Composite Events in Chimera

Rosa Meo\textsuperscript{1} \hspace{1cm} Giuseppe Psaila\textsuperscript{1} \hspace{1cm} Stefano Ceri\textsuperscript{2}

\textsuperscript{1} Politecnico di Torino, Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi, 24 - I-10129 Torino, Italy
\textsuperscript{2} Politecnico di Milano, Dipartimento di Elettronica e Informazione
Piazza L. Da Vinci, 32 - I-20133 Milano, Italy
rosimeo@polito.it, psaila/ceri@elet.polimi.it

Abstract. In this paper, we extend event types supported by Chimera, an active object-oriented database system. Chimera rules currently support disjunctive expressions of set-oriented, elementary event types; our proposal introduces instance-oriented event types, arbitrary boolean expressions (including negation), and precedence operators. Thus, we introduce a new event calculus, whose distinguishing feature is to support a minimal set of orthogonal operators which can be arbitrarily composed. We use event calculus to determine when rules are triggered; this is a change of each rule's internal status which makes it suitable for being considered by the rule selection mechanism.

The proposed extensions do not affect the way in which rules are processed after their triggering; therefore, this proposal is continuously evolving the syntax and semantics of Chimera in the dimension of event composition, without compromising its other semantic features. For this reason, we believe that the proposed approach can be similarly applied for extending the event language of other active database systems currently supporting simple events or their disjunction.

1 Introduction

Active database systems provide tight integration of Event-Condition-Action (ECA) rules within a database system. Such a tight integration is normally achieved by reusing database system components for implementing conditions (database predicates or queries) and actions (database manipulations, often embedded within a procedural component). In general, when a rule is selected for execution (or triggered), then its condition is evaluated (or considered), and if the condition is satisfied, then the action is immediately executed.\textsuperscript{3} Thus, the condition-action (CA) components of an active database have a simple and uniform behavior, which is common to most active databases.

Instead, event type specification, evaluation, and coupling to conditions and actions have to be designed and implemented specifically for each active database system. Thus, it is not surprising that the notions of elementary event type, of event type composition, and of binding between event occurrences and the CA

\textsuperscript{3} An exception is HiPAC [9] which supports several coupling modes between conditions and actions.
components are quite different in each active database, and such differences are responsible for most of the diversity of active databases.

Most active databases recognize just data manipulation operations (such as insert, delete, and update) as event types. The proposed SQL3 standard, currently under development by ANSI and ISO, associates to each rule just one event type; this can be considered as the simple extreme of a spectrum of solutions [17]. Most relational database products supporting active rules (called triggers) associate each of them to a disjunction of event types whose instances are relative to the same table [23]; this solution is also used by Starburst [24], Postgres [21], and Chimera, an active object-oriented database prototype developed at Politecnico di Milano in the context of the IDEA Esprit Project [4, 5]. More complex event calculus are supported by active database prototypes (see Section 1.1). In these approaches, rules are associated to event expressions which normally include generic boolean expressions, precedence operations, and explicit time references.

In all active rule systems, event instances cause rules to change an internal state; the corresponding state transition is called triggering of the rule. Once a rule is triggered, active rule systems react in several ways. When multiple rules are triggered at the same time, a rule selection mechanism determines which of them should be considered first; this mechanism may be influenced by priorities which are statically associated to rules. In addition, the rule selection may occur immediately after the triggering operation or be deferred to some later point in transaction execution (such as the commit time). With immediate execution, it is possible to further identify the cases of rules executing before, after, or instead of the operation generating the triggering event occurrence. Finally, the triggering and execution of rules can be repeated for each tuple or object affected by an operation (row-level granularity in [17]) or instead relate to the overall set of tuples or objects manipulated by means of the same operation (statement-level granularity in [17]).

Due to all these alternatives, active rule systems present themselves with a variety of possible behaviors (a thorough comparative analysis of semantics supported by active rule systems is presented in [10]). In order to control the introduction of complex events in Chimera, and therefore the increase of semantic complexity due to this extension, we have strictly followed some design principles:

- We have defined the event calculus by means of a minimal set of orthogonal operators.
- The semantics of the event calculus is given simply by defining the conditions upon which rules having as event type a complex event calculus expression become triggered; detriggering occurs when a rule is selected for consideration and execution. No other state transitions characterize the internal state of each rule.
- The event calculus extension does not affect the way in which rules are processed after their triggering; therefore, this proposal continuously evolves the syntax and semantics of Chimera in the dimension of event type composition, without compromising its other semantic features.