Event Processing in Mobile and Active Database Systems Using Broadcasts

Stefan Sauer
Universität Paderborn
Fachbereich Mathematik/Informatik
33095 Paderborn, Germany
sauer@uni-paderborn.de

Olaf Zukunft
Universität Oldenburg
Fachbereich Informatik
26111 Oldenburg, Germany
zukunft@acm.org

Abstract

The integration of active database technology into mobile database systems allows to limit the transfer of data between mobile computers and the stationary network. In order to employ such active database mechanisms, the rule processing component of traditional active database systems needs to be modified. This paper introduces a new model for event processing and presents a new algorithm for the composition of complex events in a mobile database environment. Furthermore, the model also takes into account the possibility of broadcast data. To validate the feasibility of the model, a prototype has been built using object-oriented technology.

1. Introduction

Recently, mobile computing has received a lot of attention. The management of data in an environment with mobile computers equipped to establish a wireless connection to a stationary network is performed by mobile database systems. These database systems are designed with the constraints of mobile environments like poor resources of the mobile computer and small bandwidth of the wireless network in mind.

One key constraint for any mobile database system is that data should only be transferred between the mobile computer and the stationary network if it is absolutely necessary. A typical situation that requires a data transfer is the update of data replicated on a mobile computer. In order to cope with this kind of situations, we have combined the concepts of mobile and active database systems. An active database system incorporates an event-based rule system into the DBMS [7]. In earlier work, we have shown the feasibility of combining mobile and active DBMS [14] and discussed issues of transaction processing [15].

In this paper, we want to examine the effect mobile computing has on the event processing of the active DBMS. The processing of events has already received a lot of attention (see for example [5, 13]). However, all existing work assumes either a centralized or a strongly connected network of computers running the active DBMS. If we integrate mobile computers into our DBMS, we have to modify existing event processing techniques to account for belated or undelivered events due to the possible disconnection of mobile units. Further opportunities arise if we assume that a broadcast of data sent through the air also contains events relevant for mobile computers.

In the remainder of this paper, we will discuss the problems that arise in a mobile and active environment. We start with an introduction to basic terms of mobile and active database technology and discuss related work. We present a new model for event processing in section 3. It defines the notion of a conflicting event, includes a new algorithm for event detection in mobile active database systems and describes the possible use of broadcast events for correct event composition. The design and implementation of MESaverde, a prototype using this model, is given in section 4. Finally, we give the conclusion drawn from our work and discuss some future perspectives.

2. Preliminaries

In order to gain a common understanding of basic terms, we give a short overview about the three research areas mobile computing, active database systems, and broadcast technology related to our work.

2.1. Mobile Computing

Our understanding of mobile computing follows the commonly accepted model as given in [9]. Mobile users have access to a portable computer and may use a wireless or a wired network. The wireless network is based on a cellular organization, where certain computers of the stationary network are equipped with devices to supply a geographical area with wireless network services. These com-
puters are called base stations. A characteristic feature of this model is that mobile users try to satisfy data accesses locally and to use network facilities only if absolutely necessary. This strategy accounts for resource limitations regarding energy and processing power as well as the usually high costs charged for using wireless networks. Hence, the state of disconnection is common for mobile users. In order to allow a processing of data in the disconnected state, relevant data is usually replicated on the mobile computer. The problem of keeping replicated data on disconnected or weakly connected mobile computers consistent has received a lot of attention (see for example [12]). Other topics in mobile database system research can be found in [2, 8].

2.2. Active Database Systems

The basic idea of active database systems is to integrate an event-based rule system into the DBMS. HiPAC [6], the first active database system, introduced the ECA-rule model: On signaling of an event, a condition is evaluated and if this evaluation yields true, an action is performed. All active database systems include an event algebra to specify complex events using operators like disjunction, conjunction and sequence. For the detection of complex events, various algorithms have been proposed, (see e.g. for example [5, 13]). The relation between mobile and active database systems is that active database technology can be ideally used to solve typical tasks within mobile database systems: Using appropriate rules, we can for example limit the transfer of data between the mobile computer and the stationary network to those situations where some required data is available. This prevents the resource-poor mobile computer from polling the stationary database over a slow and expensive wireless network link [11].

2.3. Broadcast Technologies

One characteristic feature of wireless networks is that they usually broadcast their data to all recipients within a given geographical area. In correspondence with this physical feature of the used network technology, equivalent technologies have emerged at a more abstract level. Broadcast disks [1] are used to disseminate data traditionally stored on disks by a repeated transfer of the data through a wireless broadcast to all recipients of a wireless cell. The organization of data broadcasts has also been investigated [10] and different proposals for broadcast organizations have been made with respect to the trade-off between access time for a data item and the tuning time required from the mobile computer. Using data broadcast to communicate control information in the context of concurrency control has been presented in [4].

The activity paradigm of active database systems also coincides with the concepts of broadcasting networks. There, the events are “broadcast” to all rule processing computers and to all event handlers corresponding to rules in the system. This is contrary to messages exchanged in object-oriented systems, where one specific object is addressed by the sender. Hence, the basic construct of active database systems ideally fits into the broadcast technology if we assume that events can also be broadcast (see section 3.4).

3. A New Model for Event Processing

In this section, we present our model for event processing in a mobile active DBMS based on the system model shown in the previous section. We first give an overview of what instances are involved in the event processing followed by a formalization of conflicts and an algorithm for handling such conflicts within the process of complex event composition. Finally, we describe an integrative model of broadcasting and signaling of events.

3.1. Classification of Event Detection

The basic architecture of the mobile system consists of the entity types stationary database server, base station, and mobile client. A mobile computer can be in a connected or disconnected state. Additionally, mobile clients can initiate proxies that run on a base station on behalf of the mobile client. Events can be signaled and processed by stationary as well as mobile computers. In the following, we concentrate on those cases specific to environments with mobility. Those scenarios in which events are signaled and handled by processes that run on any type of stationary host only can be regarded as specializations of the more general mobile cases. We therefore have to focus on three different scenarios:

- event processing involving a server and a mobile client with possible disconnections.
- handling of locally signaled events on a mobile computer with or without network connection.
- event processing between a server and a mobile client by introducing a proxy with possible disconnections.

3.2. Conflict Detection and Handling

To allow for a distributed event processing and detection of composite, complex events we need to consider the timestamps of event instances, i.e. the logical point in time when they are raised, under the assumption of synchronized clocks. Timestamps define a chronological order of events
that can conflict with the order in which events are delivered to a remote event handler. This holds especially in a mobile system where the signaling of remote events can be delayed by a slow network connection or even a disconnection. In this context, a disconnection can be viewed as a long, not equally distributed network delay. The longer a disconnection lasts, the higher is the effort to be additionally spent for synchronization. This effort also depends on the consistency requirements of a specific application, possibly varying among event types, and the conflict frequency. It must be decided when the effort for synchronization (for a specific event type) is no longer tenable and other reactions like compensating events, ignoring of event instances etc. need to be done. To cover the problem of conflicting events we first give a formal definition of what a conflict in the sequence of raised events is.

**Definition 1: Conflict, conflicting and conflicted events**

A conflict in the event processing occurs if an event instance \( e^* \) of an event type \( E \) with timestamp \( t(e^*) \) is signaled at an event detector at time \( t_{ed}(e^*) \) and there exists at least one event instance \( e \) of the same type with a later timestamp \( t(e) \) that has already been recognized in the event detection process at an earlier time \( t_{ed}(e) \), i.e.: \( \exists e^*, e \in EI : \)

\[
t(e^*) < t(e) \land t_{ed}(e^*) > t_{ed}(e) \land \text{type}(e^*) = \text{type}(e)
\]

where \( EI = ES \cup CE \) denotes the set of raised event instances as a union of atomic \( ES \) and complex \( CE \) events and \( \text{type} : EI \to ET \) is a function that maps an event instance \( e \) on its corresponding event type \( \text{type}(e) \in ET \), and \( ET = Type_a \cup Type_c \) is the set of corresponding atomic and complex event types. We call \( e^* \) the conflicting event and \( e \) the conflicted event.

In this work, we focus on detecting conflicts at the level of atomic events and show how to handle complex events that are composed using conflicting atomic events. We therefore specialize this definition to match sets of conflicting and conflicted atomic events.

**Definition 2: Conflicting and conflicted atomic events**

Let \( ES \) and \( CE \subset ES \) denote the sets of signaled and already combined atomic event instances, respectively. The set of conflicting atomic events is defined by

\[
ES_C = \{ e_a \in ES \mid \exists e'_a \in EC : \text{type}(e_a) = \text{type}(e'_a) \land t(e_a) < t(e'_a) \land t_{ed}(e_a) > t_{ed}(e'_a) \} \subseteq ES
\]

the set of atomic events conflicted by a conflicting event \( e_a \) is defined as

\[
EC_C(e_a) = \{ e'_a \in EC \mid \text{type}(e_a) = \text{type}(e'_a) \land t(e_a) < t(e'_a) \land t_{ed}(e_a) > t_{ed}(e'_a) \}
\]

and the set of conflicted atomic events as

\[
EC_C = \bigcup_{e_a \in ES_C} EC_C(e_a)
\]

\[
\{ e'_a \in EC \mid \exists e_a \in ES_C : \text{type}(e_a) = \text{type}(e'_a) \land t(e_a) < t(e'_a) \land t_{ed}(e_a) > t_{ed}(e'_a) \} \subseteq EC
\]

Based on this definition we can formalize the set of conflicted complex events

**Definition 3: Conflicted complex events**

The set of conflicted complex events, i.e. the complex events that are affected by a conflict of atomic events, is defined as the set of complex event instances that include a conflicted atomic event as a component, i.e.:

\[
CE_C = \{ e_c \in CE \mid \exists e_a \in ES_C : \exists e'_a \in occ(e_c) : e'_a \in EC_C(e_a) \} \subseteq CE
\]

where \( occ(e_c) \) is the set of atomic events occurring as component events in a complex event \( e_c \).

All the sets defined so far are dynamic and need to be actually computed for conflict handling as described in the next subsection. Obviously, the following relation holds for complex atomic event instances:

\[
\forall e_c \in CE_C : t(e_c) = \max\{t(e'_a) \mid e'_a \in occ(e_c)\}
\]

\[
> \min\{t(e_a) \mid e_a \in ES_C\}
\]

To cope with conflicting events in an algorithmic manner we also need to determine the corresponding event types

**Definition 4: Conflicting atomic event types**

Let \( Type_a \) and \( Type_c \) denote the domains of atomic and complex event types, respectively, defined in an active database system, \( E_a \subset Type_a \) and \( E_c \subset Type_c \). We define the conflicting atomic event types as the set of types of all conflicting atomic events, i.e.:

\[
Type(ES_C) = \{ \text{type}(e_a) \in Type_a \mid e_a \in ES_C \}
\]

By applying this definition we can determine the earliest timestamp of a conflicting atomic event of a specific type. Obviously, the set of conflicting atomic event types is identical to the set of conflicting atomic event types due to condition (3) in definition 1:

\[
Type(ES_C) = \{ \text{type}(e_a) \in Type_a \mid e_a \in ES_C \}
\]

\[
= \{ \text{type}(e_a) \in Type_a \mid e_a \in EC_C \} = Type(EC_C)
\]

**Definition 5: Conflicted complex event types**

We define the conflicted complex event types as the set of types of all conflicted complex events, i.e.:

\[
Type(EC_C) = \{ \text{type}(e_c) \in Type_c \mid e_c \in EC_C \}
\]

In the following definition, we introduce the relation between conflicting atomic events and conflicted complex event types.
Definition 6: Conflicting events to conflicted event types

We define the set of conflicting atomic event types to a conflicted complex event type $E_c \in \text{Type}(CE_C)$ by

$$
\text{Type}(ESC, E_c) = \{ e_a \in \text{Type}(ESC) \mid e_a \in OCC(E_c) \}
$$

and with this the set of conflicting atomic event instances to a conflicted complex event type $E_c \in \text{Type}(CE_C)$ by

$$
ESC(E_c) = \{ e_a \in ESC \mid \text{type}(e_a) \in OCC(E_c) \}
$$

where $OCC(e_a)$ is the set of atomic event types occurring as component types in a complex event type $E_c$.

This last definition is used for a type-specific reaction of the complex event detection depending on exactly those conflicting atomic event instances and types that influence instances of the given complex event type. With these formalizations we can now describe our algorithm for event detection and conflict handling.

3.3. A New Algorithm for Event Composition

There are several approaches to handle conflicts in the event signaling. Within this work, we have decided to resolve the full set of detected conflicts in order to design a general framework for consistent event processing throughout the mobile system. Alternative solutions like ignoring or delegation of the conflict handling are more or less application-dependent. Regarding the scope of conflict handling, i.e. whether all, some or none of the conflicts are to be solved, the algorithm can easily be extended to apply for any selection criterion like lifetime of events, their timestamps, the number of dependent transactions or affected events, or event types. The proposed algorithm works by resetting the complex event detection to the timestamp of the first conflicting event $\min\{t(e_a) \mid e_a \in ESC\}$ and restarting it from there, similar to a partial transaction rollback. All conflicted events have a later timestamp.

We assume that only atomic events are signaled between different event handlers and every event handler does its own complex event detection. The algorithm shown in figure 1 specifies the detection and handling of conflicts in the event processing in accordance to the model described above. It is a set-oriented algorithm that does not necessarily be invoked for every single remotely signaled event, but is capable of handling a set of those events during each run. Thus, it enables a more flexible handling of possible conflicts, e.g. active mechanisms can be used to determine when to initiate the algorithm.

The algorithm works as follows: New remotely signaled events are included into the local event log that stores all event instances of importance to the event detection (1). From the set of remotely signaled events, the subset of conflicting atomic events $ESC$ is determined (2). If $ESC = \emptyset$, the event processing is continued with the first not yet considered event instance, otherwise the conflict handling starts by computing the set of conflicting atomic event types $\text{Type}(ESC)$ and the timestamp $t_c(E_a) := \min\{t(e_a) \mid \text{type}(e_a) = E_a \wedge e_a \in ESC\}$ of the first conflicting event for every type $E_a \in \text{Type}(ESC)$ (4). Also, the sets of conflicting atomic and complex events $EC_C$ and $CE_C$, respectively, are determined (5) as well as the set of conflicting complex event types $\text{Type}(CE_C)$ (6).

After these computations, the actual conflict resolution is called. For each $E_C \subseteq \text{Type}(CE_C)$, the set of affiliated conflicting atomic event types $\text{Type}(ESC, E_c)$ is computed (8). Then the complex event detection for each conflicting complex event type $E_c \in \text{Type}(CE_C)$ is reset to the timestamp $\min\{t(e_a) \mid E_a \in ESC(E_c)\}$ of the first atomic event conflicting with this complex event type (10). This logically includes the reconstruction of partially combined complex events. Then, the event combination is restarted for all conflicting complex event types $E_c \in \text{Type}(CE_C)$ (11). The remaining steps deal with the actual conflict handling. The reconstruction of a consistent state is carried out by considering the instances of the atomic conflicting event types (12). If the atomic events have already been combined by earlier complex events, then they are not conflicted and can be skipped (15). Event instances that have been combined by conflicted complex events are reconsidered, and the conflicted complex event is reset to the logical state at the timestamp of the atomic event, from where it is reconstructed (18). Atomic instances that have not been combined by any other instance of this conflicted complex event type can be combined without further restrictions (21). This conflict handling mechanism (9–22) is applied until the complex event detection is completed, i.e. the end of the event log is reached.

3.4. Integrating Broadcasts and Event Signaling

To deal with the mechanisms of distributed event signaling we start off by classifying the different channels for event signaling in a mobile system. Additional to the local event signaling within the components of one active DBMS, we can distinguish two basic network channels between the event managers of different database stations:

- **uplink event channel** consisting of a wireless uplink sub-channel from a mobile client to a base station using a point-to-point communication and a wired uplink sub-channel from a base station to stationary servers using a one-to-many communication protocol, e.g. multicasting or broadcasting, and

- **downlink event channel** consisting of a wired downlink sub-channel from a stationary server to the corresponding base stations using a one-to-many commu-
In the following, we concentrate on the wireless downlink sub-channel from a base station to the served mobile clients. We propose using a repeated, aperiodic broadcasting of event data by a base station to make this information available even to mobile clients that are temporarily disconnected without having to request it. We assume a logically segregated event channel using index information to enable selective tuning of a mobile client [10]. Selectivity can also be used for supporting the exactly-once-semantics of raised events. Ideally, the order criterion, alike in the event log, reflects the requirements of the event combination strategy. Applying a FIFO-strategy, the event log has to supply a priority queue of events for each event type sorted by ascending time stamps. If all event types are of the same importance, then scheduling can simply be done by ascending timestamps over all events. Alternatively, one can think of event types of different importance to an application and therefore of distinguished priorities of them for transmission. A third criterion could be a sorting by remaining lifetime of events. By way of example, we describe implications if events are ordered by their timestamp. Then, the delay of an event ε within the broadcast correlates to the timestamp \( t(ε) \): for any two event instances \( ε_1, ε_2 \in E \):

\[
\begin{align*}
t(ε_1) < t(ε_2) & \Rightarrow t_{\text{broadcast}}(ε_1) < t_{\text{broadcast}}(ε_2) 
\end{align*}
\]

for any two event instances \( ε_1, ε_2 \), where \( t_{\text{broadcast}}(ε) \) denotes the time difference from the start of the broadcast to the first occurrence of \( ε \) on the broadcast channel. As an upper bound for the duration that an event must reappear in the broadcast program we define a periodic synchronization interval for mobile clients.

### 4. Design and Realization

The model described in the previous section has been implemented to evaluate its feasibility for event processing in a mobile environment. The prototype system mimics the distributed event processing in an active mobile database system, mobile hardware or wireless network technology have not yet been employed.

#### 4.1. Architecture

The system consists of three types of computer entities: Server, MobileClient, and BaseStation. The first two classes are generalized to DBStation, which itself has composition associations with the classes EventGenerator, EntityManager, CED (complex event detector), and EventSocket for all the stages of event processing, whereas objects of the class BaseStation are assumed not to monitor or detect events themselves. They simply act as an event broker between stationary and mobile hosts, composed from an EventSocket, a Broadcast, and a BroadcastController object, of which the latter two are specializations of EventLog and EntityManager, respectively. Instances of EventSocket are used as interfaces to network links.
Figure 2 shows the class diagram of MESAVERDE in the notation of the Unified Modeling Language (UML).

4.2. Realization

The prototype system MESAVERDE was implemented in Java. Each computer object is represented by a single window as depicted for a mobile client in figure 3. Server and MobileClient objects can automatically (randomly or periodically) or interactively generate external, i.e. application events, time events, and database events. Mobile clients can also generate resource events of predefined types. Time and resource events are only signaled to the local event manager, whereas database and application events are also transmitted to other event managers in the mobile system. The Event History textbox is a simple trace of events sent to the corresponding event manager, the Event Log textbox shows the actual state of the EventLog object when updated. The Broadcast textbox shows the sequence of broadcast events and broadcast init information. Finally, the Complex Event Detector textbox visualizes detected and reset complex events. In addition to event timestamps, information reflecting events’ state regarding processing and transmission are displayed.

MESAVERDE realizes selectivity by diverse filtering mechanisms of the broadcast stream at the client side. Due to the message sending paradigm of object-oriented programming languages between individually addressed objects only, broadcasting itself is implemented by sending data to all registered clients using the Java client/server-mechanism.

Empirical results have shown that the developed model and algorithms work as expected. Repeated broadcasting can significantly reduce the probability of undelivered event instances to a mobile client. Currently, we are working on a formal evaluation of the efficiency with respect to event charateristica and broadcast parameters.

5. Conclusion and Future Work

In this paper, we have presented a new model for the processing of events in an active database system operating in a mobile environment. The newly developed model differs from traditional models in the following aspects:

- It defines the term of a conflicting event, which forms the basis for coping with disconnection.
- It uses a new algorithm for event composition, which builds on the definition of conflicting events.
- It pays attention to the fact that in a mobile environment, broadcast data can be used to allow a scalable and asymmetric data transfer.

This model is defined in a formal way to get a precise understanding of what a conflict is and how to handle it. Additionally, the formal handling of the problem has significantly eased the implementation of the prototype system MESAVERDE. It proved that the developed algorithm for the detection of complex events works as expected and has shown that introducing the concept of broadcasting for event dissemination can enhance the performance of event processing in active database systems.

Currently, we are working on an integration of the described model into an mobile and active database management system developed at the University of Oldenburg. This system builds on the AIDE–environment as described in [3]
Figure 3. User interface of a Mesaverde mobile client.

and is designed to operate as a building block for a workflow management engine [16]. For this, we hope that the event processing techniques presented in this paper can significantly ease the integration of mobile users into the workflow management system.

References


