


5-2005

Hierarchies for Event-Based Modeling of Geographic Phenomena

Rui Zhang

Follow this and additional works at: <http://digitalcommons.library.umaine.edu/etd>

 Part of the [Geographic Information Sciences Commons](#), and the [Graphics and Human Computer Interfaces Commons](#)

Recommended Citation

Zhang, Rui, "Hierarchies for Event-Based Modeling of Geographic Phenomena" (2005). *Electronic Theses and Dissertations*. 568.
<http://digitalcommons.library.umaine.edu/etd/568>

This Open-Access Thesis is brought to you for free and open access by DigitalCommons@UMaine. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of DigitalCommons@UMaine.

HIERARCHIES FOR EVENT-BASED MODELING OF
GEOGRAPHIC PHENOMENA

By

Rui Zhang

Master of Engineering, Nanjing University, Nanjing, 2002

Bachelor of Science, Nanjing University, Nanjing, 1999

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

(in Spatial Information Science and Engineering)

The Graduate School

The University of Maine

May, 2005

Advisory Committee:

Michael F. Worboys, Professor of Spatial Information Science and Engineering, Advisor

Max J. Egenhofer, Professor of Spatial Information Science and Engineering

Kathleen E. Hornsby, Assistant Research Professor, National Center for Geographic Information and Analysis

© 2005 Rui Zhang
All Rights Reserved

HIERARCHIES FOR EVENT-BASED MODELING OF GEOGRAPHIC PHENOMENA

By Rui Zhang

Thesis Advisor: Dr. Michael F. Worboys

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Master of Science
(in Spatial Information Science and Engineering)

May, 2005

Modeling the dynamic aspect, or change, of geographic phenomena is essential to explain the evolution of geographic entities and predict their future. Event-based modelling, describing the occurrences rather than states of geographic phenomena, gives an explicit treatment of such change, but currently does not have the support of the mechanisms to enable the shifts among different granularities of events. To account for different tasks, a hierarchical representation of the event space at different granularities is needed.

This thesis presents an event-based model; a general framework for representing events based on *precondition* and *postcondition*, using Allen's temporal interval logic. It captures not only the changes to the objects, but also some contextual information that is necessary for the occurrence of events. Analogous to objects, events have types and instances, and two abstraction processes in the object-oriented paradigm, generalization and aggregation, are applied to events. Event-event relations are investigated through their preconditions and postconditions.

Our representation of relationships between events is based on two relations between events, *f*-sequences and *f*-transitions. These relationships play an important role in describing the structure of a component event in the event partonomy, and therefore provide a mechanism to construct the event partonomy automatically. This research constructs an algorithm to generate the part-whole hierarchy for events, which supports multiple representations of events and enables shifts among them. To illustrate the process of constructing the event partonomy, we give a case study of a car accident scenario.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Michael F. Worboys for his encouragement, guidance, and support for my research at all times. I also wish to acknowledge the support and advice from other members of my thesis advisory committee, Dr. Max J. Egenhofer and Dr. Kathleen E. Hornsby.

I want to thank all my fellow graduate students in the Department of Spatial Information Science and Engineering, especially to the members of the Computational GIScience group who showed interests in my thesis and gave me critical feedback: Adam Battson and Lisa Walton. A special thank you goes to Farhan Faisal, Dominik Wilmsen, Caixia Wang for their valuable discussions on my research topic.

I would like to acknowledge Ms. Jane Morse and Dr. Edward Huff for the English proof reading of this thesis. I also thank Ms. Ellen Huff for taking care of my life and improving my English. Finally, this thesis could not have been undertaken without the support and patience of my wife Ms. Han Li and other family members.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
LIST OF FIGURES	vi
Chapter	
1. INTRODUCTION	1
1.1 Background	1
1.2 Research Motivation	3
1.3 Goal	6
1.4 Scope of Thesis	7
1.5 Approach	7
1.6 Major Results	9
1.7 Organization of Remainder of Thesis	10
2. REVIEW OF REPRESENTATIONS OF DYNAMIC PHENOMENA	12
2.1 Ontologies for Spatio-Temporal Information	13
2.2 Formal Models	15
2.2.1 Fluents	15
2.2.2 SNAP-based Models	17
2.2.3 SPAN-based Models	18
2.2.3.1 Event Calculus	19
2.2.3.2 Interval Logic	19
2.3 The Event Hierarchies	22
2.4 Event-Event Relations and the Event Partonomy	24
2.5 Summary	25
3. AN EVENT-ORIENTED MODEL	26
3.1 Event vs. Object	26
3.2 Event Structure	28
3.2.1 Precondition and Postcondition	28
3.2.2 Time	35
3.3 Event Hierarchy	37

3.3.1	Event Taxonomy	39
3.3.2	Event Partonomy	41
3.4	Summary	44
4.	ALGORITHM FOR CONSTRUCTING THE EVENT PARTONOMY	45
4.1	Event-Event Relations	46
4.1.1	Sequence	49
4.1.2	Transition	50
4.2	Constructing the Event Partonomy	51
4.2.1	Generating Sequences and Transitions	53
4.2.2	Aggregating Events	54
4.2.3	Removing Transitions	56
4.2.4	Demonstration	57
4.3	Representations for Composite Events	59
4.3.1	Algorithm	60
4.3.2	Examples	61
4.4	Summary	67
5.	PROTOTYPE	68
5.1	Prototype Design	68
5.2	User Interface	70
5.3	Implementation of Data Structure	71
5.3.1	Database Schema	72
5.3.2	Event Class	73
5.4	Case Study	76
5.5	Summary	82
6.	CONCLUSIONS AND FUTURE WORK	83
6.1	Summary of the Thesis	83
6.2	Results and Major Findings	84
6.3	Limitations of the Model and Future Work	85
6.4	Summary	88
	BIBLIOGRAPHY	89
	BIOBGRAPHY OF THE AUTHOR	93

LIST OF FIGURES

Figure 1.1 A construction event example	2
Figure 1.2 A concept hierarchy for <i>location</i>	4
Figure 1.3 An event hierarchy	5
Figure 2.1 Hierarchical structure for substantial entities	14
Figure 2.2 Functional dependencies for <i>Travel</i> event	22
Figure 3.1 Object-oriented view vs. event-oriented view	27
Figure 3.2 Entity evolution	29
Figure 3.3 Entity creation	30
Figure 3.4 Entity destruction	30
Figure 3.5 Representation of event in decoupled and coupled form	31
Figure 3.6 Context information associated with the construction event	34
Figure 3.7 A rock erosion event	35
Figure 3.8 Deriving the real situation for event occurrences	37
Figure 3.9 An event taxonomy	38
Figure 3.10 An event partonomy	38
Figure 3.11 A refined representation of the event in Fig. 3.6	40
Figure 3.12 An object hierarchy	41
Figure 3.13 Generalization and specialization	41

Figure 3.14	The event partonomy structure	43
Figure 4.1	Ball scenarios	47
Figure 4.2	Transitions between f -sequence and f' -sequence	50
Figure 4.3	Transitions in the ball scenario	52
Figure 4.4	Process of constructing the partonomy	53
Figure 4.5	Different stages during constructing the partonomy	58
Figure 4.6	The event partonomy for the ball scenario	59
Figure 4.7	Process of getting the postcondition for a composite event	60
Figure 4.8	Event composition example: E_1 and E_2	64
Figure 4.9	Event composition example: E_1 and E_3	65
Figure 4.10	Event composition example: E_1 and E_4	66
Figure 5.1	System architecture	69
Figure 5.2	Use case diagram for the system	70
Figure 5.3	Layout of the application window	71
Figure 5.4	Diagram for the Event entity	72
Figure 5.5	Diagram for entities Event1 and Fluent;	73
Figure 5.6	Event structure	74
Figure 5.7	Schematic representation of a car accident scenario	77
Figure 5.8	Atomic events in the car accident scenario and their representations	78
Figure 5.9	The user window for input of event occurrences	79
Figure 5.10	The sequences and transitions for the car accident scenario	80

Figure 5.11	The event partonomy for the car accident scenario	80
Figure 5.12	Composite events in Fig 5.11	81

Chapter 1

INTRODUCTION

1.1 Background

Geographic Information Systems (GISs) capture, model, manipulate, analyze, and present geographically-referenced data (Worboys 1995). In geographic phenomena change is pervasive, and is often brought about by human activities (e.g., changes of people's locations when traveling, and changes in regional cadastral information over time). Applications dealing with changes in geographic phenomena are examples of spatio-temporal applications. Relevant work (Al-Taha and Barrera 1990, Worboys 1994, Peuquet and Wentz 1994, Frank 1994, Egenhofer and Golledge 1994) has been done to model dynamic geographic phenomena by incorporating time in the GISs. An identity-based model of change (Hornsby 1999) gives a systematic treatment of change with the goal of developing and formalizing a set of fundamental changes with respect to the existence or non-existence of identifiable objects. In these efforts, however, change is not captured explicitly, and changes must be deduced from the differences in objects over time.

Another contribution to representing and reasoning about the dynamic world comes from the disciplines of artificial intelligence and natural language computation. This work models the reasoning of intelligent agents as they plan to act in the world, and supports tasks like prediction, planning, and explanation (Shoham and

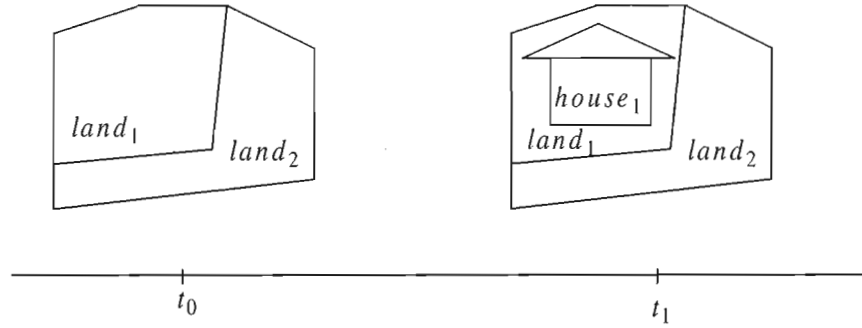


Figure 1.1: A construction event example

Goyal 1988). Work on temporal interval logic (Allen 1984, Allen and Ferguson 1994) provides a more extensive logical framework for representing and reasoning about the dynamic world.

Change is captured by occurrents, things that happen (Grenon and Smith 2004). Processes, events, and actions are all occurrents, but they are defined from different perspectives (Worboys 2005). For simplicity, we take all occurrents as events. In this thesis, we are concerned with physical changes made as a result of dynamic geographic phenomena. An *event* is taken to be a single physical change or a composite of physical changes during a time interval, and it is represented as a pair of situations, before and after the event takes place. For example, Fig. 1.1 schematically shows the construction of $house_1$ on $land_1$ as an event occurrence during the time interval $[t_0, t_1]$, represented in a discrete manner by giving states of objects $house_1$ and $land_1$ at time t_0 and t_1 .

1.2 Research Motivation

Although the event-oriented model (Worboys 2005) gives us a treatment of events when modeling dynamic geographic phenomena, representations of events often require changing from one level of detail to another, so that users can carry out a desired task (Buttenfield and Delotto 1989). The level of detail is referred to as *granularity* (Hobbs 1990, Hornsby and Egenhofer 2002). To efficiently carry out our tasks in dynamic geographic applications, that is, reasoning about our actions, GISs must support representations of events at different granularities together with capability to shift between levels of detail (Buttenfield and Delotto 1989, Hornsby 1999). In this way, each task will be associated with its representations of events at an appropriate level of detail. Monitoring of people’s travel, for example, may need to be examined at different granularities. If users want to query an information system for popular destinations, for example, they should not be asked to analyze detailed information such as individual travel plans. Shifting representations from a finer granularity to a coarser granularity can improve one’s understanding of the information space, while working at a finer granularity may uncover something that is not known (Hornsby 1999).

Our approach to providing multiple representations of events is to provide mechanisms to derive representations at lower levels of detail. In a database system, this may be done through a hierarchy, which expresses the granularities of objects from certain concept. A concept hierarchy (Han and Kambr 2001) “defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts.” This is indicated implicitly through a total or partial order among attributes. For

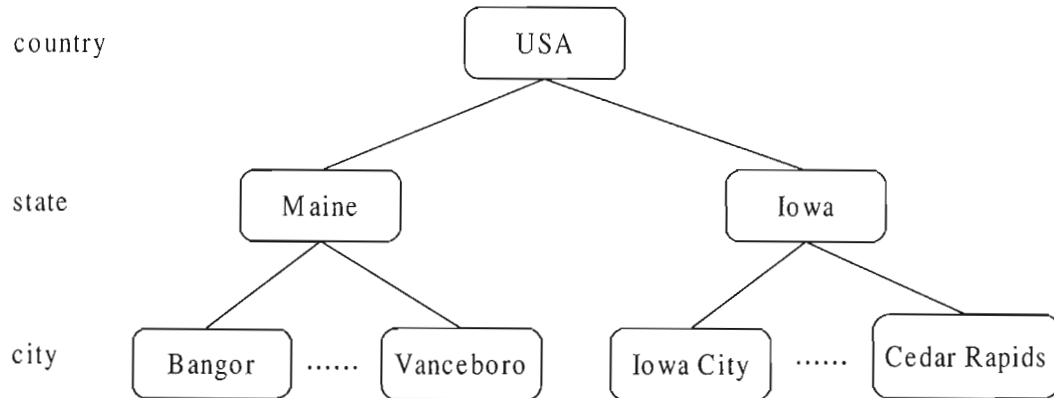


Figure 1.2: A concept hierarchy for *location*

example, the concept *location* is described by attributes *city*, *state*, and *country*. A concept hierarchy for *location* is shown in Fig. 1.2, and it defines a sequence of mapping from a set of cities (low-level concept) to states (higher-level concept), and from a set of states to countries. Concept hierarchies convey how objects are related from concepts (e.g., location) and, thereby, provide background knowledge helping us to derive representations of detailed raw data at coarser levels of detail. A concept hierarchy provides a strategy of data reduction, or an abstraction technique, which is applied to obtain a representation of the data that is smaller in stored volume, yet closely maintains the integrity of the original data. In general, a concept hierarchy is represented as a tree, but it may also be in the form of a partial order.

After events are introduced as primitives in the event-oriented model, we can construct event hierarchies to hierarchically structure the event space.

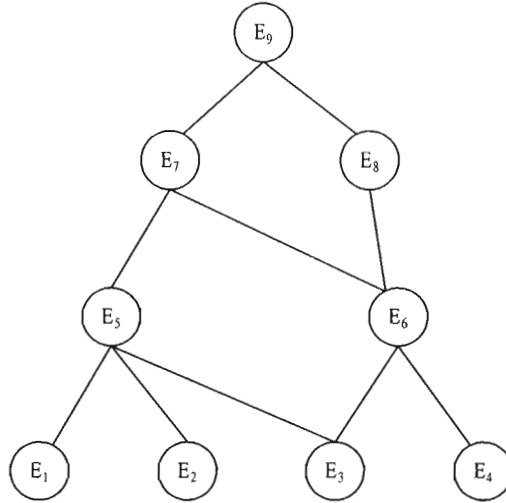


Figure 1.3: An event hierarchy

Definition 1.1 (event hierarchy). Let \mathcal{E} be the event space, which includes a set of events we are concerned with according to some application domain. An *event hierarchy* defines a sequence of mappings from low-level events to higher-level events through a partial order relation.

An event hierarchy can be depicted by a Hasse diagram or a directed acyclic graph (DAG) with nodes representing events in \mathcal{E} and edges representing partial order between events at different levels of detail. When shifting from a low-level event to a higher-level event, we get a coarser representation of the event. Instead of drawing an arrow from a low-level event to a higher-level event, we place the higher-level event higher than the low-level event and draw a line between them. Each node in the event hierarchy may have multiple parents. For example, Fig. 1.3 shows an event hierarchy for the event space $\mathcal{E} = \{E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8, E_9\}$, and the

partial order relation is

$$\{(E_1, E_5), (E_2, E_5), (E_3, E_5), (E_3, E_6), (E_4, E_6), (E_5, E_7), \\ (E_6, E_7), (E_6, E_8), (E_7, E_9), (E_8, E_9)\}$$

An event hierarchy enables shifting between levels of detail: a fundamental requirement for reasoning. This hierarchy may be provided by system users, domain experts, and/or knowledge engineers. Its manual production may be time-consuming and tedious; automatic generation of the hierarchy dynamically according to users' perspectives is highly preferred.

1.3 Goal

The goal of this thesis is to provide different representations of dynamic geographic phenomena. This is achieved through automatic generation of an event hierarchy, which enables shifting between different granularities. Specifically, we are investigating the following questions:

1. **What is the underlying structure for level of detail of dynamic geographic phenomena?**
2. **What is needed to construct this structure automatically?**

The construction of the hierarchy depends on the model used for dynamic geographic phenomena. Answers to the first question may be investigated by defining an event-based model, in which events are introduced as primitives. The basic task here is to give a framework for the representation of events and to provide event operations for deriving the representation of events at lower levels of detail. The answer to

the second question may be found by investigating various relations between events. So, primary goals of this work are to: (1) define a model for representing events, (2) investigate relationships between events, (3) propose algorithms to construct the hierarchy.

1.4 Scope of Thesis

From an ontological perspective, entities in dynamic geographic phenomena are divided into continuants and occurrents (Grenon and Smith 2004). In this thesis, we may use the word “object” to refer to a continuant entity, and take all occurrents as events. It is necessary to model both objects and events for the dynamic world, if static and dynamic aspects of the world are both be represented. These two categories of entities are relevant to each other, since events cannot exist without the participation of objects, and conversely. We do not cover the representations of objects in this thesis, and the discussion of different relations between objects and events (Grenon and Smith 2004, Worboys and Hornsby 2004) is also beyond the focus of this thesis.

1.5 Approach

In this research, we use the event-oriented model (Worboys 2005); applying concepts from the object-oriented model, but treating events as though structurally similar to objects. An *event type (event)* characterizes common properties of a set of *event occurrences (occurrences)*. A framework for representing events is proposed based on the concepts of *precondition* and *postcondition*. These two concepts are borrowed

from the field of software programming (Gries 1981), but they have different definitions in this thesis. Preconditions and postconditions describe the states of objects before and after an event occurs. The postcondition records the states of the objects after they are changed, while the precondition records these objects' initial states and also the states of relevant objects that do not appear in the postcondition (Their formal definitions will be given in Chapter 3).

In the object-oriented approach, two methods, *generalization* and *aggregation*, are used to abstract excessive information at different granularities (Smith and Smith 1977). In the event-oriented approach, they can also be applied to events, producing the *event taxonomy* and *event partonomy*.

Generalization involves the relating of a collection of abstractions taxonomically based on similarities, and the event taxonomy enables us to switch to coarser representations based on similarities within some collection of finer representations. In the event taxonomy, “is-a” forms a special relationship between events.

In an event partonomy, events are aggregated into a composite event. A composite event provides a higher-level abstraction of what is happening by suppressing the details of its component events. This is similar to *amalgamation* (Stell and Worboys 1999), a process of collapsing two or more objects into a single object at some coarser granularity. The construction of the event partonomy could be achieved by *aggregation*, combining several events into a higher-level event. This process is typically required when two representations differ in levels of detail, since some events may be viewed either as a single composite event or as a set of component events.

Aggregating events also involves other challenges, for example, discovering the semantics associated with these events and their relationships. After events are introduced into the system as primitives, event-event relations become important (Worboys 2003). Michotte (1963) argues that causal relations between events are important for people to perceive and understand events. Events possess a causal perceptual structure as well as a partonomic perceptual structure, with which the causal perceptual structure is highly correlated (Zacks and Tversky 2001). So causal relations provide important contextual information, allowing events to be aggregated, and facilitating the automatic construction of a hierarchy.

1.6 Major Results

The major result of this thesis is a general framework for representing events based on precondition and postcondition. Two types of event hierarchies are developed, an *event taxonomy* and an *event partonomy*. These provide multiple representations of events and enable shifting between different granularities. An event partonomy is generated automatically, by aggregating events based on two event relations, *sequence* and *transition*. Taxonomies and partonomies are developed to capture the relations we perceive between events. We also propose algorithms, both to construct the event partonomy and to derive representations of composite events. To implement the process of constructing event hierarchies, a prototype is designed, and applied to an example in order to demonstrate our model.

1.7 Organization of Remainder of Thesis

The remainder of this thesis is organized as follows: Chapter 2 describes work on two different ontologies for spatio-temporal information systems, SPAN and SNAP. SPAN is an ontology of explicit event entities. Different approaches to representing events are reviewed, including the event calculus and temporal interval logic. The chapter also briefly describes work on people's perceptual representations of event hierarchies according to causal relations between events.

In Chapter 3, an event-oriented model is discussed, in which concepts from object-oriented models are applied to events. The temporal interval logic studied in Chapter 2 is used to represent events. Two event hierarchies, the event taxonomy and the event partonomy, are described. These are shown to support multiple representations of events and enable shifts between them.

In Chapter 4, causal relations between events, used by people when aggregating events, is investigated from a formal perspective. We develop two event relations between events: *sequence* and *transition*, which are used to approximate to causal relation. In addition, an algorithm is given to automatically construct an event partonomy.

Chapter 5 describes a prototype implementation using the Oracle platform. The system generates the event partonomy according to a set of event occurrences as defined by the user. To show how the system works, a case study is given that demonstrates the process of constructing an event partonomy for an example application.

In Chapter 6, we review and evaluate the objective, methodologies, and major results of this thesis. In addition, we give some recommendations for future work.

Chapter 2

REVIEW OF REPRESENTATIONS OF DYNAMIC PHENOMENA

Spatio-temporal databases and other spatio-temporal knowledge systems are representations of a reality that is inherently dynamic and characterized by continuous change (e.g., your journey, or the El Nino phenomenon). One aspect of change is atemporal difference, such as differences in temperature in different parts of the human body. Another aspect of change is given by differences in objects' spatial or aspatial properties over time (Mortensen 2002). This *temporal change* is the focus of the remainder of this thesis.

Temporal changes are pervasive in dynamic geographic phenomena. They need to be introduced as primitive elements. A change occurs if and only if there exists a proposition p and distinct times t and t' such that p is true at t , but false at t' (Worboys 2001). Since objects preserve their identities through change, we can trace change by comparing states of objects at different times (Hornsby 1999).

This chapter reviews previous work on two different aspects of an ontology for dynamic geographic phenomena, in which different primitives are recognized as distinguishing between changes and objects. We also review previous work on for-

mal models of dynamic phenomena, including situation calculus, event calculus, and interval logic. Then we describe the work on perceptual representations of event hierarchies and causal relations between events.

2.1 Ontologies for Spatio-Temporal Information

To distinguish between changes and objects, entities in reality are divided into two categories: continuants (e.g., a mountain), existing in full at every instant during their lifetime, and occurrents (e.g., a football match), which could never exist in full at any single moment (Grenon and Smith 2004). Continuants do not have temporal parts, though they may have spatial parts. They may change over time, but they always preserve their identities.

Based on continuants and occurrents, two different perspectives of an ontology for dynamic phenomena, SNAP and SPAN, have been developed (Grenon and Smith 2004). From the perspective of SNAP, only continuants (SNAP entities) can be recognized, and reality is decomposed into 3-dimensional spatial slices. From the perspective of SPAN, occurrents (SPAN entities) are recognized and evolve or unfold themselves during their lifetime.

Grenon and Smith (2004) categorize SNAP entities as substantial entities, SPQR entities, and spatio-temporal regions. The substantial entities include substances and their fiat parts, boundaries, aggregates, and sites (Fig. 2.1). SPQR entities are dependent on substantial entities; that is, there is no color for the car unless the car exists. However, they are not confined to attributes or properties; they also include relationships among substantial entities. Spatio-temporal regions can be either

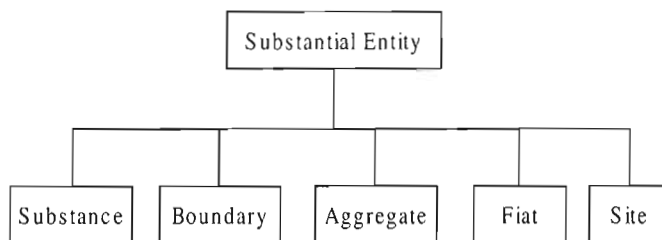


Figure 2.1: Hierarchical structure for substantial entities
(Grenon and Smith 2004)

substantial entities or attributes of some substantial entity. SPAN entities are also categorized as perduring processual entities, temporal regions, and spatio-temporal regions. Processual entities include processes and their fiat parts, boundaries, aggregates, and settings.

The terms “event,” “process,” and “action” are used to specify the SPAN entities, and some difficulties arise concerning the meaning of these terms and the differences between them (Worboys 2005). For simplicity, “event” is chosen as the term for SPAN entities, and “object” is chosen for SNAP entities. From the view of SPAN, the dynamics of spatio-temporal phenomenon are understood as events and relations arranged among them. To model dynamic systems, an event is used to denote the scenario of a single change or a composite of changes to objects (Worboys 2001). Such a definition of “event” is different from that in the event-based spatio-temporal data model (Peuquet and Duan 1995), in which an event only is used to denote changes to cells on a single thematic domain (e.g., land use of the population).

In summary, the SPAN ontology makes the notion of change explicit, and it is an important way in which we humans experience reality in our lives (Bittner 2002). Therefore, it is necessary to use a SPAN ontology when representing dynamic phe-

nomena. In general, most current systems are based on a SNAP approach. Grenon and Smith (2004) argue that a good ontology should incorporate both continuants and occurrents, so that both the static and dynamic aspects of the world can be represented.

2.2 Formal Models

This section describes two kinds of formal models for representing the dynamic world, SNAP-based and SPAN-based, as a result of two different views of the ontology for dynamic phenomena. In the SNAP-based models, changes are derived from the temporal sequence of objects, their attributes, and relations. SPAN-based models explicitly describe changes as properties of events. To begin with the discussions on these two types of models, we first introduce the notion of fluent.

2.2.1 Fluents

Fluents model time-varying SNAP entities, such as the existence and color of a car. Such configurations constitute situations. A *situation* is defined using a situation function σ , which specifies a state of the world (situation) for each instant of time. Formally we have

$$\sigma : \mathbf{T} \rightarrow \mathbf{Sit}$$

where \mathbf{T} is the time domain and \mathbf{Sit} is the set of situations. There is a one-to-one mapping from \mathbf{T} to \mathbf{Sit} if a linear time model is considered.

A fluent f is a function from situations to values (McCarthy and Hayes 1969).

$$f : \text{Sit} \rightarrow \mathbf{Range}$$

Each fluent has a codomain **Range**, specifying a set of values that SNAP entities can take in different situations. Galton (2000) simplifies the definition of a fluent by omitting all explicit references to situations and defines it to be a function from time to values.

$$(2.1) \quad f : \mathbf{T} \rightarrow \mathbf{Range}$$

For example, $\text{color}(\text{car})$ is a fluent:

$$(2.2) \quad \text{color}(\text{car}) : \mathbf{T} \rightarrow \mathbf{Color}$$

in which the range is **Color**, a set of color values.

Fluents describe SNAP entities and are classified according to the types of their ranges. For example, a particular world might be specified by, among other things, whether or not a car is moving (a Boolean fluent), the position of the car (a spatially-valued fluent), and the speed of the car (a float-valued fluent). A fluent is called a *Boolean fluent* if its range is a set of truth values (i.e., *true* or *false*), denoted by \mathcal{B} .

Any fluent gives rise to Boolean fluents. Given a fluent f (Eq. 2.1), its Boolean fluents may be defined by

$$s \equiv_{def} (f = v), v \in \mathbf{Range}$$

For each value v in **Range**, the Boolean fluent s is assigned *true* if f takes v , and *false* to the rest (Galton 2000).

$$s : \mathbf{T} \rightarrow \mathcal{B}$$

$$s(t) \equiv_{def} (f(t) = v)$$

For example, fluent $\text{color}(\text{car})$ enables us to get specific Boolean fluents, such as $\text{color}(\text{car}) = \text{white}$.

$$\text{color}(\text{car}) = \text{white} : \mathbf{T} \rightarrow \mathcal{B}$$

So the Boolean fluent records certain states of the the SNAP entity (i.e., $\text{color}(\text{car})$) when it is true. To access what SNAP entity has been recorded by s , we use the predicate $\text{fluent}(s)$, for example, $\text{fluent}(\text{color}(\text{car}) = \text{white}) = \text{color}(\text{car})$. In general, Boolean fluents are specified by $\text{holds}(s, t)$, specifying whether a Boolean fluent s is true at a particular time t

$$\text{holds} : \mathbf{S} \times \mathbf{T} \rightarrow \mathcal{B}$$

where \mathbf{S} is the set of Boolean fluents.

A fluent is essentially a temporal field allowing us to create events in just the same way we create objects from a spatial field (Galton 2003). In SNAP-based models, Boolean fluents are used to give states of objects, while in the SPAN-based models they are used to trace changes when representing events.

2.2.2 SNAP-based Models

In the situation calculus (McCarthy and Hayes 1969), the world is represented by a sequence of situations, and *actions* are defined to be transitions between situations. A situation can be tracked through an initial situation and definitions of actions. Given an initial situation and a sequence of actions, the initial situation will be transformed into the result situation through a composition of the actions.

There is no explicit temporal model in the situation calculus and actions are assumed to be point-based, occurring one after another. So it is difficult for the situation calculus to deal with temporally complex actions (e.g., actions that overlap). The situation calculus can be considered as a point-based temporal logic with a branching time model. No assertion exists during any action, and there is no mechanism to represent knowledge about how long an effect will hold.

The identity-based approach (Hornsby 1999) gives a set of primitives that define a set of fundamental change operations, which are derived from the identity status of an object and are based on the notion of existence. In this work an object's status can be one of: object existence, object non-existence without history, and object non-existence with history. Each object has a temporal sequence that describes its states during its lifetime. Temporal change is captured by transitions between adjacent object states within this sequence. Object states and changes during an object's lifetime can be tracked by means of the object's identity. In this model, change is based on the notion of existence and nonexistence of objects, so it may not be appropriate for other types of change, e.g., changes of object color or shape. In addition, change is recorded qualitatively, based on temporal order, and no quantitative or metric measures are employed to represent the temporal interval in which any change is taking place.

2.2.3 SPAN-based Models

In SNAP-based models, basic entities are continuants and their states, while in the SPAN-based models basic entities characterize change.

2.2.3.1 Event Calculus

The *event calculus* (Kowalski and Sergot 1986) is a temporal formalism to model scenarios in which basic units are events and each event is an instance of an event type. The basic concepts are *event* and *property*. A Boolean fluent, p , describes a time-varying property. Two predicates, $Initiate(e, p)$ and $Terminate(e, p)$, are used to describe the relationships between events and properties; event occurrence e initiates the property p , and event occurrence e terminates the property p . In the event calculus, p holds over a time period, ranging from the time point when an event e initiates p to the time point when another event e' terminates p . No event occurring between the occurrence of e and e' affects p .

2.2.3.2 Interval Logic

In the event calculus, events are instantaneous. To extend the event calculus, Allen (1984) proposed the temporal interval logic based on the calculus of temporal intervals (Allen 1983).

In interval logic, the predicate $holds(s, t)$ is used to indicate that the Boolean fluent s is true during t . Time intervals rather than time points are used as primitive elements in the interval logic (Allen 1984). This arises from the observation that times of events or states may be decomposed into subtimes. For example, in the event that Mary prepares dinner, it is possible to look more closely at its occurrence and decompose the event into its component events, such as the event that Mary washes vegetables and the event that Mary cooks vegetables. We cannot look more closely at an instantaneous event, because time instants cannot be further decomposed.

It is not necessary to introduce both time points and time intervals as primitives into the temporal logic, since this would force a decision as to whether the time intervals are open or closed, leading to inconsistency and truth gap problems (Allen 1984, Galton 1995). McDermott (1982) takes time points as primitives and defines intervals in terms of their endpoints. In the famous example of light-on and light-off, the light is off in the interval $t_1 = [a, b]$, and at the point b we push the switch, then the light is on during the interval $t_2 = [b, c]$. If both time points and time intervals are introduced as primitives, we need to know the state of the light at time point b . Considering the intervals to be open leads us to reason that the light is neither on nor off at time point b ; whereas considering the intervals to be closed leads to the conclusion that the light is both on and off at time point b .

There are several ways to introduce time intervals into logical formalisms: first-order logics with temporal arguments, and reified temporal logics (Allen and Ferguson 1994). The latter “reify” standard propositions of the classical first-order language as objects denoting propositional terms.

In first-order logics, a time interval is associated with each predicate directly. This is, however, insufficient to represent events, because they require potentially unbounded qualifications (Davidson 1967). For example, the event of Tom travelling from New York to Boston (E_{Travel}) might be asserted to occur during t_1 , using the predicate $\text{Travel}(\text{Tom}, \text{New York}, \text{Boston}, t_1)$. The problem arises when representing the event of “Tom travels from New York to Boston by taking the Greyhound” (E_{Travel1}). We can either introduce another predicate to represent this kind of event, like $\text{Travel1}(\text{Tom}, \text{New York}, \text{Boston}, \text{Greyhound}, t_1)$, which has an additional argument

for the means of transportation, or we can extend the predicate *Travel*, and use it to represent both kinds of events. In the former case, we produce a lot of predicates, such as *Travel* and *Travel1* describing essentially the same thing. In the latter case, we have a lot of argument positions left unspecified in some particular event description. Actually, we cannot give a limit to the number of argument positions needed. It may appear that we always need another argument to add more information that has not yet been captured (Allen and Ferguson 1994).

In reified logics, propositions are related to times through a truth predicate such as *holds*. For example, $holds(shape(rock_1) = s_1, t_1)$ means the shape of object $rock_1$ during t_1 is s_1 , or the Boolean fluent $shape(rock_1) = s_1$ is true during t_1 . Therefore, returning to the above travel example, we can represent E_{Travel} using Eq. 2.3, and represent $E_{Travel1}$ using Eq. 2.4. Two predicates $begin(t_1)$ and $end(t_1)$ are used to retrieve zero-duration intervals at the beginning and end of t_1 . $location(Tom)$ specifies the location of Tom and $transportation(Tom)$ specifies the transportation that Tom takes.

$$(2.3) \quad \begin{aligned} E_{Travel} &\equiv holds(location(Tom) = New\ York, begin(t_1)) \\ &\quad \wedge holds(location(Tom) = Boston, end(t_1)) \end{aligned}$$

$$(2.4) \quad \begin{aligned} E_{Travel1} &\equiv holds(location(Tom) = New\ York, begin(t_1)) \\ &\quad \wedge holds(location(Tom) = Boston, end(t_1)) \\ &\quad \wedge holds(transportation(Tom) = Greyhound, t_1) \end{aligned}$$

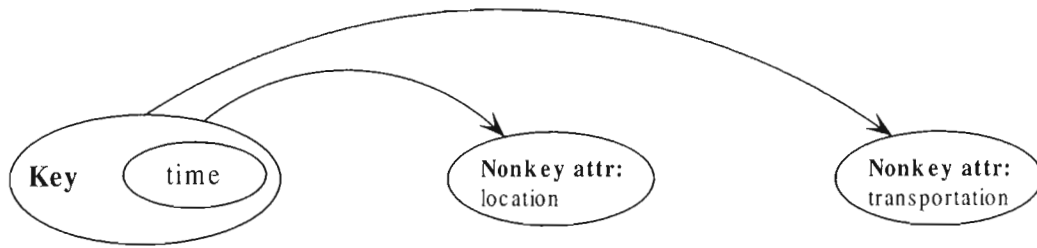


Figure 2.2: Functional dependencies for *Travel* event
 (Eclipses denote sets of attributes and arcs indicate functional dependencies)

This approach is very similar to decomposing a relation schema into a collection of Boyce-Codd Normal Form (BCNF) relation schemas, based on functional dependencies (Ramakrishnan and Gehrke 2000) (Fig. 2.2).

2.3 The Event Hierarchies

A hierarchy is one of the most common techniques for organizing and structuring complex systems (Hirtle and Jonides 1985, Hirtle 1995, Timpf 1999). In a hierarchy, a system is subdivided into smaller subsystems, and further subdivision may be recursively repeated as long as each subdivision makes sense (Koestler 1967). Physical reality can be organized through hierarchies, providing representations at levels of detail, from subatomic reality to galaxies. Levels of detail that humans can comprehend directly are in the middle of that range (Gibson 1986). Events also have hierarchies, which provide multiple representations at different levels of detail from simple physical changes to large-scale events (Zacks and Tversky 2001).

In perceptual psychology, an analogy exists between objects and events and events belong to categories and have parts (Zacks and Tversky 2001). Events can be organized into event taxonomies. There exists a special relation, *is-a*, between events at different taxonomic levels. For example, if E_1 is the event of Tom going to school by bus and E_2 is the event of Tom going to school, we may write $is-a(E_1, E_2)$. The number of features for events increases from superordinate to subordinate levels in the event taxonomy. An event partonomy is developed through the *part-of* relation between events. For example, if E_3 is the event of Tom getting to the bus station, we may write $part-of(E_3, E_1)$. Barker and Wright (1954) suggest that events are in fact perceived by people as partonomically organized. During their experiments, small units with fine detail in the event partonomy tended to be related to minor subgoals and thus went unnoticed by the participants. Large units with coarse detail, which may be related to large ongoing goals, also often went unnoticed. Between these two extremes were units that Baker and Wright described as behavior episodes. Behavior episodes are analogous to physical objects like tables, which can be seen with the “naked eye.” People develop the event partonomy of an ongoing activity as it happens, by aggregating behavior episodes into higher-level events, some of which may later be further aggregated (Zacks and Tversky 2001). When considering higher-level events, details of their component events are suppressed. Relations between component events play a role in describing higher-level events of which they may be part.

2.4 Event-Event Relations and the Event Partonomy

In the event partonomy, higher-level events are characterized by their component events and by relations between component events. These relations might help us construct event partonomies automatically.

Spatial and temporal relations between events provide clues to appropriate types of event aggregation, but are not the main factors for us to aggregate component events. For instance, the events of a traffic light turning red (E_1) and the stopping of the traffic at the intersection (E_2) can be aggregated into a higher-level event (i.e., traffic stop in front of the traffic light). In this scenario people tend to believe that the temporal relation, E_1 occurs before E_2 , determines the aggregation of events. However, the temporal relation between the occurrences of E_1 and E_2 is just a result of another type of event relation. If the event of Tom waking up (E_3) occurs right after E_1 , aggregating E_1 and E_3 together does not help us to understand the scenario. Similarly, the same problem arises when we try to aggregate events through their spatial relations. For example, it is not necessary to aggregate the events of Tom going from A to B (E_4) and Mary going from A to B (E_5), even though both E_4 and E_5 occur in the same spatial location (i.e., the route from A to B). But if there is another event of a meeting between Tom and Mary at B, the possibility of aggregating E_4 and E_5 increases.

Pearl (2000) notes that causal relation is the stable mechanism that organizes the world, and is the key relation for aggregating events. The common sense view of causal relations is that one event (the cause) is supposed to bring about or produce an occurrence of a second event (the effect). There are two views on causality:

whether a single process of perceiving causality is innate, or whether it is learned. The philosopher Hume (1739) argues that perceiving causality is a learned process, from repeated observations of the conjunction of two events, their spatio-temporal contiguity, and the temporal priority of one event relative to the other. It is not possible to “prove” the existence of causality. Kant (see Kant 1781) argues that causality is an innate and core part of cognition. The essence of causality is an “ampliation” of movement, in which the motion of the first object is transferred to the second object. Causality is characterized by a tension between the individuality of the objects and the perceptual integrity of motion that transfers from one object to the other, which corresponds to “contour discontinuities” in the temporally extended events (Zacks and Tversky 2001).

2.5 Summary

Previous work on the two ontological perspectives, SNAP and SPAN, of dynamic geographic phenomena is described in this chapter. These two ontologies relate to different formal models. Although SNAP-based and SPAN-based models can be used to describe the same geographic world, they differ in that only SPAN-based models represent changes explicitly, through events and their attributes.

Analogous to continuant objects, events also have hierarchies, providing multiple representations of events and enabling shifts between them. Finally, relations among events are discussed, which provide the key to the process of constructing event partonomies.

Chapter 3

AN EVENT-ORIENTED MODEL

The storage and representation of dynamic geographic phenomena depends on the data model used. In an event-oriented model, events that capture changes are introduced as primitives, and they are treated in the same way as objects are treated in the object-oriented model. Concepts from the object-oriented model such as classes and instances are applied to events. In this chapter, we develop a framework to represent events based on the concepts of *precondition* and *postcondition*. An event occurrence is instantiated by assigning the event a time value. We also discuss two event hierarchies, *event taxonomy* and *event paratomy*, which enable us to organize the information of the event space at different levels of detail. Properties of these hierarchies are also investigated.

3.1 Event vs. Object

From an object-oriented view, the world is viewed as a collection of objects, with properties, behaviors, and relationships between each other. However, the object-oriented approach is not capable of capturing the dynamic aspects of geographic phenomena. From an event-oriented view, dynamic geographic phenomena are viewed as collections of events, with properties and relationships between each other.

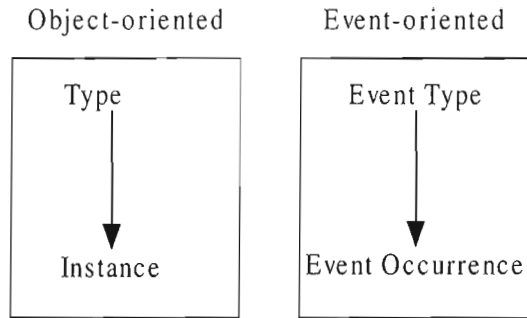


Figure 3.1: Object-oriented view vs. event-oriented view

There is a difference between the events used in this thesis and those defined in programming languages (e.g., a mouse click event). In programming languages, events are considered as special objects with attributes such as sender, receiver, and type. However, such attributes only tell static information about events without giving us any idea on how these events occur. In this thesis, events are considered as composites of physical changes. These changes constitute patterns that may occur repeatedly. A group of occurrences of the same event type share the same pattern of changes, just like objects (or events defined in programming languages) of the same class have the same attributes. Similar to the concept of object type, *event type* (or *event*) is used to specify the pattern, while *event occurrence* (or *occurrence*) is used to denote the instance of the relevant event. Fig. 3.1 shows the corresponding concepts at the same level in the object-oriented and event-oriented world. We use notation E_1, E_2, \dots to denote events. Occurrences of event E_i are denoted by e_{i1}, e_{i2}, \dots . In the following section, we will discuss the framework for representing events and occurrences.

3.2 Event Structure

In the object-oriented model, objects are recognized through their distinctive characteristics, such as shapes, colors, textures, and tactile properties. By defining classes using attributes and methods, we represent common characteristics of the objects belonging to the same class. Some of the examples of attributes or instance variables are name, shape, and position. **In our event-oriented model, an event is defined by its precondition and postcondition, a component events list, and a time attribute.**

An event denotes a composite of physical changes. Occurrences of the same event have the same pattern of changes, but they are independent, different from each other, and have distinct occurrence times. Event occurrences have temporal boundaries, and are modeled using two sorts of time. We use *local time* to specify the time interval that is relative to the starting point of the event, and *global time* to specify the time interval of the event occurrence. For example, the local time of an event may be the time slot $[8, 9]$. The global time of a specific occurrence may be specified as a time interval $[t + 8, t + 9]$ where t is the time point when the event starts to occur.

3.2.1 Precondition and Postcondition

An event is characterized by changes. We can trace these temporal changes by comparing properties of objects at different times. Changes in an event are viewed in a discrete manner through sampling the dynamic world, and are characterized by different states of objects at the beginning and end of the event occurrence. Based on

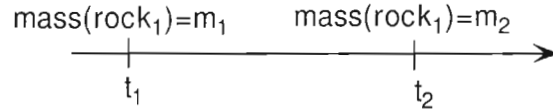


Figure 3.2: Entity evolution

the notion of existence and non-existence, changes in an event could be the creation or destruction of the object (Hornsby 1999). However, changes may include the alteration of attributes of an object or the alteration of relations between objects.

In order to model changes, we use fluents, which describe SNAP entities. Each fluent gives rise to Boolean fluents, which record SNAP entities during the time when the Boolean fluents are true (Sec. 2.2.1). Based on Boolean fluents, we can represent changes by giving different values of a SNAP entity at the beginning and end of the event occurrence. For another example, fluent $\text{mass}(\text{rock}_1)$ describes the volume of rock_1 at different times.

$$\text{mass}(\text{rock}_1) : \mathbf{T} \rightarrow \mathfrak{R}$$

This fluent gives rise to such Boolean fluents as $\text{mass}(\text{rock}_1) = m_1$ and $\text{mass}(\text{rock}_1) = m_2$. As Fig. 3.2 shows, the erosion of rock_1 is represented by the evolution of the volume of rock_1 , captured by comparing the state of $\text{mass}(\text{rock}_1) = m_1$ during t_1 with $\text{mass}(\text{rock}_1) = m_2$ during t_2 . Both time intervals t_1 and t_2 belong to *local time*. Fig. 3.3 shows the creation of rock_1 , and Fig. 3.4 shows the destruction of rock_1 . However, the description of changes itself is not enough to characterize an event, since the event is associated with contextual information, which distinguishes itself from other events involving the same kind of changes.

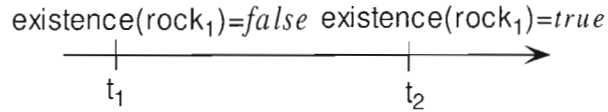


Figure 3.3: Entity creation

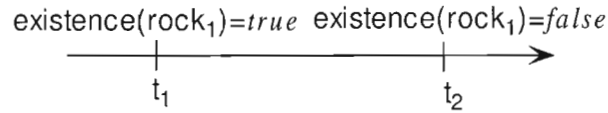


Figure 3.4: Entity destruction

According to Davidson (1969), events are identified through their causes and effects. However, it is difficult to capture the causes of events, that is, a set of states that definitely lead to occurrences of the events. For example, a lit cigarette left unattended in the forest may cause a big forest fire. But the lit cigarette itself is not the whole cause of the forest fire; other environmental issues (e.g., humidity and wind) are also involved. The effects of the event, that is, a complete set of states describing the change, can only be partially known.

Instead of formalizing the cause and effect of an event, we use two ‘weaker’ concepts, precondition and postcondition, shifting from the necessary and sufficient condition of the occurrence of an event to the necessary condition. In order to get the precondition and postcondition, we first obtain the necessary condition for the event to take place. This condition characterizes changes and some initial states of objects, and it is a conjunction of Boolean fluents in which recordings of the same SNAP entity are coupled together (We call this the *coupled form*). The next step is

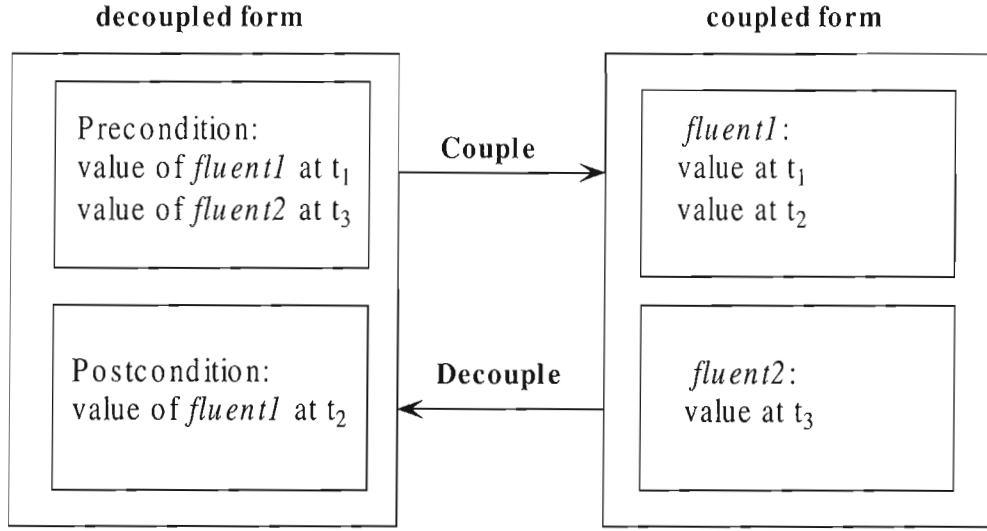


Figure 3.5: Representation of event in decoupled and coupled form

to decompose the coupled form into the precondition and postcondition of the event (Fig. 3.5). To describe the process, we begin with definitions of precondition and postcondition.

Definition 3.1 (*precondition*). The precondition of an event is a proposition that must be true for the event to occur, and it satisfies:

- (i) the precondition is in the form of $holds(s_1, t_1) \wedge \dots \wedge holds(s_n, t_n)$ where $holds(s_i, t_i)$, $i = 1, \dots, n$ specifies the local time interval t_i when the state s_i holds, and

$$\forall i, j \in \{1, \dots, n\} \ i \neq j \Rightarrow fluent(s_i) \neq fluent(s_j)$$

- (ii) s_i describes the initial states of objects that change during the event and also the states of objects that do not change.

Definition 3.2 (postcondition). The postcondition of an event is a proposition that must be true immediately after the event occurs, and it satisfies:

- (i) the postcondition is in the form of $holds(s'_1, t'_1) \wedge \dots \wedge holds(s'_m, t'_m)$ where $holds(s'_i, t'_i)$ $i = 1, \dots, m$ specifies the local time interval t'_i when the state s'_i holds, and

$$\forall i, j \in \{1, \dots, m\} \ i \neq j \Rightarrow fluent(s'_i) \neq fluent(s'_j)$$

- (ii) s'_i describes the end states of objects appearing in the precondition that are changed during the event. Formally, we have

$$\forall i \in \{1, \dots, m\} \ \exists j \in \{1, \dots, n\} \ fluent(s'_i) = fluent(s_j)$$

Note that no fluents can be introduced in the postcondition that are not in the precondition. The changes in values of fluents in the pre- and postcondition characterize the change brought about by the event.

Definition 3.3 (event). An *event* E is defined by its precondition and postcondition

$$(3.1) \quad E \equiv_{def} \langle PRE_E, POST_E \rangle$$

Below is a typical form of a pair of precondition and postcondition of E

$$(3.2) \quad E \equiv_{def} \langle holds(s_1, t_1) \wedge \dots \wedge holds(s_m, t_m) \\ \wedge holds(s_{m+1}, t_{m+1}) \wedge \dots \wedge holds(s_n, t_n), \\ holds(s'_1, t'_1) \wedge \dots \wedge holds(s'_m, t'_m) \rangle$$

There is a parallel here with the use of precondition and postcondition in programming language theory. In order to reason about the correctness of computer programs, Hoare (1969) introduced preconditions and postconditions, which are formulas in first-order logic. Whenever the precondition for the program is true, the postcondition must be true after the program executes. Given a small piece of the code, for example, $y := x + 5$, a possible pair of pre- and postconditions for the statement is:

$$\{x = 2\} y := x + 5 \{x = 2 \wedge y = 7\}$$

Our use of pre- and postconditions differs from the above in two ways. Being a result of the program, postcondition depends on its precondition, which varies depending on the value for variable x . Those variables that do not change during the execution of the program also appear in the postcondition. In our work, the precondition does not vary and must be true for the occurrence of an event, and the SNAP entity will not appear in the postcondition if it does not change.

We get the precondition and postcondition of the event from the necessary condition for its occurrence in the coupled form through the *decoupling process*. The *holds* predicates recording the same SNAP entities will be distributed into the precondition and postcondition according to the temporal order of local times of the *holds* predicates. The remainder of the *holds* predicates will be added to the precondition. In general, the number of propositions in the precondition is greater than or equal to the number of propositions in the postcondition.

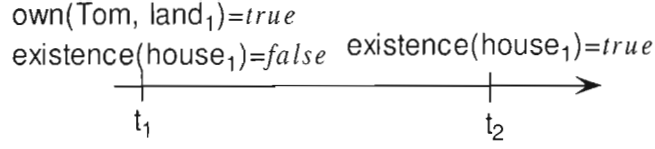


Figure 3.6: Context information associated with the construction event

For example, Fig. 3.6 shows the representation of the event of the construction of $house_1$ on $land_1$ owned by Tom. The necessary condition of its occurrence (Eq. 3.3) not only focuses on the object $house_1$ that comes into existence because of the event, but also describes the ownership relation between Tom and $land_1$, though this ownership does not change during its occurrence. The ownership works as contextual information associated with the event of constructing $house_1$ on $land_1$.

$$(3.3) \quad \begin{aligned} & holds(\text{existence}(house_1) = \text{false}, t_1) \wedge holds(\text{existence}(house_1) = \text{true}, t_2) \\ & \wedge holds(\text{own}(\text{Tom}, \text{land}_1) = \text{true}, t_1) \end{aligned}$$

After we decouple Eq. 3.3, the event shown in Fig. 3.6 is represented by the precondition-postcondition pair:

$$(3.4) \quad \begin{aligned} & < holds(\text{own}(\text{Tom}, \text{land}_1) = \text{true}, t_1) \wedge holds(\text{existence}(house_1) = \text{false}, t_1), \\ & holds(\text{existence}(house_1) = \text{true}, t_2) > \end{aligned}$$

A SNAP entity is not allowed to be recorded more than twice in the necessary condition for the occurrence of an event. Eq. 3.5 is an example of an invalid description for the event of rock erosion shown in Fig. 3.7. The same SNAP entity (i.e., $mass(\text{rock}_1)$) has been recorded three times, so we cannot distribute the predicates in Eq. 3.5 into the precondition and postcondition.

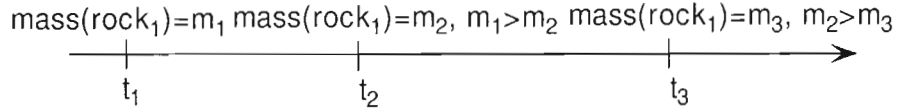


Figure 3.7: A rock erosion event

$$(3.5) \quad \begin{aligned} & holds(\text{mass}(\text{rock}_1) = m_1, t_1) \wedge holds(\text{mass}(\text{rock}_1) = m_2, t_2) \\ & \wedge holds(\text{mass}(\text{rock}_1) = m_3, t_3) \end{aligned}$$

3.2.2 Time

To deal with complicated events such as simultaneous events and overlapping events, interval logic is employed in the representation framework. Events occur and states are valid within time intervals. Time intervals are in the form of $[a, b]$, where a and b are endpoints of time intervals in some units of measurement. For events occurring at time points, we use instantaneous intervals (zero duration time intervals) to represent event occurrence times. Given a time interval ι , we use $start(\iota)$ and $end(\iota)$ to access the endpoints of ι . After shifting by means of a time interval ι_{offset} , we get a new interval

$$(3.6) \quad shift(\iota, \iota_{offset}) = [start(\iota) + start(\iota_{offset}), end(\iota) + start(\iota_{offset})]$$

We incorporate two kinds of time in our model, *local time* and *global time*. Local time t is encoded in the precondition and postcondition of the event through the truth predicate $holds(s, t)$. Global time τ is used to specify the time interval when the event actually occurs. Occurrences of event E_1 are specified using a predicate

occurs. For example, $e_{11} = occurs(E_1, \tau_1)$ indicates an occurrence of E_1 during τ_1 . Conversely, two predicates, *type* and *time*, are used to get the event and time interval of the occurrence.

$$E_1 = type(e_{11})$$

$$\tau_1 = time(e_{11})$$

Using these three predicates, we build a connection between events and their occurrences. An event may occur several times in the same space, but at the same time, only one occurrence of the event exists.

$$(3.7) \quad \forall i, j \quad type(e_i) = type(e_j) \wedge time(e_i) = time(e_j) \rightarrow e_i = e_j$$

Definition 3.4 (*event occurrence*). Given an event E , its occurrence e during time τ_1 is defined as follows:

$$(3.8) \quad e = occurs(E, \tau_1) \equiv_{def} \langle PRE_E(\tau_1), POST_E(\tau_1) \rangle$$

in which

$$(3.9) \quad PRE_E(\tau_1) = holds(s_1, shift(t_1, \tau_1)) \wedge \dots \wedge holds(s_{n+m}, shift(t_{n+m}, \tau_1))$$

$$(3.10) \quad POST_E(\tau_1) = holds(s'_1, shift(t'_1, \tau_1)) \wedge \dots \wedge holds(s'_n, shift(t'_n, \tau_1))$$

Using Eq. 3.6, we can derive the real situation for the event occurrence (e.g., e_{11}) after the local time of the event (e.g., E) is shifted by means of the occurrence time of e_{11} (e.g., τ_1). Fig. 3.8 shows the result of shifting the local time t_1 and t_2 by means of the global time τ_1 .

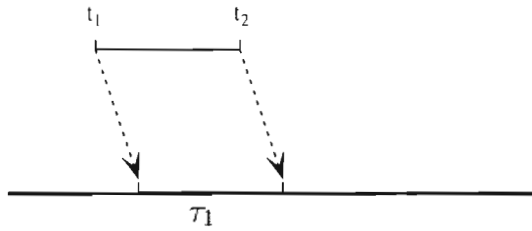


Figure 3.8: Deriving the real situation for event occurrences

3.3 Event Hierarchy

People arrange information hierarchically and use hierarchies to reason such complex systems. A hierarchy organizes information at different levels of detail, and it is conceptually constructed through abstraction, a process of eliminating the irrelevant and amplifying the essential.

Let \mathcal{E} be the event space, a set of events we are concerned with according to the application domain. An event hierarchy is proposed to hierarchically structure the event space. As we see in Def. 1.1, the event hierarchy defines a mapping from more specialized events (low-level events) to more general events (higher-level events) through a partial order relation. This event hierarchy is depicted by a Hasse diagram or a directed acyclic graph (DAG) with nodes representing events in \mathcal{E} and edges representing partial order between events at different levels of detail. Fig. 3.9 and 3.10 gives two event hierarchies.

Hierarchies are distinguished by means of how they are constructed. Two abstraction processes are studied to construct the event hierarchy: generalization and aggregation. Generalization describes the *is-a* relation between events, which leads to the *event taxonomy*. Aggregation is used to study and explain the compositional

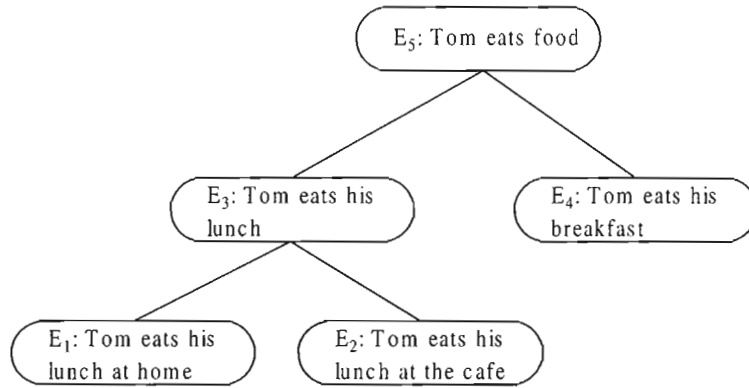


Figure 3.9: An event taxonomy

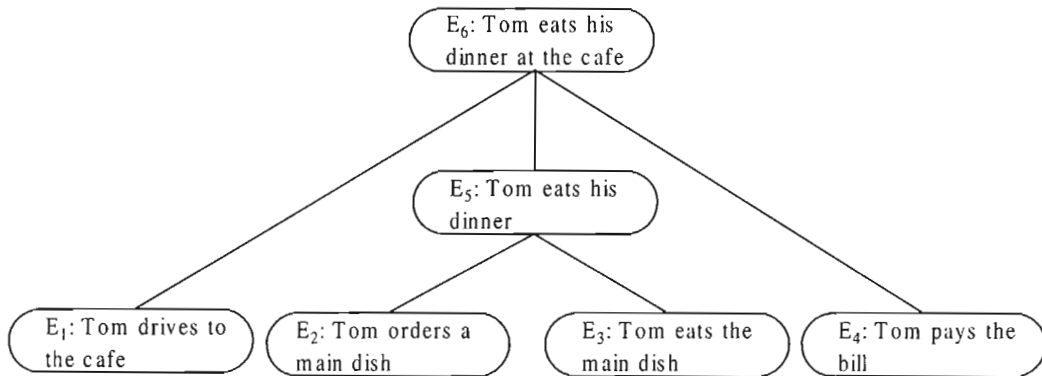


Figure 3.10: An event partonomy

structure of complex events in the system, and it is important to understand both the individual events and the relationships among them. The relationship established by aggregation is a *part-of* relationship that holds between two events when one is a part of the other. This leads to the *event partonomy*. Fig. 3.10 shows an event partonomy, and Fig. 3.9 is an event taxonomy.

3.3.1 Event Taxonomy

In the event taxonomy, *super-event* and *sub-event* refer to more general and more specialized events, respectively. The event taxonomy establishes the *is-a* relationship between events. In the event taxonomy shown in Fig. 3.9, $is-a(E_1, E_3)$ specifies the relationship between event E_1 and event E_3 . E_3 is the super-event, and E_1 is the sub-event. By extracting and sharing common parts of events' preconditions and postconditions, we can generalize events and place them higher up in the event taxonomy. Conversely, we can specialize an event by adding something that is unique to the event to its precondition and postcondition.

Events can be specialized to sub-events by adding more *holds* predicates to their preconditions and postconditions. For an event E , we can add additional *holds* predicates to the precondition of E , indicating more contextual information for E , or we can add additional *holds* predicates to both of its precondition and postcondition to indicate new and more detailed changes we find in the event. Although we can add *holds* predicates to its postcondition, the same SNAP entities recorded by the *holds* predicates must already be recorded in its precondition, so as to not violate the definition 3.2. For example, it is wrong to use Eq. 3.11 to represent the event E

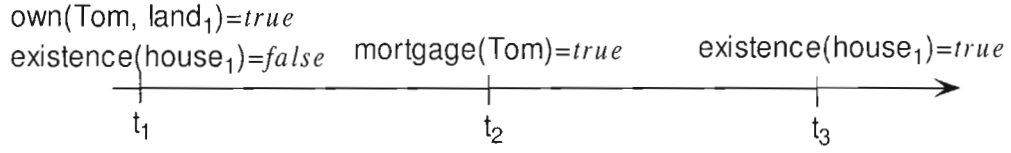


Figure 3.11: A refined representation of the event in Fig. 3.6

shown in Fig. 3.11, which is more specialized than that in Fig. 3.6.

$$\begin{aligned} < \text{holds}(\text{existence}(\text{house}_1) = \text{false}, t_1) \wedge \text{holds}(\text{own}(\text{Tom}, \text{land}_1) = \text{true}, t_1), \\ & \text{holds}(\text{existence}(\text{house}_1) = \text{true}, t_3) \wedge \text{holds}(\text{mortgage}(\text{Tom}) = \text{true}, t_2) > \end{aligned}$$

We do not have the observation of the SNAP entity (i.e., $\text{mortgage}(\text{Tom})$) in the precondition. So the additional proposition $\text{holds}(\text{mortgage}(\text{Tom}) = \text{true}, t_2)$ belongs to the precondition rather than the postcondition, indicating the contextual information of occurrences of E . The correct form for E is

$$\begin{aligned} < \text{holds}(\text{existence}(\text{house}_1) = \text{false}, t_1) \\ & \wedge \text{holds}(\text{own}(\text{Tom}, \text{land}_1) = \text{true}, t_1) \wedge \text{holds}(\text{mortgage}(\text{Tom}) = \text{true}, t_2), \\ & \text{holds}(\text{existence}(\text{house}_1) = \text{true}, t_3) > \end{aligned}$$

An occurrence of an event (e.g., E) during t is also an occurrence of its super-event (e.g., E'). That is, if E occurs during t , E' also occurs during t .

$$(3.11) \quad \text{is-a}(E', E) \wedge \text{occurs}(E', t) \rightarrow \text{occurs}(E, t)$$

Meyer (1988) observes that specialization can be regarded as both an extension and a restriction. In addition to adding more propositions into the precondition and postcondition, we can specialize events by replacing the objects involved with their sub-objects. From this viewpoint, occurrences of the sub-event represent a

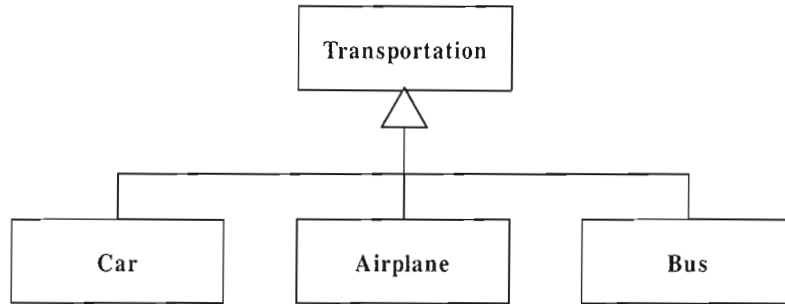


Figure 3.12: An object hierarchy

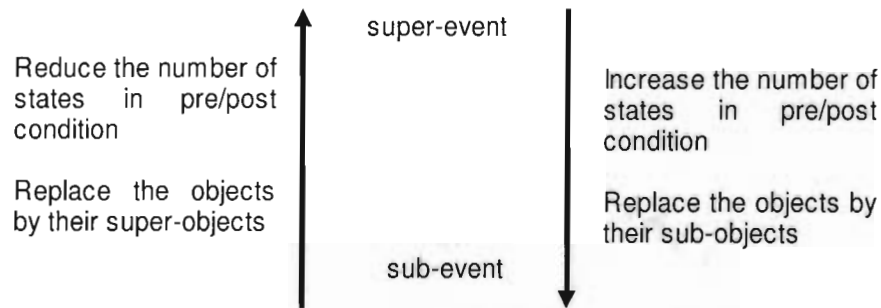


Figure 3.13: Generalization and specialization

sub-collection of all occurrences of its super-event. Replacing the objects with their sub-objects requires an object type hierarchy. For example (Fig. 3.12), we can replace the Transportation vehicle objects with Car objects in the event definition to specialize the event. Conversely, we can also replace objects with their super-objects to generalize the event. Fig. 3.13 shows the common generalization and specialization process.

3.3.2 Event Partonomy

By aggregating several associated events, we structure a *composite event* using its parts. This process is called *aggregation*, in which the events to be aggregated are called *component events*. People are sensitive to the event part structure at differ-

ent time scales, ranging from events in terms of physical change, events defined in relationship to intentional act, events characterized by plots, to events characterized thematically (Zacks and Tversky 2001). The part structure is organized into a hierarchy, called the *event partonomy*. For example, E_2 and E_3 are aggregated into E_5 in Fig. 3.10. In the event partonomy, an *atomic event* is the smallest recognizable event, which makes up the bottom level of the partonomy.

Being a constructive activity, aggregation refers to the assembly of interacting component events, which suppresses the detail of the component events. For each aggregation, instead of specifying the precondition and postcondition for the composite event, we specify the interacting component events and their temporal order through $ESet$, defined by

$$(3.12) \quad ESet = \{reloc(E_1, t_1), \dots, reloc(E_n, t_n)\}$$

in which t_i is used to denote the position of component events in the composite event, and it belongs to local time. For atomic events, $ESet$ is null. Fig. 3.14 shows the structure for the event partonomy in Fig. 3.10.

The operator *reloc* is defined as follows

$$(3.13) \quad reloc(E, t) = \langle PRE_E(t), POST_E(t) \rangle$$

$PRE_E(t)$ and $POST_E(t)$ in Eq. 3.13 are defined by

$$(3.14) \quad PRE_E(t) = holds(s_1, shift(t_1, t)) \wedge \dots \wedge holds(s_n, shift(t_n, t))$$

$$(3.15) \quad POST_E(t) = holds(s'_1, shift(t'_1, t)) \wedge \dots \wedge holds(s'_m, shift(t'_m, t))$$

In Eqs. 3.14 and 3.15, E is the event defined in Eq. 3.1. Note the similarity with Eqs. 3.9 and 3.10, but in this case, the shifts are still within local time.

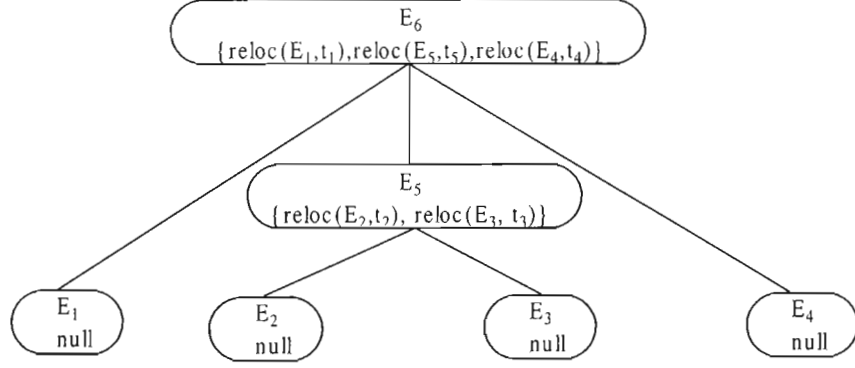


Figure 3.14: The event partitioning structure

In the event partitioning, conjunction is used to model composite events; that is, every component event should occur to compose the composite event. To aggregate events E_1, \dots, E_n , we construct the $ESet$ of the composite event and get the conjunction of all elements in $ESet$, represented by $reloc(E_1, t_1) \wedge \dots \wedge reloc(E_n, t_n)$. For example, E_2 and E_3 are aggregated into E_5 . The representation of E_5 is given by $reloc(E_2, t_2) \wedge reloc(E_3, t_3)$. However, it is not equal to $\langle PRE_{E_2}(t_2) \wedge PRE_{E_3}(t_3), POST_{E_2}(t_2) \wedge POST_{E_3}(t_3) \rangle$, and we will explain this in Sec. 4.3.2.

The relationship between component events and composite events is *part-of*. For example, $part-of(E_2, E_5)$ indicates E_2 is a component event of E_5 . Given a component event (e.g., E_2) of a composite event (E_5), if the composite event (i.e., E_5) occurs, the component event (i.e., E_2) occurs definitely. But it is not the case in reverse.

$$(3.16) \quad occurs(E, t) \wedge part-of(E', E) \rightarrow \exists t' \subset t \wedge occurs(E', t')$$

3.4 Summary

This chapter discusses an event-oriented model, in which we define the event and its occurrences through the concepts of precondition and postcondition plus local time and global time. In addition, the event taxonomy and paratomy hierarchies are introduced to provide us with a mechanism to support representations of events at different levels of detail.

Chapter 4

ALGORITHM FOR CONSTRUCTING THE EVENT PARTONOMY

In Chapter 3, we discussed two event hierarchies, the event taxonomy and the event partonomy. In this chapter, we focus on the algorithm for constructing the event partonomy. In the event partonomy, events are aggregated into composite events at coarser levels of detail, and the composite events are characterized by their internal structure, that is, the nature and inter-relationships of their components. Events may be aggregated in different ways, and this work attempts to aggregate events in ways that accord with user perception. For this, we need to take account of event-event relations.

An event is an entity with both temporal and spatial extents. However, spatial and temporal relations between events are not the main factors for aggregating component events (Sec. 2.4). The causal relations between events are key relations for aggregating events, and interactions between preconditions and postconditions help us to investigate the causal relations. In order to approximate to the causal relations, this chapter describes two constructs, *sequence* and *transition*. It is these constructs that allow us to automatically construct the event partonomy, taking account of the context in which component events occur. This chapter concludes by presenting the algorithm for deriving the representation of composite events.

4.1 Event-Event Relations

The combination of selected events describes the change to the spatial and aspatial properties over time. Consider the scenario in Fig. 4.1(a), ball₁ moves ahead from point₁, through point₂, to point₃. We use two events to describe the continuous change to the position of ball₁, i.e., E_1 (ball₁ moves from point₁ to point₂) and E_2 (ball₁ moves from point₂ to point₃). These events can be composed to form a more general event. However, there are time intervals during which change to one property is transferred to other properties. This is critical for people to perceive the event partonomic structure. Suppose, in Fig. 4.1(b), we have added ball₂, stationary at point₂. Now, ball₁ hits ball₂, and ball₁ stops at point₃ while ball₂ moves from point₂, through point₃, to point₄. We have events E_3 (ball₁ hits ball₂), E_4 (ball₂ moves from point₂ to point₃), and E_5 (ball₂ moves from point₃ to point₄). The change to ball₁'s position during the occurrence of E_1 is transferred to the change to ball₂'s movement during the occurrence of E_3 , which in turn is transferred to the change to ball₂'s position during the occurrence of E_4 . These transfers correspond to the maximum number of properties that are changing, indicating a “contour discontinuity” in the temporally extended events. People tend to mark the boundaries of the events when there is a contour discontinuity. Conversely, people will tend to aggregate those events between which transfers occur.

We define two event relations, *f-sequence relation* and *f-transition relation*, where f is a fluent in the events' pre- and/or postconditions. Event occurrences are related if they belong to the same *f-sequence*, or if there is a *f-transition* between them.

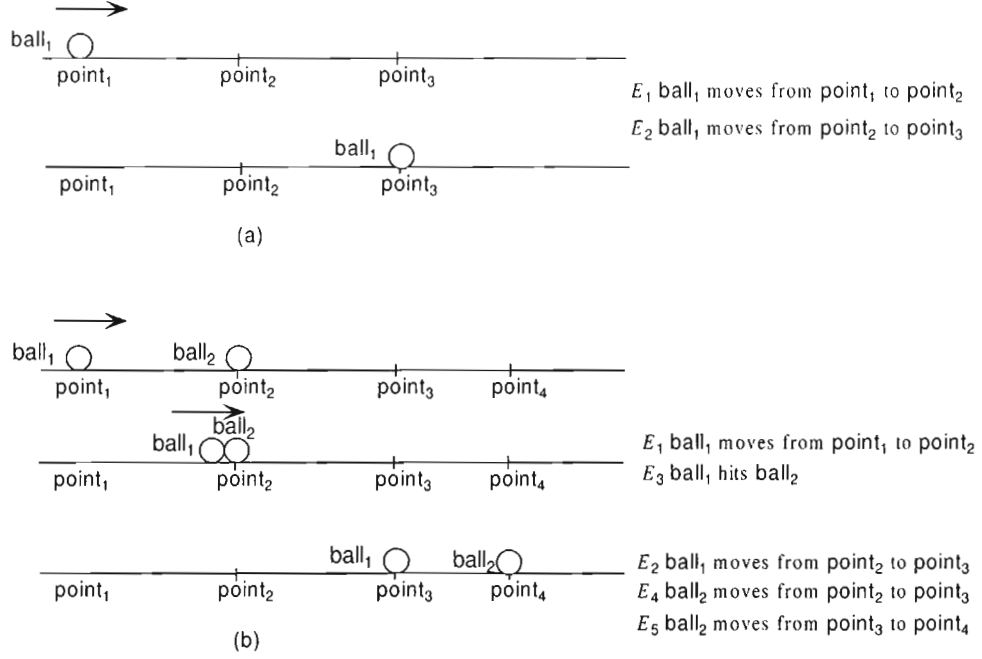


Figure 4.1: Ball scenarios

To facilitate the following discussion, we define four operators to access fluents in the precondition and postcondition.

$$(4.1) \quad FSET_{pre} : \mathcal{E} \rightarrow 2^F$$

$$(4.2) \quad FSET_{post} : \mathcal{E} \rightarrow 2^F$$

$$(4.3) \quad FVAL_{pre} : \mathcal{E} \times F \rightarrow Domain_F \cup null$$

$$(4.4) \quad FVAL_{post} : \mathcal{E} \times F \rightarrow Domain_F \cup null$$

\mathcal{E} denotes the event space, F denotes the set of fluents, and $Domain_F$ is the value domains for the fluents in F . $FSET_{pre}(E)$ returns the set of fluents in the precondition of E , and $FSET_{post}(E)$ returns the set of fluents in the postcondition of E . $FVAL_{pre}(E, f)$ retrieves the values of fluent f in the precondition of E , and

$FVAL_{post}(E, f)$ retrieves the values of fluent f in the postcondition of E . In more formal terms

$$(4.5) \quad FSET_{pre}(E) = \{f \in F \mid f \text{ is in precondition of } E\}$$

$$(4.6) \quad FSET_{post}(E) = \{f \in F \mid f \text{ is in postcondition of } E\}$$

$$(4.7) \quad FVAL_{pre}(E, f) = \begin{cases} \text{value (value} \in \text{Domain}_f), & \text{if } f \in FSET_{pre}(E) \\ \text{null,} & \text{otherwise.} \end{cases}$$

$$(4.8) \quad FVAL_{post}(E, f) = \begin{cases} \text{value (value} \in \text{Domain}_f), & \text{if } f \in FSET_{post}(E) \\ \text{null,} & \text{otherwise.} \end{cases}$$

Given an occurrence e , $FTIME_{pre}(e, f)$ and $FTIME_{post}(e, f)$ are defined to return the time intervals during which we record the values of fluent f in e 's precondition and postcondition. They return *null* if f does not appear in e 's precondition and postcondition. Formally we have

$$(4.9) \quad FTIME_{pre} : \mathcal{O} \times F \rightarrow T \cup \text{null}$$

$$(4.10) \quad FTIME_{post} : \mathcal{O} \times F \rightarrow T \cup \text{null}$$

where \mathcal{O} is the set of event occurrences. During the time interval between $FTIME_{pre}(e, f)$ and $FTIME_{post}(e, f)$, e makes some change to f .

Definition 4.1 (*f*-influence time). Given an event occurrence e and a fluent f , the *f*-influence time of e is defined to be the interval during which e makes changes to f , which is formally represented by $[end(FTIME_{pre}(e, f)), start(FTIME_{post}(e, f))]$, where $start(t)$ and $end(t)$ give the starting point and the end point of the time interval t .

Events can occur concurrently, but they do not make changes to the same fluent concurrently. That is, their occurrences do not have overlapping f -influence times, and so for each f , f -influence times can be linearly ordered.

4.1.1 Sequence

Definition 4.2 (*f*-sequence relation). Given two events E_i and E_j and a fluent f , there is a *f*-sequence relation between E_i and E_j if

(i) each event has f in both its precondition and postcondition, that is,

$$f \in FSET_{pre}(E_i) \cap FSET_{post}(E_i) \cap FSET_{pre}(E_j) \cap FSET_{post}(E_j)$$

(ii) $FVAL_{post}(E_i, f) = FVAL_{pre}(E_j, f)$

The *f*-sequence relation between E_i and E_j is denoted by *f*-sequence(E_i, E_j). Based on the definition of *f*-sequence relation, we can create sequences for different fluents.

Definition 4.3 (*f*-sequence). A list of occurrences $[e_1, \dots, e_n]$ is defined as an *f*-sequence if *f*-sequence($type(e_i), type(e_{i+1})$) for $i \in \{1, \dots, n - 1\}$, and occurrences are ordered by their *f*-influence times.

Note that each sequence is associated with one fluent and gives a description of that fluent by recording its different values at the beginning and the end of the event occurrences. So a sequence describes changes to a SNAP entity.

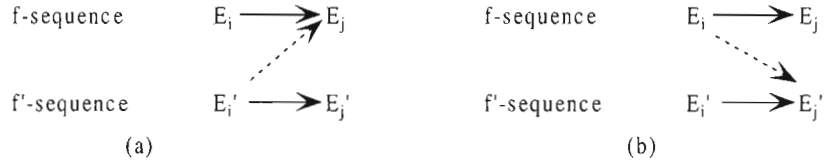


Figure 4.2: Transitions between f -sequence and f' -sequence

For example, in Fig. 4.1(b) the occurrences e_1 of E_1 and e_2 of E_2 belong to a sequence since there is a $\text{pos}(\text{ball}_1)$ -sequence relation between E_1 and E_2 , and the $\text{pos}(\text{ball}_1)$ -influence time of e_1 is before that of e_2 . The occurrences of E_4 (e.g., e_4) and E_5 (e.g., e_5) belong to a $\text{pos}(\text{ball}_2)$ -sequence. Only the occurrence of E_3 (e.g., e_3) belongs to a $\text{mov}(\text{ball}_2)$ -sequence. Therefore, there are three sequences: the $\text{pos}(\text{ball}_1)$ -sequence describes changes to the position of ball_1 , the $\text{pos}(\text{ball}_2)$ -sequence describes changes to the position of ball_2 , and the $\text{mov}(\text{ball}_2)$ -sequence describes whether ball_2 is static.

$\text{pos}(\text{ball}_1)$ -sequence	$[e_1, e_2]$
$\text{pos}(\text{ball}_2)$ -sequence	$[e_4, e_5]$
$\text{mov}(\text{ball}_2)$ -sequence	$[e_3]$

An occurrence may be associated with more than one sequence according to the number of fluents that appear in both its precondition and postcondition.

4.1.2 Transition

Events are not only related through the f -sequence relation. For example, although events E_4 and E_5 in Fig. 4.1(b) are related through a $\text{pos}(\text{ball}_2)$ -sequence relation, for E_4 to occur, ball_2 should be moving. The fluent $\text{mov}(\text{ball}_2)$ in the precondition

of E_4 becomes true after the occurrence of E_3 . This requires a connection between events E_3 and E_4 , and this is formalized using the f -transition relation.

Definition 4.4 (f -transition relation). Given two event E_i and E'_j and a fluent f , there is a f -transition relation between E_i and E'_j if

$$(i) \quad f \in FSET_{pre}(E'_j), f \notin FSET_{post}(E'_j), f \in FSET_{post}(E_i), \text{ and } FVAL_{post}(E_i, f) = FVAL_{pre}(E'_j, f)$$

$$(ii) \quad FSET_{pre}(E_i) \cap FSET_{post}(E_i) \cap FSET_{pre}(E'_j) \cap FSET_{post}(E'_j) = \phi$$

The f -transition relation between E_i and E'_j is denoted by f -transition(E_i, E'_j).

Definition 4.5 (f -transition). Given two events E_i and E'_j , there is a f -transition from the occurrence e_i of E_i to the occurrence e'_j of E'_j (Fig. 4.2(b)), if

$$(i) \quad f\text{-transition}(E_i, E'_j)$$

$$(ii) \quad e_i \text{ is the most recent occurrence in the } f\text{-sequence before } e'_j$$

For example, there is a pos(ball₁)-transition between occurrences of E_1 and E_3 , and a mov(ball₂)-transition between occurrences of E_3 and E_4 (Fig. 4.3). Transitions are represented using dotted arrows and sequences are represented using continuous arrows in Fig. 4.3. For simplicity, we do not give the occurrence times in the sequence.

4.2 Constructing the Event Partonomy

We are now ready to give the algorithm for constructing the event partonomy. The algorithm first scans the event space, checking out the sequence and transition relations between events. Based on these relations, it will generate sequences for different

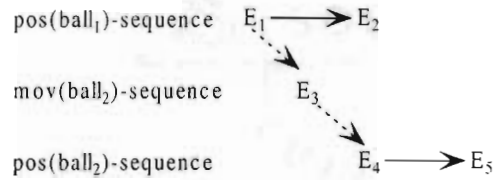


Figure 4.3: Transitions in the ball scenario

fluents and transitions between event occurrences. The general principles by which the algorithm works are as follows:

- (i) If occurrences are situated in consecutive positions in the same sequence, and are not involved in any transitions, they are viewed as a single occurrence of their composite event.
- (ii) To eliminate the transitions from the occurrence of E to the occurrence of E' , the algorithm aggregates E with events from the sequence to which the occurrence of E' belongs.
- (iii) The algorithm dynamically checks the occurrences in every sequence to see if events can be further aggregated.
- (iv) This process is repeated until there is no transition from the occurrence of E to other occurrences.

Fig. 4.4 gives the overall actual sequence for the algorithm. The input of the algorithm is a list of event occurrences, based on which the algorithm constructs an event partonomy.

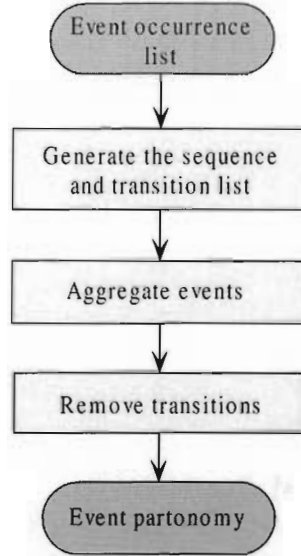


Figure 4.4: Process of constructing the partonomy

4.2.1 Generating Sequences and Transitions

Algorithm 4.1 gives the process of generating sequences and transitions. According to definitions 4.2 and 4.3, event occurrences in the f -sequence should have the fluent f in their preconditions and postconditions, and they are ordered by f -influence times. Each occurrence e_i and its succeeding occurrence e_{i+1} should meet the requirement $FVAL_{post}(type(e_i), f) = FVAL_{pre}(type(e_{i+1}), f)$.

If there is a transition from the occurrence of E_1 (e.g., e_1) in the f -sequence to the occurrence of E_2 (e.g., e_2), e_1 and e_2 cannot be in the same sequence. In addition, e_1 is the most recent occurrence in the f -sequence before e_2 , and $FVAL_{post}(E_1, f) = FVAL_{pre}(E_2, f)$. There are f -transitions from e_1 to occurrences that only appear in different sequences from e_1 does, and should occur later than e_1 .

The transitions between occurrences constitute a graph, called a *transition graph*, in which the *outdegree* of an vertex (occurrence) gives the number of transitions from it, and its *indegree* gives the number of transitions to it.

4.2.2 Aggregating Events

During the second step, the algorithm searches for the first occurrence in an f -sequence with no f -transition. If there is no such occurrence, it turns to another sequence. Otherwise it aggregates events according to Rule I:

Rule I: Given an occurrence e_1 in an f -sequence with no f -transition from it, let e_2 be the first of e_1 's succeeding occurrences that is involved in transitions, either from it or to it. The events occurring between e_1 and e_2 including $type(e_1)$ will be aggregated if there is an f' -transition from other occurrences to e_2 , otherwise, the events occurring between e_1 and e_2 including $type(e_1)$ and $type(e_2)$ are aggregated if there is an f -transition from e_2 .

The algorithm repeats the above process on the remaining occurrences of the f -sequence until there is no occurrence with no f -transition from it.

After aggregating the events, the occurrences of the component events are replaced by the occurrence of the composite event, and there are transitions from the occurrence of the composite event to other occurrences, e say, if there are transitions from the occurrence of any of its component events to e .

Algorithm 4.1: Generating sequences and transitions

Input: a list of occurrences $eList$
Output: sequences and transitions

- 1: $ESet \leftarrow \phi$; \textit{*create an event set*\}
- 2: **for** each occurrence e in $eList$ **do**
- 3: $E \leftarrow type(e)$
- 4: add E to $ESet$
- 5: **end for**
- 6: $fSeqSet \leftarrow \phi$; $fTraSet \leftarrow \phi$; \textit{*create a sequence fluent set and a transition fluent set*\}
- 7: **for** each E in $ESet$ **do**
- 8: add $FSET_{pre}(E) \cap FSET_{post}(E)$ to $fSeqSet$;
- 9: add $FSET_{pre}(E) - FSET_{post}(E)$ to $fTraSet$;
- 10: **end for**
- 11: **for** each f in $fSeqSet$ **do**
- 12: $f\text{-sequence} \leftarrow \phi$;
- 13: **for** each E in the $ESet$ **do**
- 14: **if** $f \in FSET_{pre}(E) \cap FSET_{post}(E)$ **then**
- 15: add occurrences of E in $eList$ to $f\text{-sequence}$ and sort them according to their $f\text{-influence}$ times
- 16: **end if**
- 17: **end for**
- 18: **end for**
- 19: **for** each f in $fTraSet$ **do**
- 20: $canOccList \leftarrow \phi$;
- 21: $seqSetE \leftarrow FSET_{pre}(E) \cap FSET_{post}(E)$;
- 22: **for** each E' in $ESet$ **do**
- 23: $seqSetE' \leftarrow FSET_{pre}(E') \cap FSET_{post}(E')$;
- 24: **if** $f \in seqSetE'$ and $seqSetE \cap seqSetE' = \phi$ and $FSET_{post}(E', f) = FSET_{pre}(E, f)$ **then**
- 25: add occurrences of E' in $eList$ to $canOccList$;
- 26: **end if**
- 27: **end for**
- 28: $f\text{-transition} \leftarrow \phi$;
- 29: **for** each E in $ESet$ **do**
- 30: **if** $f \in FSET_{pre}(E) - FSET_{post}(E)$ **then** \textit{*occurrences of E need transitions to them*\}
- 31: **for** each occurrence e of E in $eList$ **do**
- 32: $t \leftarrow time(e)$;
- 33: get the occurrence e' which is the most recent occurrence to e in $canOccList$ and add (e', e) to $f\text{-transition}$;
- 34: **end for**
- 35: **end if**
- 36: **end for**
- 37: **end for**
- 38: return $f\text{-sequences}$ and $f\text{-transitions}$

4.2.3 Removing Transitions

Each occurrence in the sequences is associated with transitions as a result of Sec. 4.2.2. Through removing transitions between occurrences, the algorithm aggregates events and constructs the event partition. The algorithm for removing transitions is described in algorithm 4.2, and the basic idea for the algorithm is to make occurrences appear in the same sequence. For example, to remove the transition from the occurrence of E_i to the occurrence of E'_j in Fig. 4.2(b), the algorithm aggregates E_i with E'_j and replaces their occurrences with the occurrence of the composite event. There is no transition between the occurrence of the composite event in the f -sequence and the occurrence of E'_j since these two occurrences now appear in the same sequence (i.e., f' -sequence). If the occurrence of E'_j is the first occurrence in the f' -sequence, the algorithm simply places the occurrence of E_i before the occurrence of E'_j . After this, the algorithm scans the sequences to see if events can be further aggregated according to Rule I. The process of removing transitions from the occurrence of E_i will be repeated until there is no transition from the occurrence of E_i , and thereby we can apply Rule I to the occurrence of E_i and its succeeding occurrences in the f -sequence for aggregating events.

Algorithm 4.2: Removing transitions

Input: sequences and transitions as a result of section 4.2.2
Output: a list of composite events

- 1: **for** each f -sequence **do**
- 2: **for** each occurrence e in the f -sequence **do**
- 3: $E \leftarrow \text{type}(e); t \leftarrow \text{time}(e); \backslash * \text{get its event and occurrence time} * \backslash$
- 4: get a set S of occurrences to which there is a f -transition from e
- 5: **for** each occurrence e' in S **do**
- 6: **for** each f -sequence to which e' belongs **do**
- 7: locate the e' in the f -sequence, and its previous occurrence e'_p
- 8: $\tau_1 \leftarrow \text{time}(e); \tau_2 \leftarrow \text{time}(e'_p); \backslash * \text{get their occurrence times} * \backslash$
- 9: $t \leftarrow \min(\tau_1, \tau_2); t_1 \leftarrow \tau_1 - t; t_2 \leftarrow \tau_2 - t \backslash * \text{get their local times in the composite event} * \backslash$
- 10: $E_1 \leftarrow E; E_2 \leftarrow \text{type}(e'_p); \backslash * \text{get their event types} * \backslash$
- 11: compose E_1, E_2 according to their local times t_1, t_2
- 12: update the sequences and transitions with the composite event
- 13: **if** indegree of $e' = 0$ **then** $\backslash * \text{no transition to } e' * \backslash$
- 14: aggregate $\text{type}(e')$ and events occurring before e' in the f -sequence
- 15: update the sequences and transitions with the composite event
- 16: **end if**
- 17: **end for**
- 18: remove the f -transition(e, e');
- 19: **end for**
- 20: **if** outdegree of $e = 0$ **then** $\backslash * \text{no transition from } e * \backslash$
- 21: aggregate E and events occurring after e in the f -sequence
- 22: update the sequences and transitions with the composite event
- 23: **end if**
- 24: **end for**
- 25: **end for**

4.2.4 Demonstration

To construct the event partition for Fig. 4.1(b), the algorithm creates the sequences and transitions as Fig. 4.3 shows. According to Rule I, E_4 and E_5 are aggregated into a composite event, denoted by E_{45} . The transition between occurrences of E_1 and E_3 is removed by placing the occurrence of E_1 before the occurrence of E_3 in $\text{mov}(\text{ball}_2)$ -sequence (Fig. 4.5(I)). Since there is no transition from E_1 , the algorithm

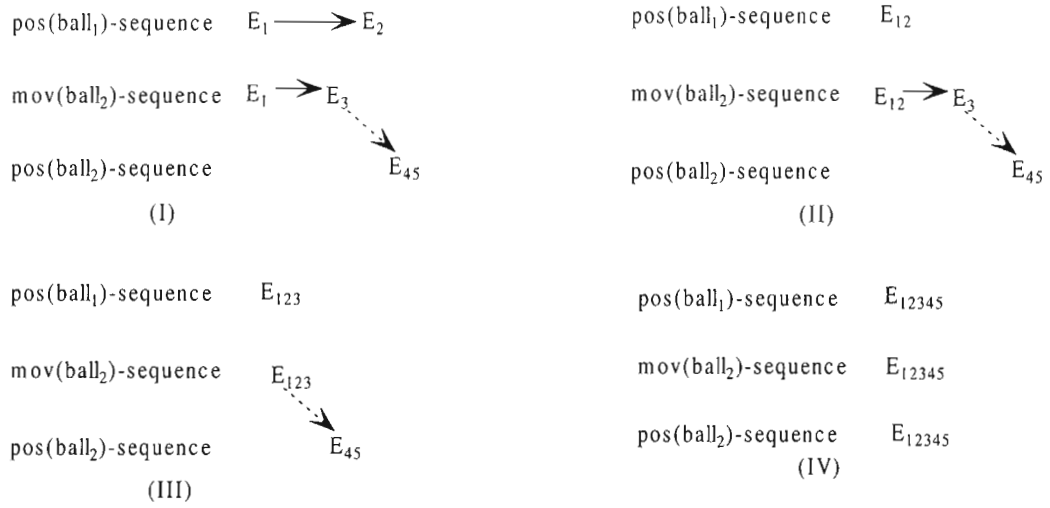


Figure 4.5: Different stages during constructing the partonomy
(I) after removing pos(ball₁)-transition(E_1, E_3)
(II) after aggregating events E_1 and E_2
(III) after aggregating events E_{12} and E_3
(IV) after removing all transitions

aggregates E_1 and E_2 into event E_{12} according to Rule I, and replaces the occurrences of E_1 and E_2 with the occurrence of E_{12} (Fig. 4.5(II)). On the other hand, there is no $\text{mov}(\text{ball}_2)$ -transition from the occurrence of E_{12} , so E_{12} is aggregated with E_3 in the $\text{mov}(\text{ball}_2)$ -sequence into the composite event E_{123} (Fig. 4.5(III)). Repeating this process systematically for each event results in the event partonomy. After removing all transitions, the sequences for Fig. 4.1(b) are shown in Fig. 4.5(IV), and the algorithm generates the event partonomy for Fig. 4.1(b) as Fig. 4.6 shows.

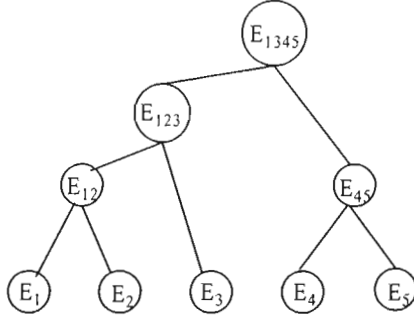


Figure 4.6: The event partition for the ball scenario

4.3 Representations for Composite Events

The composite events are represented by their preconditions and postconditions, and their representations can be derived through the conjunction of all elements in the $ESet$ (Sec. 3.3.2). However, they are not that simply the Boolean “and” and “or” of the component events. Given two events E_i and E_j

$$E_i = \langle PRE_{E_i}, POST_{E_i} \rangle$$

$$E_j = \langle PRE_{E_j}, POST_{E_j} \rangle$$

we specify their temporal order through $reloc(E_i, t_i)$ and $reloc(E_j, t_j)$. According to formulas 3.12 and 3.13, the representation for the composite event is

$$\langle PRE_{E_i}(t_i), POST_{E_i}(t_i) \rangle \wedge \langle PRE_{E_j}(t_j), POST_{E_j}(t_j) \rangle$$

It is different from the conjunctions of the component events’ preconditions and postconditions

$$(4.11) \quad PRE_{E_i}(t_i) \wedge PRE_{E_j}(t_j)$$

$$(4.12) \quad POST_{E_i}(t_i) \wedge POST_{E_j}(t_j)$$

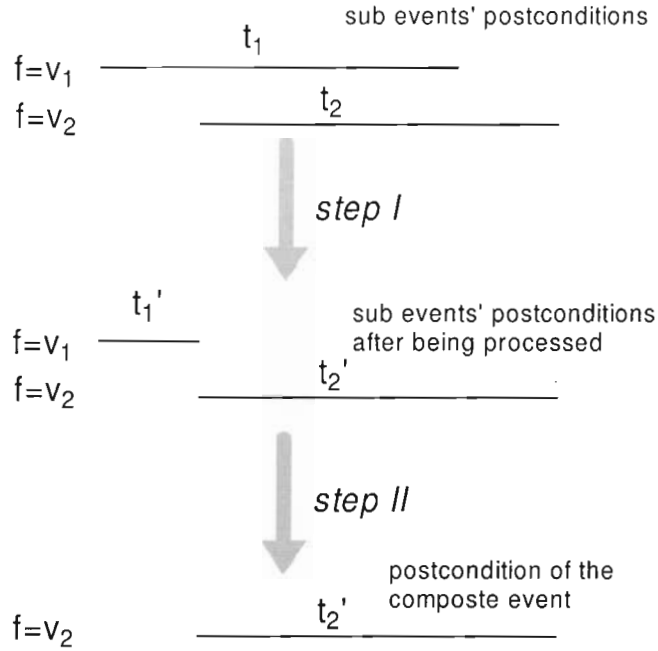


Figure 4.7: Process of getting the postcondition for a composite event

which inevitably contradicts the constraints on the definition of the event: each fluent exists only once in the event's precondition and postcondition. The next section describes the algorithm for constructing this composition.

4.3.1 Algorithm

The construction of the precondition and postcondition for the composite event, as algorithm 4.3 shows, begins with the assembly of the component events' preconditions and postconditions. According to the component events' positions in the composite event, we relocate the local time intervals for *holds* predicates in the component events. As a result, we get two formulas 4.11 and 4.12. They need to be modified to give the precondition and postcondition for representing the composite event.

In the postcondition, the values of a fluent are recorded as a result of making changes to the fluent, so we have disjoint times intervals to record the values of the fluent. If a fluent has been recorded more than once, we only need to keep the most updated one. For example, the conjunctive form of postcondition, $holds(f = v_1, t_1) \wedge holds(f = v_2, t_2)$, will be changed to $holds(f = v_2, t'_2)$ (Fig. 4.7). First, time intervals t_1 and t_2 need to be modified into disjoint intervals t'_1 and t'_2 , and we get $tPOST = holds(f = v_1, t'_1) \wedge holds(f = v_2, t'_2)$. During the second step, the obsolete recordings of the fluent are eliminated.

Processing the conjunctions of the component events' preconditions consists of two parts. First, the proposition $holds(s, t)$ in the conjunction of preconditions will be removed if it is implied by $tPOST$. Through this, we suppress the detailed information about the component events. If the remaining propositions still share a common fluent, we will say that the events cannot be aggregated.

4.3.2 Examples

This section demonstrates the algorithm for deriving the preconditions and postconditions of the composite events created for Fig. 4.1(b).

The first example is the aggregation of E_1 and E_2 , which are represented by

$$E_1 = \langle holds(\text{pos}(\text{ball}_1) = \text{point}_1, t_{11}), holds(\text{pos}(\text{ball}_1) = \text{point}_2, t_{12}) \rangle$$

$$E_2 = \langle holds(\text{pos}(\text{ball}_1) = \text{point}_2, t_{21}), holds(\text{pos}(\text{ball}_1) = \text{point}_3, t_{22}) \rangle$$

where t_{11} , t_{12} , t_{21} , and t_{22} are shown in Fig. 4.8(a). To compose the composite event E_{12} , we relocate E_1 with a zero offset and E_2 with an offset of the duration of E_1 (Fig. 4.8(b)), and get conjunctions of their preconditions and postconditions

Algorithm 4.3 Deriving the representations of the composite events

Input: component events E_i and their time intervals t_i relevant to the composite event E ($i = 1, \dots, n$)

Output: precondition PRE_E , and postcondition $POST_E$ of the composite event E

- 1: $PRE_E \leftarrow \phi$; $POST_E \leftarrow \phi$; **initialize the result**
- 2: **for** $i = 0$ to n **do**
- 3: insert(PRE_E , pre(E_i, t_i)); insert($POST_E$, post(E_i, t_i)); **assemble the precondition and postcondition of E_i after relocating the local time intervals, and propositions in PRE_E and $POST_E$ are sorted by starting points of their time intervals**
- 4: **end for**
- 5: $tPOST \leftarrow$ plotPost($POST_E$); **modify the time intervals for the propositions into disjoint intervals**
- 6: **for each** p in PRE_E **do**
- 7: **if** shareFluent(p , $POST_E$) **then**
- 8: **if** imply($p, POST_E$) **then** ** p can be satisfied by $POST_E$ **
- 9: delete(p , PRE_E);
- 10: **else** ** $POST_E$ makes p impossible**
- 11: return $\langle \phi, \phi \rangle$;
- 12: **end if**
- 13: **else**
- 14: **if not** compatible(p , PRE_E) **then**
- 15: return $\langle \phi, \phi \rangle$;
- 16: **end if**
- 17: **end if**
- 18: **end for**
- 19: $p \leftarrow$ tail($tPOST$); **get the uptodate proposition from the postcondition list**
- 20: **while** $p \neq$ null **do**
- 21: **for each** h before p in $tPOST$ **do**
- 22: **if** fluent(h) = fluent(p) **then**
- 23: delete($tPOST$, h);
- 24: **end if**
- 25: **end for**
- 26: $p \leftarrow$ getPrevious(p);
- 27: **end while**
- 28: $POST_E \leftarrow tPOST$;
- 29: return $\langle PRE_E, POST_E \rangle$;

Procedure: shareFluent

Input: a proposition p , and a proposition list $plist$

Output: a boolean value

```
1: for each  $p_t$  in  $plist$  do
2:   if  $\text{fluent}(p_t) = \text{fluent}(p)$  then
3:     return TRUE;
4:   end if
5: end for
6: return FALSE;
```

Procedure: imply

Input: a proposition p , and a proposition list $plist$

Output: a boolean value

```
1:  $p_t \leftarrow \text{getMostRecent}(plist, \text{fluent}(p), \text{time}(p))$ ;
2: if  $p_t \neq \text{null}$  and  $p_t$  imply  $p$  then
3:   return TRUE;
4: else
5:   return FALSE;
6: end if
```

Procedure: compatible

Input: a proposition p in the proposition list $plist$

Output: a boolean value, and update the proposition list $plist$

```
1: for each  $p_t$  in the  $plist$  after  $p$  do
2:    $t \leftarrow \text{join}(p, p_t)$ ; \*get the proposition satisfying the requirement to the same fluent*\
3:   if  $t \neq \text{null}$  then
4:     update( $p$ ,  $t$ ); delete( $plist$ ,  $p_t$ );
5:   else
6:     return FALSE;
7:   end if
8: end for
9: return TRUE;
```

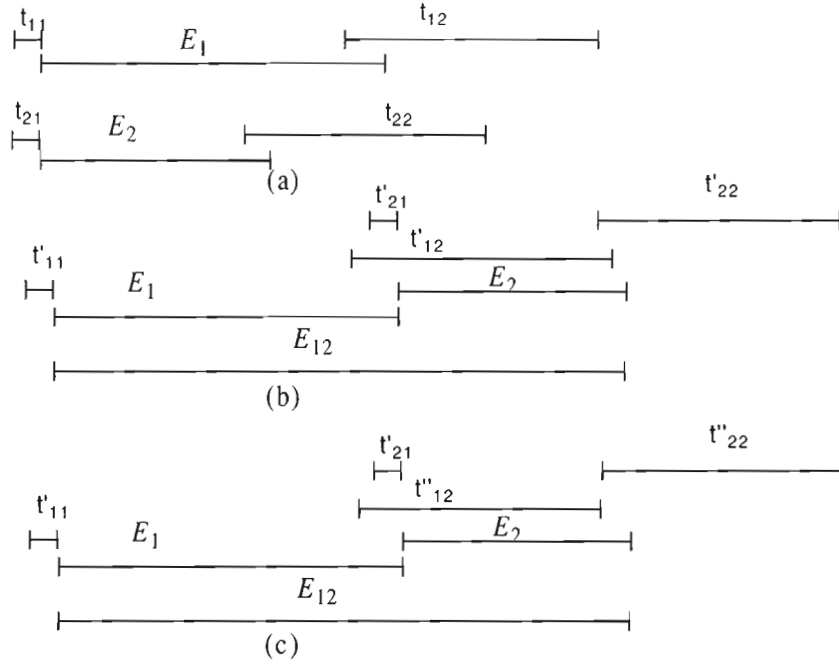


Figure 4.8: Event composition example: E_1 and E_2

$$(4.13) \quad holds(pos(ball_1) = point_1, t'_{11}) \wedge holds(pos(ball_1) = point_2, t'_{21})$$

$$(4.14) \quad holds(pos(ball_1) = point_2, t'_{12}) \wedge holds(pos(ball_1) = point_3, t'_{22})$$

Based on formula 4.14, we get the $tPOST$

$$(4.15) \quad holds(pos(ball_1) = point_2, t''_{12}) \wedge holds(pos(ball_1) = point_3, t''_{22})$$

where t''_{12} and t''_{22} are shown in Fig. 4.8(e). Since $holds(pos(ball_1) = point_2, t'_{21})$ can be implied by 4.15, it is left out of formula 4.13. The recording of $pos(ball_1)$ during t''_{12} is obsolete, and is eliminated from formula 4.15. So the representation of the composite event is

$$E_{12} = \langle holds(pos(ball_1) = point_1, t'_{11}), holds(pos(ball_1) = point_3, t''_{22}) \rangle$$

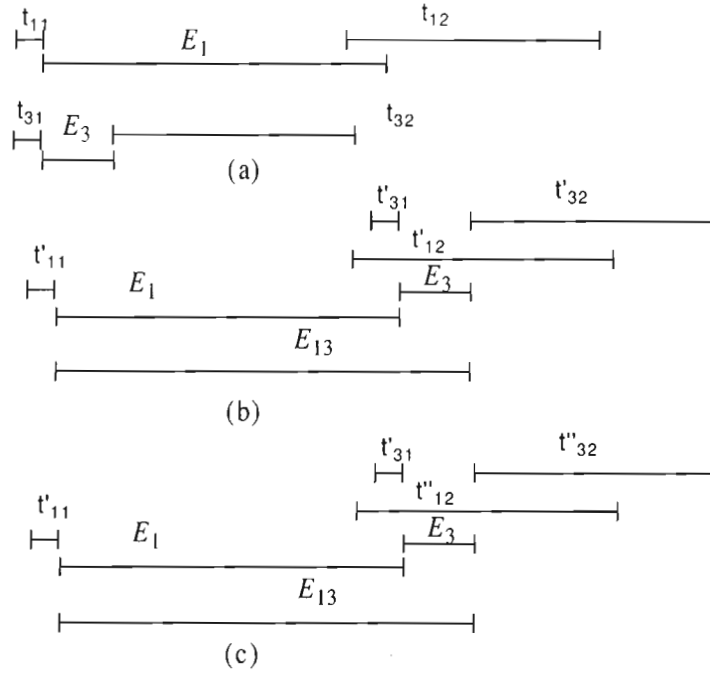


Figure 4.9: Event composition example: E_1 and E_3

To aggregate the events E_1 and E_3 , E_1 is relocated with a zero offset and E_3 is relocated with an offset of the duration of E_1 . The representation for E_3 is

$$E_3 = \langle \text{holds}(\text{pos}(\text{ball}_1) = \text{point}_2, t_{31}) \wedge \text{holds}(\text{mov}(\text{ball}_2) = \text{false}, t_{31}), \\ \text{holds}(\text{mov}(\text{ball}_2) = \text{true}, t_{32}) \rangle$$

where t_{31} and t_{32} are shown in Fig. 4.9(a). After relocating E_1 and E_3 , we get the conjunctions of their preconditions and postconditions (Fig. 4.9(b))

$$\text{holds}(\text{pos}(\text{ball}_1) = \text{point}_1, t'_{11}) \wedge \text{holds}(\text{pos}(\text{ball}_1) = \text{point}_2, t'_{31}) \\ (4.16) \quad \wedge \text{holds}(\text{mov}(\text{ball}_2) = \text{false}, t'_{31})$$

$$(4.17) \quad \text{holds}(\text{pos}(\text{ball}_1) = \text{point}_2, t'_{12}) \wedge \text{holds}(\text{mov}(\text{ball}_2) = \text{true}, t'_{32})$$

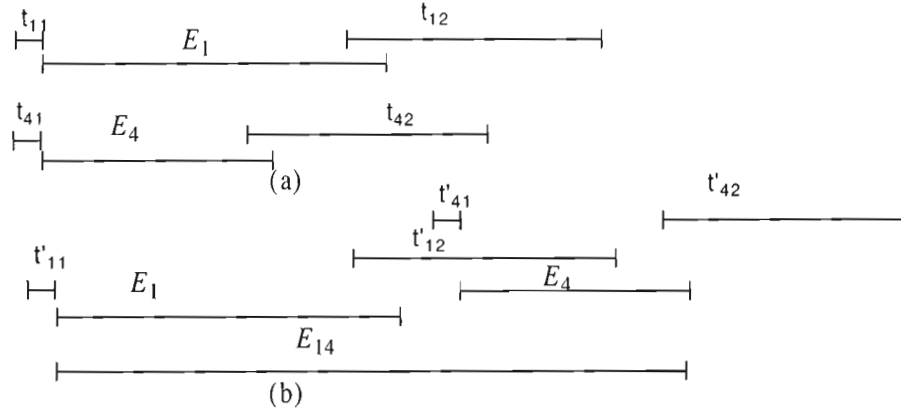


Figure 4.10: Event composition example: E_1 and E_4

Based on 4.17, we get the $tPOST$

$$(4.18) \quad holds(pos(ball_1) = point_2, t''_{12}) \wedge holds(mov(ball_2) = true, t''_{32})$$

where t''_{12} and t''_{32} are shown in Fig. 4.9(c). Since $holds(pos(ball_1) = point_2, t'_{31})$ is implied by 4.18, it is left out of formula 4.16. As a result, the representation of event E_{13} is

$$E_{13} = \langle holds(pos(ball_1) = point_1, t'_{11}) \wedge holds(mov(ball_2) = false, t'_{31}), \\ holds(pos(ball_1) = point_2, t''_{12}) \wedge holds(mov(ball_2) = true, t''_{32}) \rangle$$

The simplest case is to aggregate two events that are not related, e.g., E_1 and E_4 . Given the representation of E_4 (Fig. 4.10(a))

$$E_4 = \langle holds(pos(ball_2) = point_2, t_{41}), holds(pos(ball_2) = point_3, t_{42}) \rangle$$

the composite event E_{14} (Fig. 4.10(b)) is represented by conjunctions of its component events' preconditions and postconditions

$$E_{14} = \langle \text{holds}(\text{pos}(\text{ball}_1) = \text{point}_1, t'_{11}) \wedge \text{holds}(\text{pos}(\text{ball}_2) = \text{point}_2, t'_{41}), \\ \text{holds}(\text{pos}(\text{ball}_1) = \text{point}_2, t'_{12}) \wedge \text{holds}(\text{pos}(\text{ball}_2) = \text{point}_3, t'_{42}) \rangle$$

We calculate new time intervals t'_{11} , t'_{12} , t'_{41} , and t'_{42} through *reloc* operators.

4.4 Summary

This chapter focuses on the algorithm for constructing the event partonomy. To achieve this goal, we develop two event relations, the *f*-sequence relation and the *f*-transition relation, which represent causal relations between events; therefore, it becomes possible to approximate human perception of event partonomies.

Since the representations of composite events are not as simple as conjunctions of their component events' preconditions and postconditions, we construct an algorithm to derive the preconditions and postconditions of the composite events.

Chapter 5

PROTOTYPE

In this chapter, a prototype system is developed that implements our algorithms for constructing the event partonomy, and this is illustrated using a case study. In the following sections, we will discuss the design of the prototype, the user interface, and the implementations of data structures. To conclude the work, we demonstrate the construction of the event partonomy for the case study.

5.1 Prototype Design

The prototype system (Fig. 5.1) consists of three parts: a user interface, an Event Processing System, and an Oracle 9i database. The user interface facilitates the input of event occurrences by users and event representations by domain experts, and the display of the partonomy. The theory of constructing the event partonomy is encoded in the Event Processing System, which accesses the database through the technology of Java Database Connectivity (JDBC) (Oracle 2002). The description of events (i.e., precondition and postcondition) is stored as tables in the database, and relation schemas of these tables will be discussed in Sec. 5.3.2.

The Event Processing System not only constructs the event partonomy according to the event occurrences list, but also helps the domain expert maintain the event database. Users only specify the events stored in the database and their occurrence

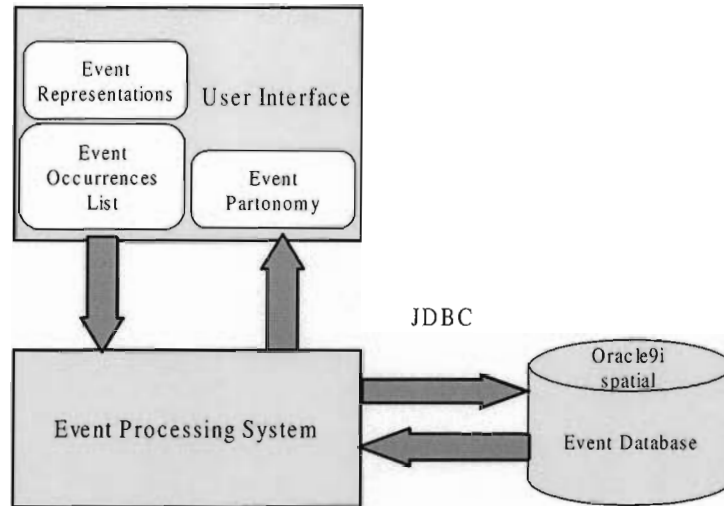


Figure 5.1: System architecture

times, while domain experts specify preconditions and postconditions of events. The functions of the system are indicated by the use case diagram (Fig. 5.2). Domain experts create, update, delete, and query descriptions of events in the database. Users query the descriptions of the events and get the event partonomy according to occurrences they input. This chapter mainly focuses on the functions provided for normal users.

The prototype was implemented in an object-oriented environment using Java 2. Java 2 Software Development Kit (J2SDK) including the core Java classes and Java's Swing classes was selected to build the system. J2SDK also includes JDBC, a standard interface used to connect Java and relational databases. In addition, we used Oracle JDBC to get the support of extensions of Oracle-specific data types and enhance the performance.

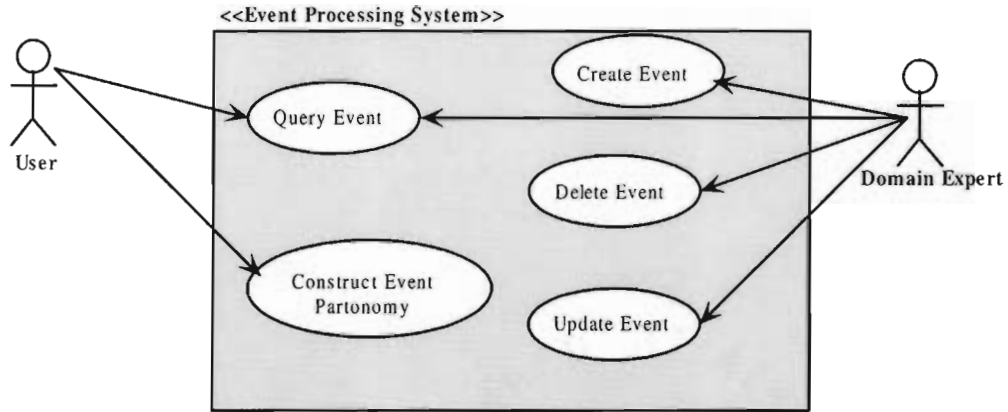


Figure 5.2: Use case diagram for the system

5.2 User Interface

The user interface facilitates the interaction between users and the system, and helps users access the following functions: querying the events and constructing the event partonomy automatically.

The layout of the application window is shown in Fig. 5.3, and it is divided into four areas for menu bar, tool bar, input panel, and a pane for displaying the results of the query and the event partonomy.

Users input is a list of event occurrences, specifying the event types and their occurrence times. These event types can be chosen from a list of events loaded from the database. Users can save their input in a file, which will be imported into the system later.

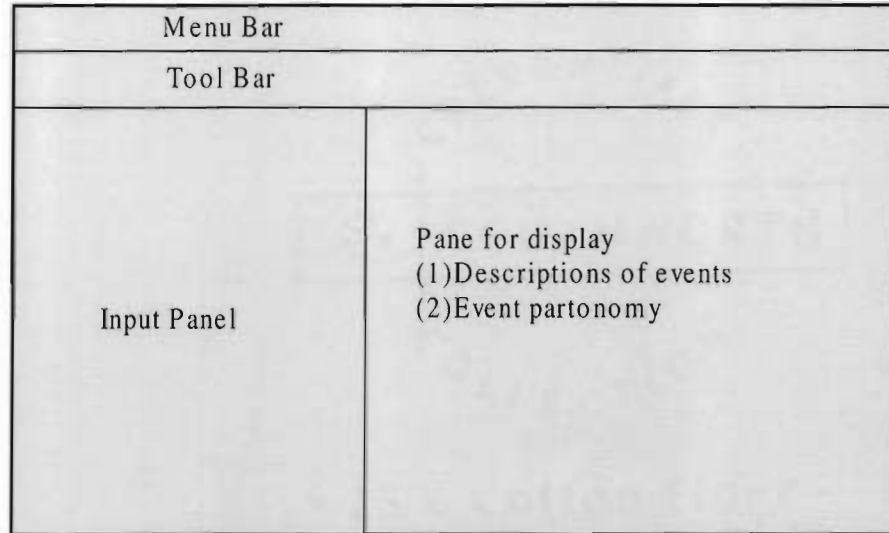


Figure 5.3: Layout of the application window

In the pane, the descriptions of selected events are given through their preconditions and postconditions in text format. The partonomy is displayed by descriptions of composite events as well as their component events names. Users can save the results in a file and load it later into the pane for review.

5.3 Implementation of Data Structure

In the prototype system, atomic events are stored in the database. Before performing the construction process, domain experts need to create the event database for the domain. It is assumed that domain experts have already created an event database for the application domain. Local time is represented quantitatively, and is given in the database as values of **interval**¹ type that are offsets to the starting point of the event. We will go through the database design in Sec. 5.3.1.

¹Type **interval** is defined in Sec. 5.4.

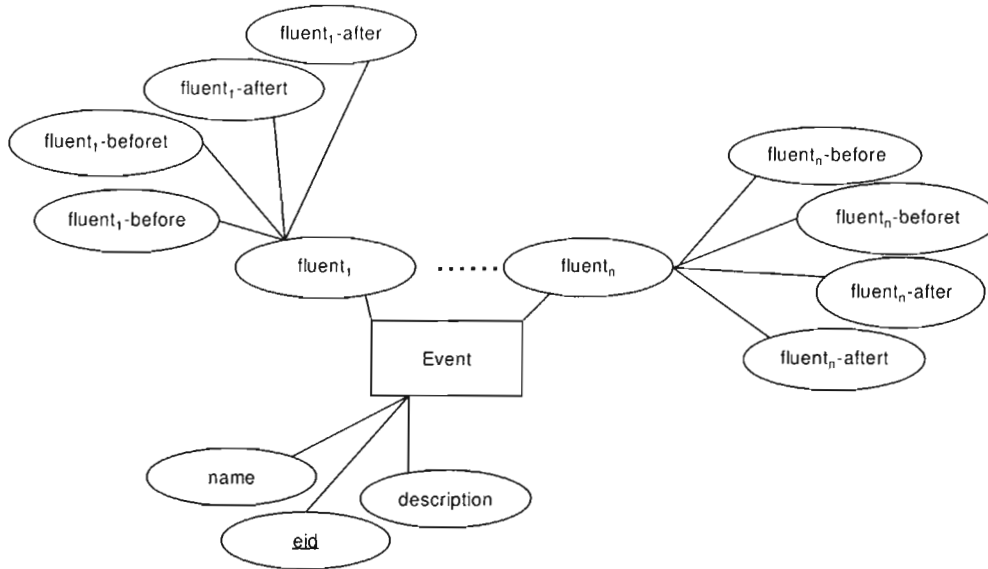


Figure 5.4: Diagram for the Event entity

Based on the representations of atomic events in the database, the system constructs the paratomy automatically using the data structures described in Sec. 5.3.2.

5.3.1 Database Schema

Events are stored as tables in the database. As described in Chapter 3, events are defined through their preconditions and postconditions. However, we represent events using their coupled form (Fig. 3.5), in which recordings of the same fluent in both events' pre- and postconditions are coupled together. For example, we use $\text{fluent}_i\text{-before}$ and $\text{fluent}_i\text{-after}$ to record values of fluent_i during time intervals $\text{fluent}_i\text{-beforet}$ and $\text{fluent}_i\text{-aftert}$ in events' pre- and postconditions. The diagram for the Event entity in the database is shown in Fig. 5.4, and attribute eid is its primary key. The relation schema for Event needs to be normalized since there are depen-

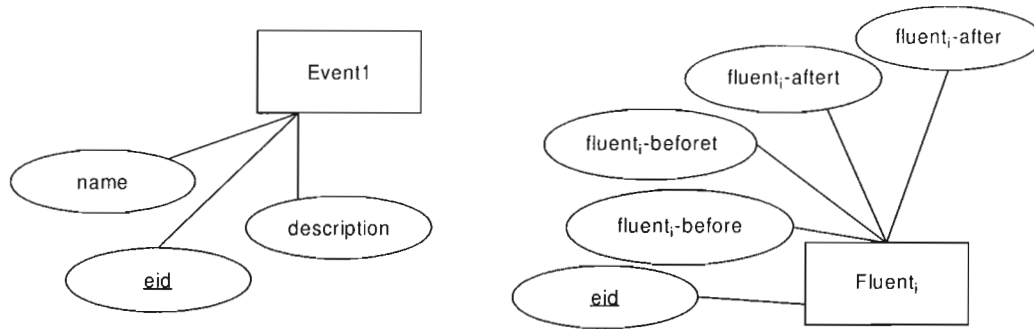


Figure 5.5: Diagram for entities Event1 and Fluent_i

dependencies between eid and different fluents (e.g., fluent_i). Therefore, the **Event** entity is decomposed into the **Event1** entity and entities for different fluents (e.g., **Fluent_i**) as Fig. 5.5 shows.

To construct the event database, domain experts need to register the fluents and their types, so that the system is able to create the fluent tables in the database with the schema:

$\text{Fluent}_i(\text{eid}, \text{fluent}_i\text{-before}, \text{fluent}_i\text{-beforet}, \text{fluent}_i\text{-after}, \text{fluent}_i\text{-aftert})$

Through registering the type of fluent_i , domain experts specify the types of attributes $\text{fluent}_i\text{-before}$ and $\text{fluent}_i\text{-after}$ in the fluent table (i.e., **Fluent_i**). Attributes $\text{fluent}_i\text{-beforet}$ and $\text{fluent}_i\text{-aftert}$ are values of type **interval**. After registering these fluents, domain experts can create events based on the fluents they have registered.

5.3.2 Event Class

During the construction of the event paratomy, we define an event class **MyEvent** to represent event occurrences.

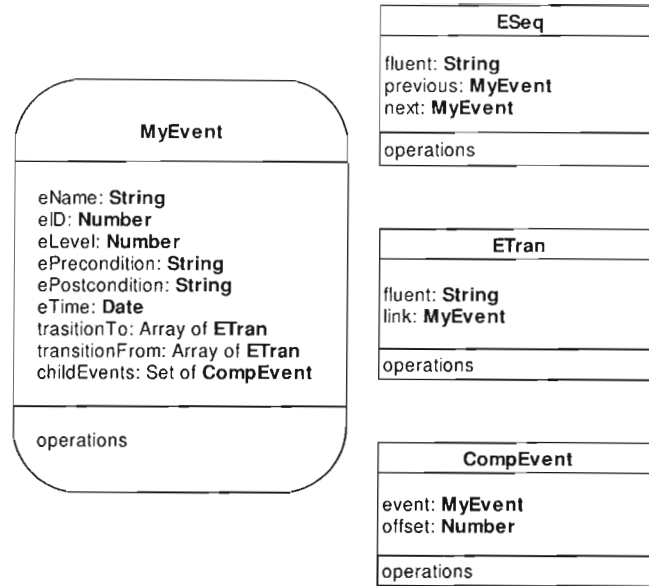


Figure 5.6: Event structure

The structure for **MyEvent** is shown in Fig. 5.6. We use the UML-based notation to describe the event structure, and specify attributes and operations as in UML static class diagrams. To make events distinct from object classes, a rounded rectangle is used. Each event has its name **eName**, identifier **eID**, precondition **ePrecondition**, and postcondition **ePostcondition**.

The event partonomy is represented through **childEvents**, which specifies a set of component events of type **MyEvent** as well as their offsets in the composite event. It is modeled through a **CompEvent** object. Atomic events do not have component events, so **childEvents** of an atomic event is null. Residing in memory, composite events are not stored permanently in the database, but their representations can be derived based on the event partonomy and representations of atomic events in the database. The derivation process is described in Algorithm 4.3. The level of an event

in the paratomy is specified through `eLevel`. Atomic events are zero-level events, and levels of composite events are one level higher than the highest level of their component events.

As discussed in Chapter 4, constructions of the event paratomy need sequence and transition relations between events. The sequence relation is implemented through a doubly-linked list structure **ESeq** in which `previous` and `next` point to the previous and the next event occurrence in the fluent-sequence. This structure not only gives the first event occurrence in each sequence but also helps determine the positions of occurrences in the sequence. Occurrences in the fluent-sequence are sorted by their fluent-influence times rather than occurrence times. To make the sequence consistent, multi-version data for the events are not permitted. Therefore, the component events will be omitted if a new composite event is created from them. This also requires that atomic events should be consistent, and each two atomic events cannot describe changes to the same fluent in an interleaving manner.

The transition relation between events is modeled through an **ETran** object. **ETran** specifies the type of transition fluent, and the event link, to which the current event is related. Attributes `transitionTo` and `transitionFrom` determine the direction of the transition. So `transitionTo` describes the transition from the current event to others, while `transitionFrom` describes the transition from other events to itself.

5.4 Case Study

To illustrate the construction process of the system, we apply the prototype system to a case study. Fig. 5.7 gives a scenario of a car accident. The vehicle car_1 starts from $place_6$ and turns left at $intB_W$, the western part of the intersection B. The vehicle car_2 starts from $place_5$ and turns right at $intB_E$. They collide at the northern part of the intersection B, $intB_N$. So the police car denoted by car_p in Fig. 5.7 makes its way to $intB_N$ where the accident has taken place, and car_3 stops to yield to the vehicle car_p . The accident blocks the traffic at intersection $intB$. Realizing that the traffic is blocked ahead, car_4 turns right at $intA_E$.

Objects car_1 , car_2 , car_3 , car_4 , car_p , $intB_W$, $intB_E$, $intB_N$, and $intA_E$ are involved in the car scenario. We use the following fluents to specify preconditions and postconditions of events (Fig. 5.8) in the scenario: $pos(car_1)$, $pos(car_2)$, $dir(car_1)$, $dir(car_2)$, $state(car_1)$, $state(car_2)$, $collision(car_1, car_2)$, $state(car_p)$, $pos(car_p)$, $dir(car_p)$, $state(car_3)$, $pos(car_3)$, $traffic(intB)$, $state(car_4)$, $pos(car_4)$, and $dir(car_4)$. Local time is specified by an interval with its starting point and ending point in seconds, and Inf indicates infinity, which means the state will last forever unless there is another event involving changes to such a state.

Domain experts create the event table and 16 fluent tables (e.g., $pos(car_1)$). The eid is a sequence created in the Oracle database, and it is used to join the **Event1** table and fluent tables.

```
create sequence event_seq start with 1000 increment by 1 nocache nocycle;
```

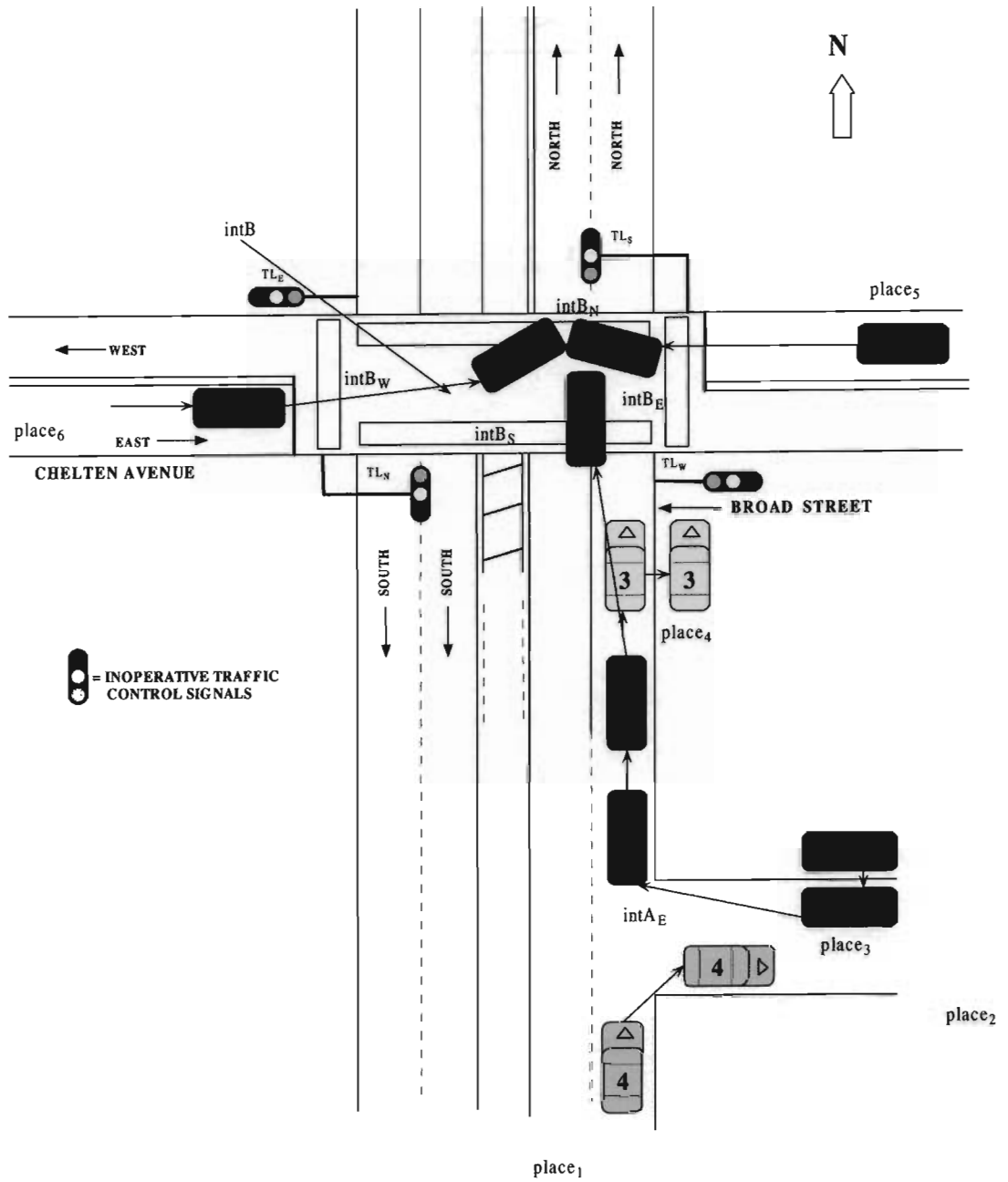


Figure 5.7: Schematic representation of a car accident scenario

Event	Name	Precondition	Postcondition
E ₁	car ₁ moves from place ₆ to intB _W	holds(state(car ₁)=go, (0,299)) holds(pos(car ₁)=place ₆ , (0, 0))	holds(pos(car ₁)=intB _W , (299, 299))
E ₂	car ₁ turns left at intB _W	holds(state(car ₁)=go, (0,15)) holds(pos(car ₁)=intB _W , (0, 0)) holds(dir(car ₁)=east, (0,5))	holds(pos(car ₁)=intB _N , (15, 15)) holds(dir(car ₁)=north, (10,15))
E ₃	car ₂ moves from place ₅ to intB _E	holds(state(car ₂)=go, (0,197)) holds(pos(car ₂)=place ₅ , (0, 0))	holds(pos(car ₂)=intB _E , (197, 197))
E ₄	car ₂ turns right at intB _E	holds(state(car ₂)=go, (0, 15)) holds(pos(car ₂)=intB _E , (0, 0)) holds(dir(car ₂)=west, (0,5))	holds(pos(car ₂)=intB _N , (15, 15)) holds(dir(car ₂)=north, (8,15))
E ₅	car ₁ and car ₂ collide at intB _N	holds(pos(car ₂)=intB _N , (0, 0)) holds(pos(car ₁)=intB _N , (0,0)) holds(collision(car ₁ , car ₂)=false, (0,2))	holds(collision(car ₁ , car ₂)=true, (2,Inf))
E ₆	car _p starts at place ₃	holds(collision(car ₁ , car ₂)=true, (0,0)) holds(state(car _p)=stop, (0,2))	holds(state(car _p)=go, (2,Inf))
E ₇	car _p moves from place ₃ to intA _E	holds(state(car _p)=go, (0,47)) holds(pos(car _p)=place ₃ , (0, 0))	holds(pos(car _p)=intA _E , (47, 47))
E ₈	car _p turns right at intA _E	holds(state(car _p)=go, (0,12)) holds(pos(car _p)=intA _E , (0, 0)) holds(dir(car _p)=west, (0,5))	holds(pos(car _p)=intA _N , (12, 12)) holds(dir(car _p)=north, (8,12))
E ₉	car _p moves from intA _N to place ₄	holds(state(car _p)=go, (0,30)) holds(pos(car _p)=intA _N , (0, 0))	holds(pos(car _p)=place ₄ , (30, 30))
E ₁₀	car ₃ stops at place ₄	holds(state(car ₃)=go, (0,4)) holds(pos(car _p)=place ₄ , (0, 0)) holds(pos(car ₃)=place ₄ , (0, 0))	holds(state(car ₃)=stop, (4,Inf))
E ₁₁	car _p moves from place ₄ to intB _N	holds(state(car _p)=go, (0,48)) holds(pos(car _p)=place ₄ , (0, 0))	holds(pos(car _p)=intB _N , (48, 48))
E ₁₂	Traffic is blocked at intB	holds(collision(car ₁ , car ₂)=true, (0,Inf)) holds(traffic(IntB)=false, (0,0))	holds(traffic(IntB)=true, (1, Inf))
E ₁₃	car ₄ moves from place ₁ to intA _E	holds(state(car ₄)=go, (0,36)) holds(pos(car ₄)=place ₁ , (0, 0))	holds(pos(car ₄)=intA _E , (36, 36))
E ₁₄	car ₄ turns right at intA _E	holds(state(car ₄)=go, (0,8)) holds(pos(car ₄)=intA _S , (0, 0)) holds(dir(car _p)=north, (0,3)) holds(traffic(IntB)=true, (0,0))	holds(pos(car ₄)=intA _E , (8,8)) holds(dir(car _p)=east, (4,8))
E ₁₅	car ₄ moves from intA _E to place ₂	holds(state(car ₄)=go, (0,44)) holds(pos(car ₄)=intA _E , (0, 0))	holds(pos(car ₄)=place ₂ , (44, 44))

Figure 5.8: Atomic events in the car accident scenario and their representations

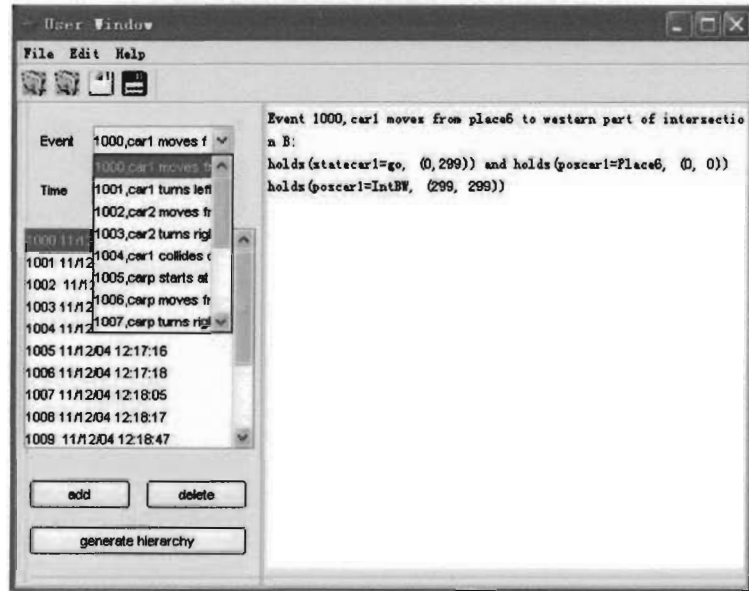


Figure 5.9: The user window for input of event occurrences

A new **interval** data type for the local time is defined in Oracle and it specifies the starting point and end point of the interval in seconds.

```
create type interval as object (t_start NUMBER, t_end NUMBER );
```

After the users input the occurrences list by means of the interface (Fig. 5.9), the system first generates the sequences and transitions, which are shown schematically in Fig. 5.10. During the construction process, **childEvents** of new composite events will be created, and an event partonomy is constructed according to values of their **childEvents**. Preconditions and postconditions of the composite events are also given in the display pane (Fig. 5.3). Fig. 5.12 gives a complete list of composite events in the event partonomy (Fig. 5.11), their descriptions, and their **childEvents**. For example, event E_{19} encompasses E_{16} and E_{17} , which encompass atomic events E_1 , E_2 , and E_3 , E_4 . So event E_{19} describes that car_1 moves from $place_6$ to $intB_N$, car_2

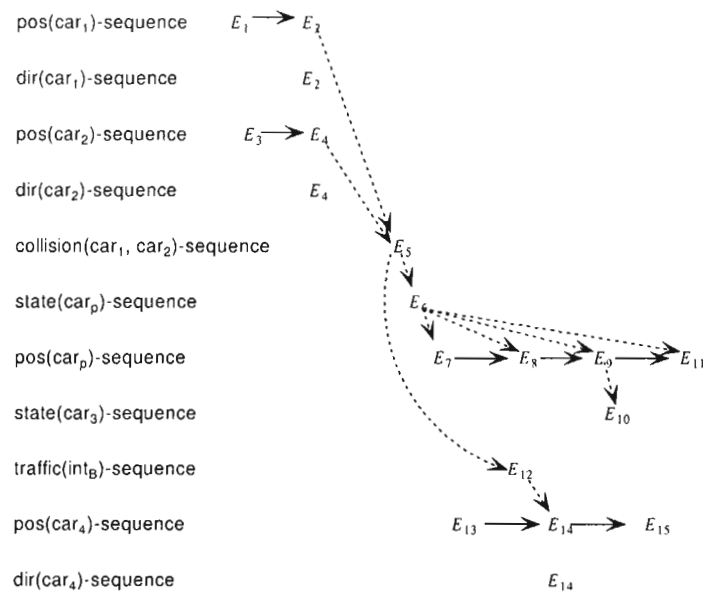


Figure 5.10: The sequences and transitions for the car accident scenario

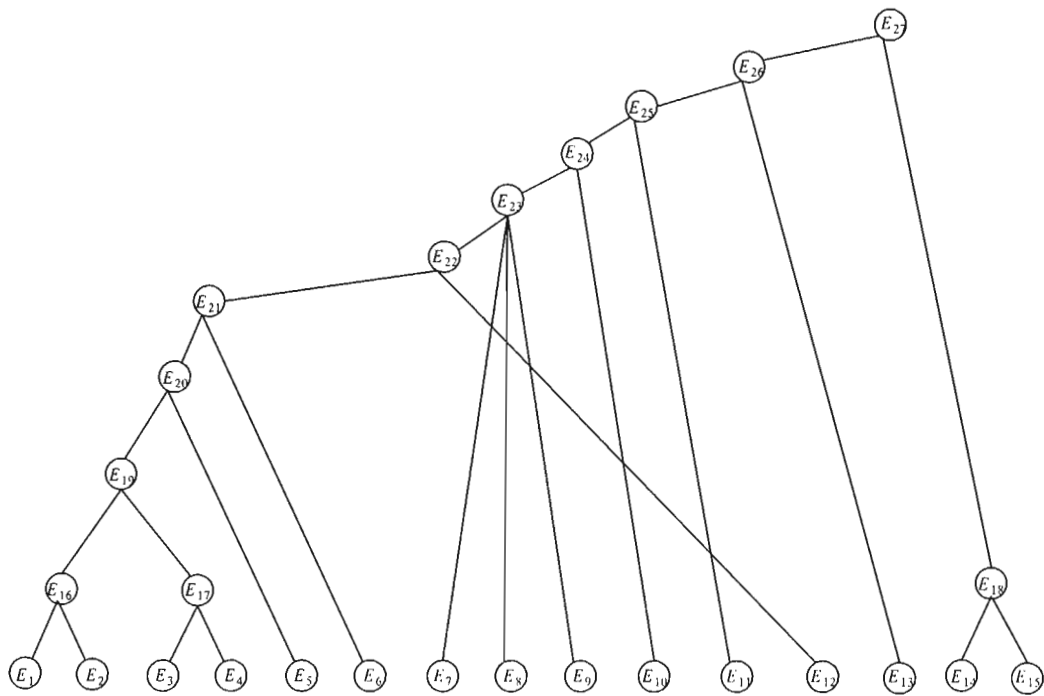


Figure 5.11: The event partitioning for the car accident scenario

Event	description	childEvents
E_{16}	car_1 moves from $place_6$ to $intB_N$ and moves northward	E_1, E_2
E_{17}	car_2 moves from $place_5$ to $intB_N$ and moves northward	E_3, E_4
E_{18}	car_4 moves from $intA_E$ to $place_2$	E_{14}, E_{15}
E_{19}	car_1 moves from $place_6$ to $intB_N$ and car_2 moves from $place_5$ to $intB_N$, and they move northward	E_{16}, E_{17}
E_{20}	car_1 and car_2 collide at $intB_N$ after car_1 moves from $place_6$ to $intB_N$ and car_2 moves from $place_5$ to $intB_N$	E_{19}, E_5
E_{21}	car_p starts at $place_3$ after car_1 and car_2 collide at $intB_N$	E_{20}, E_6
E_{22}	traffic is blocked at $intB$ after car_1 and car_2 collide at $intB_N$	E_{21}, E_{12}
E_{23}	car_p moves from $place_3$ to $place_4$ after traffic is blocked at $intB$	E_{22}, E_7, E_8, E_9
E_{24}	car_3 yields to car_p when car_p moves from $place_3$ to $place_4$ after traffic is blocked at $intB$	E_{23}, E_{10}
E_{25}	car_p moves from $place_3$ to $intB_N$ after traffic is blocked at $intB$	E_{24}, E_{11}
E_{26}	car_4 moves from $place_1$ to $intA_E$ after the traffic is blocked at $intB$	E_{25}, E_{13}
E_{27}	car_4 moves from $place_1$ to $place_2$ after the traffic is blocked at $intB$	E_{26}, E_{18}

Figure 5.12: Composite events in Fig 5.11

moves from $place_5$ to $intB_N$, and both car_1 and car_2 move northward. Event E_{20} encompasses events E_{19} and atomic event E_5 , and it describes that car_1 and car_2 collide at $intB_N$ after car_1 moves from $place_6$ to $intB_N$ and car_2 moves from $place_5$ to $intB_N$. Event E_{21} encompasses events E_{20} and E_6 , describing that the police car is coming after the accident, while event E_{22} encompasses events E_{21} and E_{12} , describing the traffic is blocked after the accident.

5.5 Summary

An implementation of the partonomy construction algorithm is given in this chapter. At the beginning, we give the design of the prototype system. We also describe the database schemas for storing atomic events in the database, and the structure of the event class **MyEvent** for representing events in the Event Processing System. To assess the construction algorithm, we apply the prototype system to the car accident scenario, and the event partonomy for the scenario is generated automatically. This prototype is evaluated in the next chapter.

Chapter 6

CONCLUSIONS AND FUTURE WORK

This chapter reviews the goal of the thesis, and summarizes the approach and the results. Possible research topics are discussed as an improvement to the model and methodology.

6.1 Summary of the Thesis

The goal of this thesis is to support multiple representations of dynamic phenomena, so that representations at the appropriate level of detail can be provided to users. It has been achieved through automatic generation of an event partonomy, which provides representations of events at different levels of detail.

In this thesis, dynamic phenomena are modeled as collections of events and relations between them. There are two event hierarchies: the event taxonomy and the event partonomy, which enable shifting between representations of events at different levels of detail. In this research, we develop two relations between events, *f*-sequence and *f*-transition, based on which an event partonomy is constructed automatically. A prototype system is also designed and applied to a car accident scenario, and it creates an event partonomy for the scenario.

6.2 Results and Major Findings

The major contributions of the thesis are to provide a general framework for representing dynamic phenomena using events, and to provide a mechanism to enable multiple representations of event-based phenomena. The major findings of the thesis are as follows.

The event-oriented model gives us more power to explicitly model events in spatio-temporal applications. In our framework, dynamic phenomena are modeled through events, represented by the common preconditions and postconditions of these event occurrences. Occurrences of the same type of event have the same pattern of precondition and postcondition, but they are different in their occurrence times. Preconditions and postconditions are related to the concepts of cause and effect relations, and they give domain experts the flexibility to define events according to their understandings of events' causes and effects. Similar to abstractions of objects, preconditions and postconditions provide a way of abstracting events.

Based on this framework, events may be formed into two event hierarchies that provide multiple representations for dynamic phenomena. In the event taxonomy, events are related through the *is-a* relation, and more specialized events may be defined as variations of more general events. This is achieved through reducing the number of *holds* predicates in events' preconditions and postconditions. The event partonomy provides another mechanism to abstract the event space, where an event may be defined as a aggregation of several component events. The event partonomy

supports representations of events at different levels of detail through aggregating several component events into one composite event, and suppressing the detail of component events.

Event-event relations play an important role in constructing the event partonomy automatically. Causal relations between events provide important contextual information, allowing events to be aggregated. To investigate causal relations between events, we develop two relations, sequence and transition, which are used to approximate to the causal relations. Based on sequence and transition relations, we develop algorithms to construct the event partonomy automatically.

6.3 Limitations of the Model and Future Work

This research is based on representations of events using their preconditions and postconditions, which only capture a part of the dynamic world. First, we cannot represent events that do not involve any changes, for instance, “Tom stands at the corner”. Second, preconditions and postconditions are necessary conditions for event occurrences. That is, the fact that an event’s precondition and postcondition are true does not force the event to occur. This results from the difficulty of formalizing causes and effects of events. Lastly, sequence and transition relations are used to approximate to causality during the construction process. But a transition between two event occurrences does not mean one event occurrence will lead to the other one. So, again it does not fully formalize the common sense notion of causality.

There may be more than one way to construct the event partonomy. The algorithm we propose constructs the event partonomy by transition and sequence relations between events, but events may be aggregated according to spatial, temporal, and spatio-temporal relations between them. In this work, users do not have options on how to construct the event partonomy. In addition, multi-version data is not permitted during the construction process, and this may lead to some potential composite events being omitted. For instance, given the transition as described in Fig. 4.2(a) (Sec. 4.1.2 and Sec. 4.2.3), we first aggregate E'_i and E_i into a composite event, E_k ; then it replaces E_i and E'_i in the f -sequence and f' -sequence. We miss the possible composite events created from E'_i and E'_j .

We have identified the following research questions:

1. *How are spatial, temporal, and spatio-temporal relations applied to the process of constructing the event partonomy?*
2. *How does the introduction of disjunction affect the representation power of the event model?*
3. *What are the algorithmic implications of adding disjunction?*

1. The sequence and transition relations are developed in Chapter 4 to approximate to the causal relation between events. Events may not be related through transition or sequence relations, but they may still be aggregated into composite events by referencing the spatial, temporal, and spatio-temporal relations between them. To address the first question, one solution could be to assign ranks to relations by default settings or by users' preference.

2. In this thesis, preconditions and postconditions of events are just conjunctions of *holds* predicates. For example, we are unable to represent the event of Tom entering Boardman if there are two doors door_1 and door_2 to the building Boardman. After we introduce disjunction, this event can be represented by

$$\begin{aligned}
E_1 = & \langle \text{holds}(\text{in}(\text{Tom}, \text{Boardman}) = \text{false}, \mathbf{t}_1) \\
& \wedge ((\text{holds}(\text{open}(\text{door}_1) = \text{true}, \mathbf{t}_2) \vee \text{holds}(\text{open}(\text{door}_2) = \text{true}, \mathbf{t}_3)), \\
& \text{holds}(\text{in}(\text{Tom}, \text{Boardman}) = \text{true}, \mathbf{t}_4) \rangle
\end{aligned}$$

Indeterminacy will be introduced when the disjunction is used for representations of events. After introducing the “or” operator in events’ preconditions and postconditions, we convert them into Disjunctive Normal Form (DNF)

$$(6.1) \quad \langle PRE_1 \vee \dots \vee PRE_n, POST_1 \vee \dots \vee POST_m \rangle$$

where each term PRE_i and $POST_i$ is a conjunction of *holds* predicates, and the occurrence of the event represented by formula 6.1 only represents the occurrence of one of the events of $E_{ij} = \langle PRE_i, POST_j \rangle$, $i = 1, \dots, n$, $j = 1, \dots, m$. Thus, disjunction represents event possibility. In our example, E_1 represents the two possible events of Tom entering Boardman through door_1

$$\begin{aligned}
E_{11} = & \langle \text{holds}(\text{in}(\text{Tom}, \text{Boardman}) = \text{false}, \mathbf{t}_1) \\
& \wedge (\text{holds}(\text{open}(\text{door}_1) = \text{true}, \mathbf{t}_2), \\
& \text{holds}(\text{in}(\text{Tom}, \text{Boardman}) = \text{true}, \mathbf{t}_4) \rangle
\end{aligned}$$

and Tom entering Boardman through door₂

$$E_{12} = \langle \text{holds}(\text{in}(\text{Tom}, \text{Boardman}) = \text{false}, t_1) \\ \wedge \text{holds}(\text{open}(\text{door}_2) = \text{true}, t_3), \\ \text{holds}(\text{in}(\text{Tom}, \text{Boardman}) = \text{true}, t_4) \rangle$$

3. After introducing the disjunction, we also need to extend the algorithms for constructing the event partition. In our example, let E_2 be the event of opening door₁, and E_3 be the event of opening door₂. There are transitions from E_2 to E_1 or from E_3 to E_1 , but it is not necessary to aggregate E_1 with both E_2 and E_3 . On the other hand, if there are disjunctions in the postcondition of one event, and there are transitions from this event to other events, these transitions are possible, but not guaranteed. This problem requires evaluations of the transitions to the events.

6.4 Summary

In this chapter, we have briefly summarized the work of this thesis and evaluated the goal of our work. We have also suggested some areas for future research.

BIBLIOGRAPHY

- Al-Taha, K. and Barrera, R. (1990), Temporal data and GIS: an overview, *in* 'GIS/LIS'90', Anaheim, CA, US, pp. 244–254.
- Allen, J. F. (1983), 'Maintaining knowledge about temporal intervals', *Communications of the ACM* **26**, 832–843.
- Allen, J. F. (1984), 'Towards a general theory of action and time', *Artificial Intelligence* **23**(2), 123–154.
- Allen, J. F. and Ferguson, G. (1994), Actions and events in interval temporal logic, Technical Report TR521, Computer Science Department, University of Rochester.
- Barker, R. G. and Wright, H. F. (1954), *Midwest and its Children: The Psychological Ecology of an American Town*, Row, Peterson and Company, Evanston, Illinois.
- Bittner, T. (2002), An ontology for spatio-temporal databases. <http://citeseer.nj.nec.com/469674.html>.
- Buttenfield, B. and Delotto, J. (1989), Multiple representation, Technical Report 89-3, National Center for Geographic Information and Analysis (NCGIA), Santa Barbara, CA.
- Davidson, D. (1967), The logical form of action sentences, *in* N. Rescher, ed., 'The Logic of Decision and Action', University of Pittsburgh Press, pp. 96–103.
- Davidson, D. (1969), The individuation of events, *in* N. Rescher, ed., 'Essays in Honour of Carl G. Hempel', D. Reidel Publishing, Dordrecht, Holland, pp. 216–234.
- Egenhofer, M. and Golledge, R. (1994), Time in geographic space: Report on the specialist meeting of research initiative 10, Technical Report 94-9, National Center for Geographic Information and Analysis (NCGIA), Santa Barbara, CA.

- Frank, A. (1994), Qualitative temporal reasoning in GIS-ordered time scales, *in* T. Waugh and R. Healey, eds., ‘Proceedings of Sixth International Symposium on Spatial Data Handling’, Edinburgh, Scotland, pp. 410–431.
- Galton, A. (1995), Towards a qualitative theory of movement, *in* A. Frank and W. Kuhn, eds., ‘Spatial Information Theory: A Theoretical Basis for GIS’, Springer-Verlag, Berlin, pp. 377–396.
- Galton, A. (2000), *Qualitative Spatial Change*, Oxford University Press, New York, US.
- Galton, A. (2003), Desiderata for a spatio-temporal geo-ontology, *in* W. Kuhn, M. F. Worboys and S. Timpf, eds., ‘COSIT 2003’, Kartause Ittingen, Switzerland, pp. 1–12.
- Gibson, J. J. (1986), *The Ecological Approach to Visual Perception*, Lawrence Erlbaum, Hillsdale, NJ, US.
- Grenon, P. and Smith, B. (2004), ‘SNAP and SPAN: towards dynamic spatial ontology’, *Spatial Cognition and Computation* 4(1), 69–103.
- Gries, D. (1981), *The Science of Programming*, Monographs in Computer Science, Springer-Verlag, Secaucus, NJ, USA.
- Han, J. and Kambr, M. (2001), *Data Mining: Concepts and Techniques*, Morgan Kaufmann, San Francisco, California.
- Hirtle, S. (1995), Representational structures for cognitive space: Tree, ordered trees and semi-lattice, *in* A. Frank and W. Kuhn, eds., ‘Spatial Information Theory: A Theoretical Basis for GIS’, Springer-Verlag, Berlin, pp. 327–340.
- Hirtle, S. and Jonides, J. (1985), ‘Evidence of hierarchies in cognitive maps’, *Memory and Cognition* 13(3), 208–217.
- Hoare, C. A. R. (1969), ‘An axiomatic basis for computer programming’, *Communications of the ACM* 12(10), 576–580.
- Hobbs, J. R. (1990), Granularity, *in* D. Weld and J. Kleer, eds., ‘Readings in Qualitative Reasoning about Physical Systems’, Morgan Kaufmann, San Mateo, CA, US, pp. 542–545.
- Hornsby, K. E. (1999), Identity-based reasoning about spatio-temporal change, Ph.D thesis, University of Maine.
- Hornsby, K. E. and Egenhofer, M. J. (2002), ‘Modeling moving objects over multiple granularities’, *Annals of Mathematics and Artificial Intelligence* 36, 177–194.

- Hume, D. (1739), *A Treatise of Human Nature*, Oxford University Press, Reprint edition (1980), Oxford, UK.
- Kant, I. (1781), *Critique of Pure Reason*, St. Martin's Press, Reprint edition (1965), New York.
- Koestler, A. (1967), *The Ghost in the Machine*, Macmillan, New York.
- Kowalski, R. and Sergot, M. (1986), 'A logic-based calculus of event', *New generation computing* 4(1), 67–95.
- McCarthy, J. and Hayes, P. J. (1969), Some philosophical problems from the standpoint of artificial intelligence, in B. Meltzer and D. Michie, eds., 'Machine Intelligence', Vol. 4, American Elsevier Publishing, pp. 463–502.
- McDermott, D. V. (1982), 'A temporal logic for reasoning about processes and plans', *Cognitive Science* 6, 101–155.
- Meyer, B. (1988), *Object-Oriented Software Construction*, Prentice Hall, Englewood Cliffs, NJ.
- Michotte, A. E. (1963), *The Perception of Causality*, Basic Books, New York.
- Mortensen, C. (2002), Change, in E. N. Zalta, ed., 'The Stanford Encyclopedia of Philosophy (Winter 2002 Edition)', <http://plato.stanford.edu/archives/win2002/entries/change/>.
- Oracle (2002), 'JDBC developer's guide and reference'.
- Pearl, J. (2000), *Causality: Models, Reasoning, and Inference*, Cambridge University Press, Cambridge, UK.
- Peuquet, D. and Duan, N. (1995), 'An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographical data', *International Journal of Geographical Information System* 9(1), 7–24.
- Peuquet, D. and Wentz, E. (1994), An approach for time-based analysis of spatiotemporal data, in T. Waugh and R. Healey, eds., 'Proceedings of Sixth International Symposium on Spatial Data Handling', Edinburgh, Scotland, pp. 489–504.
- Ramakrishnan, R. and Gehrke, J. (2000), *Database Management Systems*, McGraw Hill.
- Shoham, Y. and Goyal, N. (1988), Representing time and action in artificial intelligence. <http://citeseer.ist.psu.edu/154456.html>.

- Smith, J. M. and Smith, D. C. (1977), 'Database abstractions: Aggregation and generalization', *ACM Transactions on Database Systems* 2(2), 105–133.
- Stell, J. and Worboys, M. F. (1999), Generalizing graphs using amalgamation and selection, in R. Gutting, D. Papadias and F. Lochovsky, eds., 'Advances in Spatial Databases, 6th International Symposium, SSD'99', Hong Kong, China, pp. 19–32.
- Timpf, S. (1999), Abstraction, levels of detail, and hierarchies in map series, in C. Freksa and D. Mark, eds., 'Spatial Information Theory - Cognitive and Computational Foundations of Geographic Information Science', Springer-Verlag, Stade, Germany, pp. 125–140.
- Worboys, M. F. (1994), 'A unified model of spatial and temporal information', *Computer Journal* 37(1), 26–34.
- Worboys, M. F. (1995), *GIS: A Computing Perspective*, Taylor & Francis, London, UK.
- Worboys, M. F. (2001), Modeling changes and events in dynamic spatial systems with reference to socio-economic units, in A. Frank, J. Raper and J.-P. Cheylan, eds., 'Life and Motion of Socio-Economic Units', Taylor & Francis, London, UK, pp. 129–138.
- Worboys, M. F. (2003), Knowledge discovery using geosensor networks, in 'Geo-Sensor Network', Portland, ME.
- Worboys, M. F. (2005), 'Event-oriented approaches to geographic phenomena', *International Journal of Geographical Information Science*, *Accepted for publication*.
- Worboys, M. F. and Hornsby, K. (2004), From objects to events: GEM, the geospatial event model, in 'GIScience 2004', Maryland.
- Zacks, J. and Tversky, B. (2001), 'Event structure in perception and conception', *Psychological Bulletin* 127(1), 3–21.

BIOGRAPHY OF THE AUTHOR

Rui Zhang was born in Ningguo, P. R. China on May 14, 1978. He attended school in Jixi, and graduated from Jixi Middle School in 1995.

He obtained a B.S. in Mathematics from Nanjing University, Nanjing, P. R. China in 1999, and an M.E. in Computer Science from Nanjing University in 2002. His research interests during this time related to security database. He was a research assistant at the Database Laboratory in Nanjing University, and was involved in the project of designing a security database management system at B1 level. He was also interested in spatial database, and his master thesis was titled, "An Implementation of Spatial Database Based on PostgreSQL." He worked as a part time software programmer for NandaSoft Co. Ltd., Nanjing, P. R. China when he was studying for his master degree.

After moving to Maine in September 2002, he was enrolled for graduate study at the University of Maine and served as a Graduate Research Assistant in the Department of Spatial Information Science and Engineering. He is a candidate for the Master of Science degree in Spatial Information Science and Engineering from the University of Maine in May, 2005