Lecture Notes in
Computer Science 759

Nabil R. Adam Bharat K. Bhargava (Eds.)

Advanced Database Systems

Springer-Verlag

A-264 Nabil R. Adam Bharat K. Bhargava (Eds.)

Advanced Database Systems

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Lecture Notes in Computer Science 759

Contents

Part I

Object-Oriented Databases

Chapter 1

COMPOSE: A System For Composite Specification And Detection

Narain Gehani*, H. V. Jagadish†, O. Shmueli‡

1.1 Introduction

An "event" is a happening of interest. Events can be simple such as, the stock price going above a certain price, the beginning of a transaction, the update of an object, or the temperature going above a specified limit. New events can also be formed as a combination of other events, for example, three successive discount rate cuts without an intervening increase, all withdrawals following a large deposit, and the temperature going above a specified limit and staying there for more than some time period. We call such events "composite events".

We have developed a model for specifying composite events [216, 215]. We were motivated to explore the specification of composite events as part of an effort to design "trigger" facilities for the Ode object database [7, 217]. Triggers are the key facility that distinguishes active databases [138, 524, 40, 378, 556, 395] from passive databases. A trigger consists of an event-action pair. When an event is detected, the associated action is executed.

The use of triggers moves code from the application to the database. This simplifies application writing because the application now does not have to check for the conditions specified by the triggers. Triggers also eliminate duplicate code since the same conditions may have to be checked in multiple applications.

A trigger facility in which triggers fire on the occurrence of composite events is more powerful than one in which triggers fire on the occurrence of simple events because it allows users to write triggers that could not be easily expressed before. Composite event specification is useful for many application domains besides databases:

1. Financial Applications: Trades can be executed in response to an observed pattern of (trading) events in a stock market.

*AT&T Bell Laboratories
†AT&T Bell Laboratories
‡AT&T Bell Laboratories and Technion -Israel Institute of Technology

2. Fraud Detection: Particular sequences of credit card purchases may point to fraudulent use.

3. Production Management: Particular sequences of defects could indicate difficulties that must be brought to the attention of a supervisor.

Composite events are specified as event expressions. Our basic notation for specifying composite events has the same expressive power as regular expressions. Thus the occurrence of a composite event is detected by a finite automaton that implements the event expression. Despite the equivalence of expressive power, our notation is specially suited for specifying composite events. For example, it allows for the easy specification of composite events whose components can overlap and allows uninteresting events to be screened out.

We extend our basic notation with "masks", correlation variables, and parameters, thereby stepping beyond the domain of regular expressions. However, we can still implement event expressions that use these facilities by using automata augmented with "mask" events and by using "generic" automata. This allows us to use finite automata optimization techniques to generate efficient implementations for recognizing the occurrence of composite events.

We have built a prototype system, COMPOSE, for specifying and detecting composite events. A real-time stock trade feed is used to experiment with specifying and detecting stock market related events.

In this chapter, we describe how composite events are specified, illustrate composite event specification, give an overview of COMPOSE, and describe the construction of the finite automata.

1.2 Event Expressions

Primitive events are events that are known to or supported by the database system. Examples of some primitive events, in object-oriented databases [215], are object manipulation actions such as creation, deletion, and update or access by an object method (member function). Events can be specified to happen just prior to or just after the above actions. In addition, events can be associated with transactions and specified to happen immediately after a transaction begins, immediately before a transaction attempts to commit, immediately after a transaction commits, immediately before a transaction aborts, and immediately after a transaction aborts. Examples of other events are time events such as clock ticks, the passage of a day, an hour, a second, or some other time unit. Finally, stock trades and the raising or lowering of interest rates are examples of financial events, and company announcements are examples of news events.

Composite events are specified as event expressions. An *event expression* can be $NULL$, any primitive event a, or an expression formed using the basic operators $\wedge$, ! (not), *relative* and *relative+*.

Formally, event expressions are mappings from *histories* (sequences of primitive events) to histories:

$$E : histories \rightarrow histories$$

The result of applying an expression E to a history h, which is also a history, is denoted by $E[h]$.

Here are the semantics of some simple event expressions:

1. $E[null] = null$ for any event E, where $null$ is the empty history.
2. $NULL[h] = null$.
3. $a[h]$, where a is a primitive event, is the maximal subset of h composed of all event occurrences of the symbol a.

1.2.1 Basic Operators

Let E and F denote event expressions and h, h_1, h_2 denote event histories. Here are the semantics of expressions formed using the basic operators:

1. $(E \wedge F)[h] = h_1 \cap h_2$ where $h_1 = E[h]$ and $h_2 = F[h]$.
2. $(!E)[h] = (h - E[h])$.
3. $relative(E, F)[h]$ are the event occurrences in h at which F is satisfied assuming that the history started immediately following *some* event occurrence in h at which E takes place.

   Formally, $relative(E, F)[h]$ is defined as follows. Let $E^i[h]$ be the i^{th} event occurrences in $E[h]$; let h_i be obtained from h by deleting all event occurrences before $E^i[h]$. Then $relative(E, F)[h] = \bigcup_{i=1}^{E[h]} F[h_i]$.
4. $relative+(E)[h] = \bigcup_{i=1}^{\infty} relative^i(E)[h]$ where $relative^1(E) = E$ and $relative^i(E) = relative(relative^{i-1}(E), E)$.

1.2.2 Additional Operators

Besides the basic operators, we provide some additional operators that make composite events easier to specify. These operators do not add to the expressive power provided by the basic operators. Consequently, they can be defined in terms of the basic operators.

Let h denote a non-null history, and E, F, and E_i denote event expressions. The new operators are

1. $E \vee F = !(!E \wedge !F)$.
2. *any* denotes the disjunction of all the primitive events.
3. $prior(E, F)$ specifies that an event F that takes place after an event E has taken place. E and F may overlap. Formally, $prior(E, F) = relative(E, any) \wedge F$.
4. $prior(E_1, ..., E_m)$ specifies occurrences, in order, of the events E_1, E_2 ,..., E_m. $prior(E_1, ..., E_m) = prior((prior(E_1, ..., E_m - 1), E_m)$.
5. $sequence(E_1, ..., E_m)$ specifies immediately successive occurrences of the events E_1, E_2 ,..., E_m:

   (a) $sequence(E_1, ..., E_m) = sequence((sequence(E_1, ..., E_m - 1), E_m)$.

   (b) $sequence(E_1, E_2) = relative(E_1, !(relative(any, any))) \wedge E_2$.
   The first operand of the conjunction specifies the first event following event E_1. The second operand specifies that the event specified by the complete event expression must satisfy E_2.
6. *first* identifies the first event in a history.
   $first = !relative(any, any)$.