of the number of views of such operations, however, does not represent the importance of these operations in the cognitive process of exploring and analyzing dynamic environments. An observer cannot simulate the functionality behind this group of operations simply by manipulating the start points of the objects' behaviors. The group of *cycle compositions* is the most significant example of the importance and complexity behind the operations of the model. For example, an operation that maximizes the incidence of a certain temporal relation takes into account temporal relations among the occurrences of the cycle at different levels of conceptual neighborhood and the topological distances of these relations. Moreover, the mere calculation of the number of views does not consider additional qualitative advantages of the model for *virtual exploration of animations*. Operations of this model are performed across different levels of granularity of object's behaviors. In the example discussed in this thesis, however, we consider for the sake of comparison only the number of operations performed over the behavior of individual objects. The number of different views produced by manipulating individual pieces of an objects' behavior or the behavior of groups of objects, increases the number of views produced by the model. Additionally, we assume that all views are equally important and have the same weight in the comparison of the models. This assumption does not consider, for example, the possibility of the *Individual VCR* model to produce a view that is not coherent with the constraints holding in the application domain.

8.5 Summary

This chapter introduced a graphical user interface for the model of *virtual exploration of animations*. The user interface is composed of an *animation editor* and an *animation browser*. The *animation editor* implements all operations that allow a user to manipulate the content of modeled animations. The *animation browser* allows an observer to analyze the modified version of the animation in a non-immersive virtual reality environment.

An illustrative example was introduced and used to discuss the *animation editor* and *animation browser* user interface. The example was also used to compare views that can be produced by the models for *virtual exploration of animations* and *individual VCR* and to test the hypothesis of this thesis.

The next chapter summarizes major contributions and highlights the major findings of this thesis. Future work is also discussed.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

This thesis focuses on the presentation and analysis of geographic spaces in VR settings. The research presents a data model that supports the manipulation, analysis, and presentation of dynamic geographic objects in VR environments, giving attention to the representation of interactions between the user and the data set in the spatial and temporal domains.

9.1 Summary of Thesis

The main motivation of this thesis is the lack of a framework that properly supports the exploration of geographic information in a multi-dimensional and multi-sensorial environment (i.e., temporal virtual reality geographic information systems). More specifically, this thesis is concerned with a data model and a user interface that allows non-expert users to analyze, explore, and produce different views of the data in order to gain insights and discover relationships among geographic phenomena.

Five research issues drove the search for a model that supports the exploratory analysis of dynamic objects in Temporal VRGIS applications. The first issue is that multiple levels of abstractions are needed to represent objects' behavior. The second issue is that complex temporal structures are needed to capture the richness of geographic phenomena. The third issue is that known dependencies among geographic phenomena must be represented to incorporate knowledge from the application domain. The fourth issue is that the semantics of geographic objects in a multi-sensorial environment need to be represented. The fifth is that new operations are needed to support the manipulation of objects' behavior. The conceptualization of the model for *virtual exploration of animations* addresses these issues.

Entities of the action and temporal parts of the model considered two major requirements of a GIS application. These entities are both abstract enough to allow a non-expert user to perform qualitative spatio-temporal reasoning and robust enough to support a wide variety of geographic applications. Moreover, the organizations of these entities are similar. This characteristic gives a user a coherent representation across different domains and allows a quick assimilation of the model's structure.

In the model for *virtual exploration of animations*, the manipulation of the dynamic environment is accomplished through a set of operations performed over abstractions that represent temporal characteristics of actions. An important feature of the model is that the temporal information is treated as first-class entities and not as a mere attribute of action's representations. Therefore, entities of the temporal model have their own built-in functionality and are able to represent complex temporal structures.

In an environment designed for the manipulation of the temporal characteristics of actions, the knowledge of relationships among objects' behaviors plays a significant role in the model. This information comes from the knowledge base of the application domain and is represented in the model through constraints among entities of the temporal model. These constraints are used to keep the modified version of the information coherent with the information available in the application domain. The complexity of these constraints increases with the complexity of the temporal structures being represented. Such constraints vary from simply relating the end points of two intervals to a complex mechanism that takes into account all relations between sequences of intervals.

The fact that the exploration of the information takes place in a virtual reality environment imposes new requirements on the data model that supports the presentation of the information. In such an environment, the observer becomes an increasing part of the data. This thesis introduces a new classification of objects in a VR environment and describes the associated semantics of each element in the taxonomy. This thesis also introduces a mechanism to represent such semantics over time and the rationale to alter object's semantics characteristics under the modification by a user of the object's temporal characteristics. These semantics are used to direct the way an object interacts with an observer and with other objects in the environment.

This thesis also introduces a prototype implementation for the model of *virtual exploration of animations*. The prototype is composed of an *animation editor* and an *animation browser*. The *animation editor* implements all functionalities of the model and presents, through a graphical user interface, all operations that allow an observer to manipulate the content of modeled animations. The *animation browser* allows the exploration of the information in a virtual reality environment.

9.2 Results and Major Findings

This thesis introduced a data model and a user interface that give users a cognitively plausible framework to analyze, explore, and produce different views of dynamic environments. The major advantages of the model of *virtual exploration of animations* over existing data models are:

- The model supports the representation of objects' behavior at different levels of granularity. This characteristic allows a non-expert user to control the pieces of the behavior a single object, the entire behavior an object, the behavior of groups of object, or all objects in the environment.

- The model supports the representation of more elaborate temporal structures. These structures are able to represent complex geographic phenomena and support temporal reasoning over cyclic behaviors.

- The model incorporates temporal constraints among entities at different levels of abstractions. These constraints are used to preserve some known relationships among objects' behavior during the manipulation of the animation by the user. A salient contribution of this thesis is a novel mechanism to represent temporal constraints between cyclic phenomena.

- The model supports the representation of the evolution of the VR objects' semantics over time. The semantics associated with VR objects provide valuable information in the process of the virtual exploration of an environment and play a significant role in interactions with observers and the data.

- The model has a set of operations that allow a user to create new views of the environment. It was demonstrated that the number of different views that can be produced by a user is greater than in a model where each object's behavior has an associated VCR control.

9.3 Future Work

This thesis addresses specific research questions concerning a framework for the presentation and analysis of multi-dimensional geographic information. Although the model of *virtual exploration of animations* represents a step forward for a truly temporal VRGIS framework, there are some research questions that need to be addressed in future work.

9.3.1 Anticipation of Mutual Interference

The user interface to manipulate the content of animations presents the organization of action objects in a tree-like structure. The temporal characteristics of these objects are depicted as temporal intervals positioned along the animation timeline. In such a representation, an observer does not have any spatial information about the objects' behaviors. Thus, the fact that two objects interact in the temporal domain (e.g., the objects move at the same time) does not mean that the objects interact in the spatial domain (e.g., the objects collide). In order to verify spatial interactions, the observer need to explore the environment.

The model of virtual exploration deals with pre-orchestrated object's behaviors (i.e., the behavior of the object is known ahead of time). Thus, the model has all information needed to allow an application to anticipate any kind of spatial interaction.

Allowing the application to process spatial interactions before the presentation of the animation has two main advantages. First, an observer knowing that the manipulation of the animation causes the collision of two objects can change the temporal configuration to avoid the collision or keep the temporal configuration and analyze the way in which these objects interact. In the later scenario, an observer also knows when the interaction occurs. Thus, he or she can focus only on small segments of the animation. Second, developers of the model can implement a more elaborate algorithm to treat collisions between complex geometries and to bring the collision detection mechanism to the animation editor. This approach alleviates the computational demand of the animation browser, allowing the presentation of complex four-dimensional environments.

9.3.2 Incorporating Knowledge

The knowledge from the application domain is represented in the model through temporal constraints among the entities of the temporal part of the framework. This knowledge is sometimes incomplete or wrong. In our model, however, an observer cannot change this kind of information. An observer exploring the dynamic environment can gain insights and discover new relationships among geographic phenomena. In this way, it is necessary to incorporate in the model a mechanism that allows an observer to add or remove information from the knowledge base of the application domain. The insights and understandings achieved by observers exploring the modified dynamic environment must be used to re-feed the knowledge base of the application domain, creating a virtuous cycle.

9.3.3 Support for Causalities

The model for *virtual exploration of animations* has no explicit notion of causality. Causality can be treated only implicitly, as a temporal relationship between objects behavior. In order to capture the complexity of geographic phenomena and their relationships, the set of temporal constraints need to be extended with a formal representation of causal constraints and a richer set of causal predicates (e.g., cause, prevent, enable, help, and hinder). This kind of information allows the extension of the rationale used to update the semantics and temporal characteristics of object's behavior during the manipulation of the animation in a way that more complex feedbacks can be presented to an observer. Thus, the support of causalities may increase an observer capability to get insights from the animated environment.

Consider, for example, the behavior of two objects (o1 and o2) related through the causal predicate *help*, that is, *help*(o1,o2). One possible rationale to change the behavior of the object o2 when the user disables the behavior of o1 is to slow down the movement of the second object. Thus, an observer can perceive that something that was "helping" the movement of the object o2 is not in place anymore.

9.3.4 Cycles Behaviors at Different Levels of Granularity

Cyclic behaviors are allowed only at the level of granularity representing the entire behavior of an object (i.e., an object Course of Actions). It is important to extend the model to support the representation of cycles at both a finer and a coarser level of granularity. At a finer level of granularity, it is important to represent pieces of an object's behavior with a cyclic pattern of repetition. This feature avoids the duplication of certain pieces of information in the model, thus minimizing sources of errors. At a coarse level of granularity, cycles can be defined qualitatively in terms of other cyclic behaviors (e.g., a phenomena that occurs only between occurrences of other cyclic behaviors, or only when occurrences of others cycles overlaps in time).

9.3.5 Uncertainties

The temporal characteristics of objects behavior are defined with attributes that can assume only a single value. Thus, it is impossible to deal with uncertainties in the model. Possible approaches to incorporate uncertainties in the model are to allow the temporal attributes to deal with an interval of values or a triple with a *minimum*, a *maximum*, and a *likely* value. Either approach has a tremendous impact on the functionality of the model an on the interface used to treat these representations.

Incorporating uncertainties in the model also has an impact on the temporal constraint mechanism. This mechanism needs to be extended to support imprecise and complex constraints (e.g., a certain behavior meets or overlaps another behavior).

REFERENCES

Allen, J. F. (1983) Maintaining Knowledge About Temporal Intervals. *Communications of the ACM.* 26(11): 822-843.

Allen, J. F. and G. Ferguson (1997) Actions and Events in Interval Temporal Logic. in: O. Stock (Ed.) *Spatial Temporal Reasoning*, Kluwer Academic, 205-245.

Allen, J. F. and P. J. Hayes (1990) Moments and Points in an Interval-Based Temporal Logic. *Computational Intelligence.* 5: 225-238.

Andrienko, N., G. Andrienko and P. Gatalsky (2000) Supporting Visual Exploration of Object Movement. in: *Advanced Visual Interfaces*, Palermo, Italy, 217-220.

ANSI (1985) Ansi (American National Standards Institute), American National Standard for Information Processing Systems - Computer Graphics - Graphical Kernel System (Gks) Functional Description, Ansi X3.124-1985. New York, ANSI.

ANSI (1988) Ansi (American National Standards Institute), American National Standard for Information Processing Systems - Programmer'S Hierarchical Interactive Graphics System (Phigs) Functional Description, ANSI X3.144-1988. New York, ANSI.

Balbiani, P., A. Osmani, J.-F. Condotta and L. F. d. Cerro (1988) A Model for Reasoning About Generalized Intervals. in: *Sixth International Conference on Principles of Knowledge Representation and Reasoning - KR'98*, Trento, Italy, 124-130.

Blok, C., B. Kobben, T. Cheng and A. A. Kuterama (1999) Visualization of Relationships between Spatial Patterns in Time by Cartographic Animation. *Cartography and Geographical Information Science*. 26(2): 139-151.

Brodlie, K., J. Dykes, M. Gillings, M. Haklay, R. Kitchin and M.-J. Kraak (2002) Geography in Vr: Context. in: P. Fisher and D. Unwin (Eds.) *Virtual Reality in Geography*, New York, Taylor & Francis, 7-16.

Brodlie, K. W., L. A. Carpenter and R. A. Earnshaw (1992) *Scientific Visualization, Techniques and Applications*, Berlin, Springer-Verlag.

Buckley, A., M. Gahegan and K. Clark (2000) *Geographic Visualization as an Emerging Research Theme in Giscience*. Technical Report UCGIS.

Campos, J., M. Egenhofer and K. Hornsby (2003a) Animation Model to Support Exploratory Analysis of Dynamic Environment. in: *SCSC'03 Summer Computer Simulation Conference*, Montral, Canada, 761-766.

Campos, J., K. Hornsby and M. Egenhofer (2002) A Temporal Model of Virtual Reality Objects and Their Semantics. in: *8th International Conference on Distributed Multimedia System (DMS'02)-Workshop on Visual Computing*, San Francisco, CA 581-588.

Campos, J., K. Hornsby and M. Egenhofer (2003b) A Model for Exploring Virtual Reality Environments. *Journal of Visual Languages and Computing*. 14(5): 471-494.

Cartwright, W. (1999) Extending the Map Methaphor Using Web Delivered Multimedia. *International Journal of Geographical Information Science*. 13(4): 335-353.

Chandru, V., N. Mahesh, M. Manivannan and S. Manohar (2000) Volume Sculpting and Keyframe Animation System. in: *Computer Animation 2000*, Philadelphia, PA, 134-139.

Chen, S. E. (1995) Quicktime Vr: an Image-Based Approach to Virtual Environment Navigation. in: *Siggraph*, Los Angeles, CA, 29-38.

Chitaro, L. and I. Scagnetto (2001) Is Semitransparency Useful for Navigating Virtual Environments. in: *Virtual Reality Software and Technology (VRST)*, Banff, Canada, 159-166.

Chomicki, J., Y. Liu and P. Revesz (1999) Animating Spatiotemporal Constraint Databases. in: M. H. Bohlen, C. S. Jensen and M. O. Scholl (Eds.) *STDBM'99*, Edinburgh, Scotland, 224-240.

Cohen, M. F. (1992) Interactive Spacetime Control for Animation. *Computer Graphics*. 26(2): 293-302.

Cosatto, E. and H. P. Graf (2000) Photo-Realistic Talking-Heads from Image Samples. *Transactions on Multimedia*. 2(3): 152-163.

Cuckierman, D. and J. Delgrande (2000) A Formalization of Structural Temporal Objects and Repetitions. in: *TIME'2000*, Cape Bleton, Canada, 13-20.

Dam, A. v., A. Forsberg, D. Laidlaw, J. LaViola and R. Simpsom (2000) Immersive Vr for Scientific Visualization: A Progress Report. *IEEE Computer Graphics and Applications*. 20(6): 26-52.

DiBiase, D., A. MacEachren, J. Krygier and C. Reeves (1992) Animation and the Role of Map Design in Scientific Visualization. *Cartography and Geographical Information Systems*. 19(4): 201-214,265-266.

Dollner, J. and K. Hinrichs (1997) Object-Oriented 3d Modeling, Animation and Interaction. *Journal of Visualization and Computer Animation*. 8(1): 33-64.

Dykes, J., K. Moore and J. Wood (1999) Virtual Environment for Student Fieldwork Using Networked Components. *International Journal of Geographical Information Science*. 13(4): 397-416.

Egenhofer, M. and K. Hornsby (1998) Spatio-Temporal Reasoning About Identity and Visibility. Integrating Spatial and Temporal Databases. Schloss Dagstuhl, Germany, November 1998: (Presentation).

Elliott, C., G. Schechter, R. Yeung and S. Abi-Ezzi (1994) Tbag: A High Level Framework for Interactive, Animated 3d Graphics Applications. in: *SIGGRAPH'94*, Orlando, FL, 421-434.

Erwig, M., R. H. Güting, M. Schneider and M. Vazirgiannis (1998) Abstract and Discrete Modeling of Spatio-Temporal Data Types. in: *6th International Symposium on Advances in Geographic Information Systems*, Washington D.C., 131-136.

Eynde, F. V. (1987) Iteration, Habituality, and Verb Form Semantics. in: *Third Conference of European Chapter of the Association for Computational Linguistics*, Copenhagen, Denmark, 270-277.

Fairbain, D., G. Andrienko, N. Andrienko, G. Buziek and J. Dykes (2001) Representations and Its Relationships with Cartographic Visualization. *Cartography and Geographical Information Science*. 28(1): 13-28.

Faust, N. L. (1995) The Virtual Reality of Gis. *Environment and Planning B: Planning and Design*. 22(3): 257-268.

Fiume, E., D. Tsichritzis and L. Dami (1987) A Temporal Scripting Language for Object-Oriented Animation. in: *Eurographics 1987*, Holland, Elsevier Science Publishers, 283-294.

Foley, J., A. V. Dam, S. Feiner and J. Hughes (1997) *Computer Graphics Principles and Practice*. Reading, MA, Addison-Wesley.

Forlizzi, L., R. H. Güting, E. Nardelli and M. Schneider (2000) A Data Model and Data Structures for Moving Objects Databases. in: *International Conference on Management of Data and Symposium on Principles of Database Systems*, Dallas, TX, 319-330.

Frank, A. U. (1998) Different Types of "Times" in Gis. in: M. J. Egenhofer and R. G. Golledge (Eds.) *Spatial and Temporal Reasoning in Geographic Information Systems*, New York, Oxford University Press, 40-62.

Freksa, C. (1992) Temporal Reasoning Based on Semi-Intervals. *Artificial Intelligence*. 54: 199-227.

Fuhrmann, S. and A. MacEachren (1999) Navigating Desktop Geovirtual Environments. in: *IEEE Information Visualization 99*, San Francisco, CA, 11-14.

Funge, J. (2000) Cognitive Modeling for Games and Animations. *Communications of the ACM.* 43(7): 41-48.

Gahegan, M. (2001) Geovisualization. in: *CSISS Workshop*, Ann Arbor, MI.

Gall, D. L. (1991) A Video Compress Standard for Multimedia Applications. *Communications of the ACM.* 34(4): 46-58.

Gonzalez, R. and R. Woods (1992) *Digital Image Processing*. Reading, MA, Addison-Wesley.

Green, M. and S. Halliday (1996) A Geometric Modeling and Animation System for Virtual Reality. *Communications of the ACM.* 39(5): 46-53.

Haklay, M. (2002) Virtual Reality and Geographical Information Systems: Analysis and Trends. in: P. Fisher and D. Unwin (Eds.) *Virtual Reality and Geography*, London, Taylor & Francis, 47-57.

Hardisty, F., A. MacEachren, M. Gahegan, M. Takatsuka and M. Wheeler (2001) Cartographic Animation in Three Dimensions: Experimenting with the Scene Graph. in: *20th International Cartographic Conference*, Beijing, China.

Herring, J. R. (1991) The Mathematical Modeling of Spatial and Non-Spatial Information in Geographic Information Systems. in: D. Mark and A. U. Frank (Eds.) *Cognitive and Linguistic Aspects of Geographic Space*, Dordrecht, Kluwer, 313-350.

Hornsby, K. and M. Egenhofer (2000) Identity-Based Change: A Foundation for Spatio-Temporal Knowledge Representation. *International Journal of Geographical Information Science*. 14(3): 207-224.

Hornsby, K., M. Egenhofer and P. Hayes (1999) Modeling Ciclic Change. in: P. Chen, D. Embley, J. Kouloumdjian, S. Liddle and J. Roddick (Eds.) *Advances in Conceptual Modeling*, Paris, France, Springer-Verlag, 98-109.

Horton, I. (1999) *Beginning Java 2*. Birmingham, UK, Wrox Press.

Jacobson, R. (1991) Virtual Worlds, Inside and Out. in: D. M. Mark and A. U. Frank (Eds.) *Cognitive and Linguistic Aspects of Geographic Space*, Netherlands, Kluwer Academics, 507-514.

Java-3D (2004) Java 3D Application Programming Interface (API). URL:http://java.sun.com/products/java-media/3D/.

Jensen, C. S., J. Clifford, S. K. Gadia, A. Segev and R. T. Snodgrass (1992) A Glossary of Temporal Database Concepts. in: *International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, 25-29.

Kalawsky, R. S. (1993) *The Science of Virtual Reality and Virtual Environments: A Technical, Scientific, and Engineering Reference on Virtual Environments*. Reading, MA, Addison-Wesley.

Koller, D., P. Lindstrom, W. Ribarsky, L. F. Hodges, N. Faust and G. Turner (1995) Virtual Gis: A Real-Time 3d Geographic Information System. in: *IEEE Visualization'95*, Worcester, MA, 94-100.

Koved, L. and W. Wooten (1993) Groop: An Object-Oriented Toolkit for Animated 3d Graphics. in: *Conference on Object-Oriented Programming Systems, Languages, and Applications-OOPSLA'93*, Washington, DC, 309-325.

Kraak, M.-J. (2002) Visual Exploration of Virtual Environments. in: P. Fisher and D. Unwin (Eds.) *Virtual Reality in Geography*, New York, Taylor & Francis, 58-67.

Kraak, M.-J., G. Smets and P. Sidjanin (1999) Virtual Reality, the New 3-D Interface for Geographical Information Systems. in: A. S. Câmara and J. Raper (Eds.) *Spatial Multimedia and Virtual Reality*, London, Taylor & Francis, 131-136.

Krygier, J. (1994) Sound and Geographic Visualization. in: A. MacEachren and D. R. F. Taylor (Eds.) *Visualization in Modern Cartography*, Oxford, Pergamon Press, 149-166.

Ladkin, P. (1986) Time Representation: A Taxonomy of Interval Relations. in: *Fifth National Conference on Artificial Intelligence (AAAI'86)*, Philadelphia, PA, 360-366.

Leban, B., D. McDonald and D. Forster (1986) A Representation for Collections of Temporal Intervals. in: *Fifth National Conference on Artificial Intelligence (AAAI'86)*, Philadelphia, PA, 367-371.

Lee, G. S. (1998a) A Classification of File Formats for Animation. in: *Western Symposium on Computer Graphics*, Whistler, Canada.

Lee, G. S. (1998b) A General Specification for Scene Animation. in: *International Symposium on Computer Graphics, Image Processing, and Vision - Sibgrapi'98*, Rio de Janeiro, Brazil, IEEE, 1-8.

Lieberman, E. (1991) Integration Gis, Simulation, and Animation. in: B. L. Nelson, W. D. Kelton and G. M. Clark (Eds.) *Winter Simulation Conference*, Phoenix, AZ, 771-775.

Lin, C.-R., R. B. Loftin and J. H. Roice Nelson (2000) Interaction with Geoscience Data in an Immersive Environment. in: *Virtual Reality 2000 Conference*, New Brunswick, NJ, IEEE Computer Society, 55-62.

Little, T. D. C. and A. Ghafoor (1993) Interval-Based Conceptual Models for Time-Dependent Multimedia Data. *IEEE Transactions on Knowledge and Data Engineering (Special Issue on Multimedia Information System)*. 5(4): 551-563.

Luttermann, H. and M. Grauer (1999) Vrml History: Storing and Browsing Temporal 3d-Worlds. in: *ACM Fourth Symposium on VRML*, Paderborn, Germany, 150-163.

MacEachren, A. and M.-J. Kraak (2001) Research Challenges in Geovisualization. *Cartography and Geographical Information Science*. 28(1): 3-12.

Macedonia, M. (2002) Games Soldiers Play. IEEE Spectrum: 32-37.

Manoharan, T., H. Taylor and P. Gardiner (2002) A Collaborative Analysis Tool for Visualization and Interaction with Spatial Data. in: *7th International Conference on 3D Web Technology - Web3D'02*, Tempe, AZ, 75-83.

Miller, S. (1993) *The Quicktime How-to Book*. San Francisco, CA, Sybex.

Morris, R. A. and L. Khatib (1998) Quantitative Structural Temporal Constraints on Repeating Events. in: *TIME'98*, Sanibel Island, FL, 74-80.

Morris, R. A., W. D. Shoaff and L. Khatib (1996) Domain Independent Temporal Reasoning with Recurring Events. *Computational Intelligence*. 12(3): 450-477.

Najork, M. and M. Brown (1995) Obliq-3d: A High-Level, Fast-Turnaround 3d Animation System. *IEEE Transactions on Visualization and Computer Graphics*. 1(2): 175-193.

Neves, J. N., P. Gonçalves, J. Muchaxo and J. P. Silva (1999) A Virtual Gis Room: Interfacing Spatial Information in Virtual Environments. in: A. S. Câmara and J. Raper (Eds.) *Spatial Multimedia and Virtual Reality*, London, Taylor & Francis, 147-156.

Neves, N., J. P. Silva, P. Gonçalves, J. Muchaxo, J. M. Silva and A. Câmara (1997) Cognitive Spaces and Metaphors: A Solution for Interacting with Spatial Data. *Computers & Geoscience*. 23(4): 483-488.

Ogleby, C. (2002) Virtual World Heritage Cities: The Ancient Thai City of Ayutthaya Reconstructed. in: P. Fisher and D. Unwin (Eds.) *Virtual Reality in Geography*, New York, Taylor & Francis.

OpenGL (1992) *Opengl Reference Manual : The Official Reference Document for Opengl, Release 1, Opengl Architecture Review Board.* Reading, MA, Addison-Wesley.

Osmani, A. (1999) Introduction to Reasoning About Cyclic Intervals. *Lecture Notes in Artificial Intelligence*. 1611: 698-706.

Pentland, A. and J. Williams (1989) Good Vibrations: Modal Dynamics for Graphics and Animations. *Computer Graphics*. 23(3): 215-222.

Peterson, M. (1999) Active Legends for Interactive Cartographic Animation. *International Journal of Geographical Information Science*. 13(4): 375-383.

Raper, J. (2000) *Multidimensional Geographic Information Science*. London, Taylor & Francis.

Reddy, M., L. Iverson and Y. G. Leclerc (2000) Under the Hood of GeoVRML 1.0. in: *Web3D-VRML 2000 Fifth Symposium on Virtual Reality Modeling Language*, Monterey, CA, 23- 28.

Reddy, M., Y. Leclerc, L. Yverson and N. Bletter (1999) Terravision II: Visualizing Massive Terrain Databases in VRML. *IEEE Computer Graphics and Applications*. 19(2): 30-38.

Reynolds, C. W. (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model. *Proceedings of SIGGRAPH'87*. 21(4): 25-34.

Rhyne, T. M. (1997) Going Virtual with Geographic Information and Scientific Visualization. *Computer & Geosciences*. 23(4): 489-491.

Schmitz, P. (2002) Multimedia Meets Computer Graphics in Smil2.0: A Time Model for the Web. in: *WWW 2003*, Honolulu, HW, 45-53.

Strauss, P. (1993) Iris Inventor, a 3d Graphics Toolkit. in: *Conference on Object-Oriented Programming Systems, Languages, and Applications-OOPSLA'93*, Washington, DC, 341-349.

Strauss, P. and R. Carey (1992) An Object-Oriented Graphics Toolkit. *Computer Graphics*. 26(2): 341-349.

Terenziani, P. (2003) Towards a Comprehensive Treatment of Temporal Constraints About Periodic Events. *International Journal of Intelligent Systems*. 18(4): 429-468.

Thalmann, N. M. and D. Thalmann (1985) *Computer Animation: Theory and Practice*. Tokyo, Springer-Verlag.

Thalmann, N. M. and D. Thalmann (1994) *Computer Animation: A Key Issue for Time Visualization*. New York, Academic Press.

Tobler, W. (1970) A Computer Movie Simulation Urban Growth in the Detroit Region. *Economic Geography*. 46(2): 234-240.

USGS (2003) U.S. Geological Survey. URL:http://www.ucgs.org.

Verbree, E., G. V. Maren, R. Germs, F. Jansen and M.-J. Kraak (1999) Interaction in Virtual World Views - Linking 3D GIS with VR. *International Journal of Geographical Information Science*. 13(4): 385-396.

Web3D (2003) Web3d Consortium, URL:http://www.web3d.org.

Weiss, R., A. Duda and D. Gifford (1995) Composition and Search with a Video Algebra. *IEEE Multimedia*. 2(1): 12-25.

Weisstein, E. (2002) *CRC Concise Encyclopedia of Mathematics*, New York, CRC Press.

Wenzel, S. and U. Jensen (2001) The Integration of 3-D Visualization into the Simulation-Based Planning Process of Logistics Systems. *SIMULATION*. 77(3-4): 114-127.

Williams, N. A. (1999) Four-Dimensional Virtual Reality Gis (4d Vrgis): Research Guidelines. in: B. Gittings (Ed.) *Innovations in Gis 6*, London, Taylor & francis, 201-214.

Zeleznik, R., D. Conner, M. Wloka, D. Aliaga, N. Huang, P. Hubbard, B. Knep, H. Kaufman, J. Hughes and A. v. Dam (1991) An Object-Oriented Framework for Integration of Interactive Animation Techniques. *Computer Graphics*. 25(4): 105-112.

Zlatanova, S. (2000) *3D for Urban Development*. Thesis, Graz University of Technology, Graz, Austria.

BIOGRAPHY OF THE AUTHOR

Jorge was born in Salvador, Bahia, Brazil on July 1, 1962. He graduated with a Civil Engineering Degree from Universidade Federal da Bahia - Brazil in 1986. He received his Masters Degree in Civil Engineering from Pontifícia Universidade Católica do Rio de Janeiro – Brazil in 1991. Jorge has worked in the academic and in the public sectors. In the academia, Jorge worked as an Assistant Professor in the Civil Engineering Department of the Universidade Federal da Bahia, and as an Associate Professor in the Computer Science Department of Universidade Salvador. At the Universidade Salvador, Jorge also joined the Computer Network Group, where he developed research in Geographic Information Systems, Virtual Reality, and Scientific Visualization. In the public sector, Jorge worked as a consultant on systems development and computer network administration. In the fall of 2000, he entered the Ph.D. program of the Department of Spatial Information Science and Engineering at the University of Maine.

Jorge Campos is a candidate for the Doctor of Philosophy degree in Spatial Information Science and Engineering from The University of Maine in August, 2004.