A Unified Model of Internet Scale Alerting Services

Annika Hinze, Daniel Faensen
Institute of Computer Science
Freie Universität Berlin, Germany
{hinze,faensen}@inf.fu-berlin.de

Abstract

In the last years, alerting systems have gained strengthened attention. Several systems have been implemented. For the evaluation and cooperation of these systems, the following problems arise: The systems and their models are not compatible, and existing models are only appropriate for a subset of conceivable application domains. Due to modeling differences, a simple integration of different alerting systems is impossible. What is needed, is a unified model that covers the whole variety of alerting service applications.

This paper provides a unified model for alerting services that captures the special constraints of most application domains. The model can serve as a basis for an evaluation of alerting service implementations.

In addition to the unified model, we define a general profile structure by which clients can specify their interest. This structure is independent of underlying profile definition languages. To eliminate drawbacks of the existing non-cooperating solitary services we introduce a new technique, the Mediating Alerting Service (MediAS). It establishes the cooperation of alerting services in an hierarchical and parallel way.

1 Introduction

The number of scientific publications doubles every 10 - 15 years [Odl95]. Electronic publication becomes very popular. Since the readers do not want to be forced to regularly search for information about new documents, there is strong need for alerting services (AS). An alerting service keeps its clients informed about new documents and events they are interested in. But alerting services are not restricted to the area of scientific publications. Examples for applications that could benefit from alerting services are applications such as digital libraries, stock tickers, and traveler information systems. Currently, several implementations of alerting services already exist for the different applicational domains, such as Salamander [MJS97], Siena [Car98], Keryx [BK97a, BK97b, LRW97] or OpenCQ [LPT99, PL98, LPR98] and Conquer [LPTH99]. The underlying models of these services do not meet all requirements found in applications suitable for wide area networks, such as digital libraries. Additionally, the models for existing alerting services mainly cover the applications the services are designed for.

In this paper, we provide a *unified model for alerting services* that considers the special constraints of the different application domains. The interests of clients are defined as so-called profiles. Since several profile definition languages are used in the different services, we give a *general structure of profiles* for alerting services, independently from the profile definition language.

The large number of existing alerting services for a certain application domain has several drawbacks. The users have to define their interest at different services in different ways. The available notifications are mostly bound to the supply of individual services, information from different suppliers is not combined. We therefore introduce and propose the use of a *Mediating Alerting Service (MediAS)*, that connects several suppliers and clients.

The remainder of this paper is structured as follows: In Section 2, we provide an overview of the structure and tasks of an alerting service. In Section 3, we introduce scenarios for the conceivable application domains of alerting services and name the problems with existing alerting services in detail. Section 4 introduces our architectural model for alerting services and Section 5 outlines our event-based model. In Section 6, we propose the use of a mediating alerting service. Section 7 provides an overview of some related systems and models. Section 8 gives some directions for our future work.

2 Event Notification Service

In this section, we introduce the general structure and tasks of an alerting service. Alerting services connect suppliers of information and interested clients. In our example of scientific papers the suppliers are publishing houses and the clients are the interested scientists. Alerting services inform the clients about the occurrences of events on objects of interest. Objects of interest are located at the supplier's side. Events can be for example changes on existing objects or the creation of new objects, e. g., the publication of a journal article. Clients define their interest by personal profiles. The information about the occurring events is filtered according to these profiles and notifications are sent to the interested users (clients). Figure 1 depicts the data-flow in a high-level architecture of an alerting service. Keep in mind that the data-flow is independent from the delivery mode, such as push or pull.

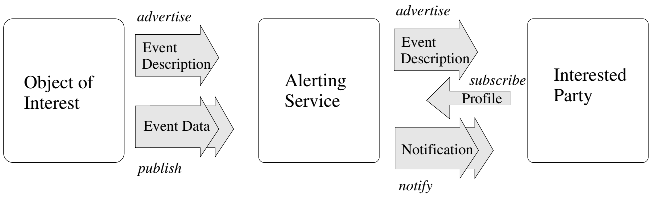

Figure 1: Data-flow in an Alerting Service

The tasks of an alerting service can be subdivided into the following steps: First, the observable event classes are to be determined and offered to the clients. Then, the client's profiles have to be defined and stored. The occurring events have to be observed and filtered. Before creating notifications, the events are integrated in order to detect combinations of events (e. g., two conferences happen to be at the same time). After duplicate recognition the messages can be buffered in order to enable efficient notification (e. g., by merging several messages into one notification). According to a given schedule, the clients have to be notified.

3 Scenarios and Arising Problems

We present a collection of possible scenarios in which Internet-scale event services are applicable. The main purposes of this collection are the motivation and validation of an event model that covers all applications. Based on the given scenarios and the derived common requirements for alerting services, we will point out the problems with existing alerting services and their models.

3.1 Scenarios

Event services are applicable in a variety of scenarios of wide area network usage. In this section, we present a selection of these applications to demonstrate the need for a unified and extended event model and a Mediating Alerting Service.

Stock ticker Selected stock values are pushed to registered clients. The clients subscribe to selected stocks. Notifications are delivered only to the subscribers. Delivery can be immediate (to paying customers) or deferred (by 20 min). Clients can be offline. In that case, notifications are lost without consequences. A client can be a PC with an analysis software that reacts on events like threshold crossing of a share value by notifying its user or by reacting autonomously.

Cardinality of the relation between supplier and client is usually $1 : n$, size of notification messages is small (a few bytes) but they are sent with high frequency. Encryption can be required. Objects of interest are identifiable in advance, the clients subscribe to objects by selecting from a given list of objects (n out of m).

Digital Library In a digital library, users want to be notified on new publications they are interested in. They define their interest by specifying certain bibliographical meta-data (e. g., a journal or an author) or by Information Retrieval-like queries. Information suppliers can be publishing houses or universities' technical report servers. The offered documents reside on publisher's side within a database, file systems, or other repositories. Within the profiles, the clients have to specify the source.

Notifications can include the full document or a pointer to it (DOI, URL). To avoid an unnecessary high frequency of notification deliveries, users can specify a time interval (e. g., weekly) within which notifications are collected and then delivered altogether. Since users do not know each supplier and do not want to register at different suppliers' interfaces, a service covering many suppliers and unified access to their repositories operates between clients and information sources. Departments or working groups have overlapping user profiles. That allows for hierarchical cooperating alerting services to ensure scalability.

Cardinality between suppliers and clients can be $n : m$, notifications can be large (several MB). Frequency of notifications is low, delivery has to be guaranteed. Objects of interest are unknown at the time of profile definition and usually come into existence later. An arising problem is the notion of composed objects: A mathematical proof is part of an article, which is part of a journal issue, which in turn is part of a journal.

Software Update Registered users of software (programs, data) automatically get updates pushed from their vendors via the Internet. To avoid too frequent delivery, users can specify that only every second update is really of interest.

While notification frequency is low, their size can become huge. It depends on previous events whether an update event is to be forwarded to the client. We call dependence of events on other events *event patterns*.

Remote monitoring and control A power station is equipped with a variety of probes and sensors. Multiple devices of the same type ensure reliability by redundant measurement. Measured values are pushed in real-time to the monitors. Monitors are displaying devices supervised by a human, or software agents that evaluate the data and react, e. g., by alerting a technician or by shutting down parts of or the whole power plant. Redundancy is not restricted to the probes and sensors. The bus for sending measurement values is multiplied. Monitoring is done at different places, several control rooms and replicated software agents. A client can request information from certain probes or probe classes. Additionally, it can require to be notified only if two or more probes deliver the same value or if a probe did not deliver new measurements for a certain time interval.

Cardinality of supplier to client can be $1 : (1 \text{ out of } m)$, that means 1 of m redundant clients has to handle en event. Reliable connections and real-time delivery is required. Notification delivery in the case of at least two related event occurrences is another example of an *event pattern*. Events that indicate that nothing happened in a time interval are called *passive events*.

Replication Services In a replication service, a DBS (or file system or Web site) has knowledge of several mirror sites. These have to be notified on any changes that occur. With the notification, a change log (e. g., transaction log) is delivered that allows the mirrors to update themselves. There is not necessarily a master. That means any "mirror" can accept local changes and notifies the other mirrors. Mirrors can define a profile to subscribe to subsets of the masters (or peer's) repository (e. g., "sports-related Web pages", "transactions on private customer accounts", or "small documents daily, larger ones only weekly"). To ensure consistency, the receiver can acknowledge the notification.

Cardinality can range from $1 : 1$ to $m : n$. Event producer and subscriber roles can switch continuously. Notifications can occur frequently (and then are small) or less frequently (bundling changes to large notifications).

Mobile Computing Portable end-user devices lack the power to compute complex tasks, e. g., the detection of certain conditions to alarm a share holder. Transfer of application logic to a server is a typical such scenario. Moreover, mobile devices cannot be online permanently. That rises (i) the need to buffer notifications and (ii) the necessity of defining complex event patterns in the profile. Mobile users of a digital library avoid automatic delivery of large documents by stating in their profile that only pointers to the objects are to be delivered.

3.2 Dimensions for Model Evaluation

From the scenarios described in the previous section, the following dimensions to classify and evaluate event models and alerting services emerge:

Cardinality Associations between suppliers and clients cover the range from $1 : 1$ to $m : n$. The Remote Monitoring and Control scenario shows that the notion of a $1 : (n \text{ out of } m)$ cardinality is useful.

Notification size Depending on application type, the size of a notification can range from a few bytes (e. g., stock ticker) to several megabytes (digital library, software update). By delivering pointers to objects instead of the objects themselves, the size can be reduced significantly.

Notification frequency Can vary from high frequent (in the range of seconds) to, say, once a year or only once at all.

Guaranteed delivery In a digital library, for instance, it is necessary to guarantee delivery of notifications even if clients are offline.

Real time Remote monitoring and control can require real time delivery of notifications.

Passive events In some cases, it is useful to be notified if during a specified interval *nothing* happened, e. g., if a server does not handle requests anymore.

Event pattern Clients register for events that depend on other events.

Composed objects Objects do not need to be atomic, but can consist of other objects (e. g., journals consists of articles).

Object repositories Clients can subscribe to repositories to get informed about the changes within that repository. To subscribe to information objects that do not exist at the time of the profile definition, clients refer to the repository the object will appear in.

Profile definition Clients can subscribe to concrete objects (e. g., by referring to their identifier), by specifying meta-data that describe the objects of interests or (in the case of digital libraries) using an IR-like query.

Scalability Can be achieved by redundant alerting services (or duplicated parts of them). If profiles of different clients are overlapping, a hierarchy of cooperating alerting services can improve scalability.

Encryption Scenarios that cover delivery of privacy data or data that are liable for costs can require encrypted delivery. Encryption is handled on protocol level.

Reliability and Acknowledgment In the case of remote monitoring and control, reliable connections are required. Acknowledgments can be used to implement reliable delivery. These characteristics will not be considered in our model, since they have no influence within the modeling level used here.

In the following part, we show the drawbacks of the existing models for alerting services in covering the requirements derived from the different scenarios.

1. **Terminology:** On the one hand there exist several names for this kind of service (Alerting Service , Notification Service, Profile Service, etc.), while on the other hand several different concepts are called notification service (see Section 7).

   Additionally, the different models for alerting systems use identical terms to describe different concepts. For example consider the term *Channel*: In the CORBA model, an event channel is an intervening object that allows multiple suppliers to communicate with multiple consumers asynchronously [OMG97].

CDF [Ell97] or Netcaster Channel [Net] are similar to television broadcast channels. In contrast to CORBA, a CDF-Channel has an observer function for the channel objects. Further evaluation of the implementation of alerting services with channels can be found in [FHS98].

2. **States of non-existing objects:** In most event-based models, an event is defined as *a state transition of an object of interest at a particular time*, where the state of an object is the current value of all its attributes (e. g., [OMG97, KR95]). Other definitions refer more to the observation of events and therefore identify events by their physical representation as messages (e. g., [Car98, TIB]). Here, we tend to the first approach, as it is more complete and also covers the existence of unobserved events and is therefore the superordinate case. The binding of events to the object of interest cannot be weakened in general as the events are strongly related to the objects (opposite to clock-time events).
   Consider the case of a scientific paper or article that is published. This object (publication) appears at a specific time, the state of the object is then the content of the paper and its meta-data such as author and title. But what is the state of the object before it exists? So the terms of an event as a state transition of the object is not appropriate here. Rosenblum and Wolf [RW97] define an event as an *instantaneous effect of the termination of an invocation of an operation on an object*. This definition associates the event with the invoker of the operation instead of with the object of interest. As a consequence, the invoker has to communicate with the observer in order to announce the event. It cannot be generally presupposed that invokers actively announce events to observers, due to several reasons, e. g., they are not known to each other or suppliers of documents refuse to support the observer.

3. **Composed objects:** The notifications sent to the clients are the messages that are seen as the physical representation of the events [Car98] that the clients are subscribed to. Since the events relate to identifiable objects of interest, the notifications contain or refer to these objects. (Example: The client subscribes to all articles by author X, the publishing house publishes a journal (the object) that contains an article by X, the client gets the journal, or rather the information about the journal.) However, clients are often not interested in whole documents or sites (they are interested in an article instead of the whole journal, or even in a single mathematical proof instead of the article), therefore, substructures need to be identifiable as objects.

4. **How to register that nothing happened:** Example: “Send message if the value of share S does not change for a period of days” (see Section 3.1). Existing models of alerting services cannot handle this kind of profile, as neither is an operation performed on the object of interest, nor does the object of interest change its state. We are aware of the fact that this construct is contradictory to the intuitive notion of an event as *something that happens*.

4 Architectural Model

In this section we introduce a general architectural model for alerting services that can be applied for existing implementations and is used for the identification of components involved in the event model presented in Section 5. It serves as a basis for the development of MediAS, the Mediating Alerting Service that notifies users of a digital library of electronically available scientific publications from different suppliers. A diagram that shows the involved components and their relations is shown in Figure 2.

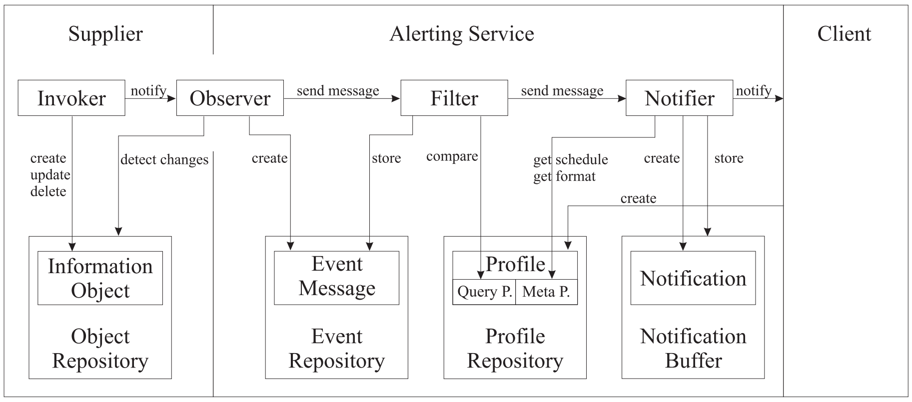

Figure 2: Architectural Model of an Alerting Service

Objects of interest are so-called *information objects* that are located at the supplier's side, optionally in an *object repository*. Information objects can be persistent (e. g., documents) or transient (e. g., measured values). The objects can be organized hierarchically.

Changes of theses objects (creation, update, deletion) are induced by an *invoker*. Responsibility of the *observer* is the detection of changes of single objects or in the *object repositories*. Change detection can be an active task of the observer, performed periodically, if the invoker does not inform the observer by itself. Any change is an event. A detailed definition of the term event will be given in Section 5.1.

Events are reported as materialized *event messages* to the *filter*. The filter has knowledge of the client's profiles and compares the event with the query part of the profiles. If a profile and event match, the filter creates an *event message* and delivers it to the *notifier*. For the detection of *event patterns*, events are stored in the *event repository*.

The *notifier* in turn checks the schedule part of the *profile*. If immediate delivery is demanded, the event message is edited according to the format specified by the client and delivered. Otherwise, it is buffered until the notifications become due. The notifier keeps track of the due-dates. The buffering of notifications is also needed for the notification of offline clients to guarantee delivery.

Client's *profiles* consist of two parts (see Section 5.1). The *query-profile* (used by the filter) specifies the set of information objects the client is interested in. In the *meta-profile*, a *schedule*, a notification *protocol*, and a notification *format* (e. g., for data type conversion) are defined. Schedule, protocol and format are attributes used by the notifier.

The components of the alerting service can be (and usually are) deployed and duplicated for scalability and reliability. Invoker and object repository usually reside at the supplier's side. Not all information suppliers implement an observer; an alerting service that covers this type of suppliers could implement an observer as a wrapper for each supplier. The observer is enforced to keep various information on the repository (e. g., previous states) if it is not notified by the invoker and the supplier's interface does not offer a search for changes since a specified date. Alternatively, the observer can be moved to the supplier's side (if allowed) and perform its tasks as an agent of the alerting service there.

5 Unified Event Model

In this section, we introduce our event-based model for alerting services. First, we define the terms used within the model, then we formally describe the tasks of a MediAS.

5.1 Terms and Definitions

Object: In correspondence to other models, we use objects to encapsulate the functionality of model participants. In our model, an *object* can be any logical entity residing on a hardware component within a network, such as files and processes. Hardware and human beings can also participate but are represented by their software-based proxies. Each object is uniquely identifiable, for simplicity reasons, we refer to the identifier as a handle (already used in [CGM97]). A handle can be, e. g., a URL or a DOI. Considerations of a naming model for alerting services can be found in [RW97].
The objects that are offered by *suppliers*, such as journals, news-pages, or movies, we call *objects of interest*. Objects have a *state*. The state of an object is the value of its attributes. A set of objects offered by a supplier is referred to as *repository*. A supplier can offer one or more repositories, examples are databases, web-sites, or a set of documents on an ftp-server. Since repositories can also be seen as objects of interest, we consider a hierarchy of objects, whereas the items within the repositories are called information objects. Information objects can also be composed of other objects, e. g., journals consist of articles. Therefore, objects need to carry information about their position within the hierarchy.

Event: Based on the scenarios, we get a set of possible events regarding information objects: A new information object appears; existing information objects are changed; existing information objects are deleted; for a certain interval of time the information object remains unchanged.

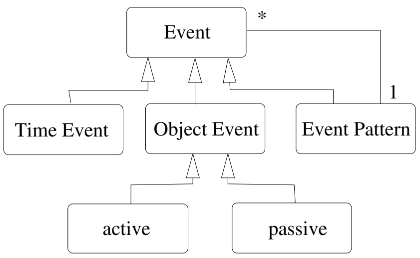

Figure 3: Event Types

Similar to the model used for Event Action Systems [KR95] we divide events into two classes: *time events* and *object events* (see Figure 3). Time events involve clock times, dates, and time intervals. Object events involve changes of non-temporal objects. We additionally distinguish *active* and *passive events*. Active events are state transitions of the repository at a particular time; they are observer independent. State transitions can be actions such as insertion, deletion, or change of a data-object. In the context of databases, as in digital libraries, the notion of state transition is in close relation to integrity constraints. Each transaction on the database underlies several constraints (e. g., the key of a tuple has to be unique) and the operations are accepted only if the constraints are fulfilled. Allowed operations transfer the database from a valid state to another valid state. This process is called a *state transition*. As final consequence, both constraints and given set of values ensure an invariant state space. A state of a particular information object is a defined attribute of an object that has a finite number of predefined values (e. g., the output of a logic circuit can be high, low, or high-Z, an interrupt-flag can be set or not). A state transition occurs if the attribute value is changed. Passive events involve counters and object properties at a specified time. They have to be observed. Passive events model the fact that for a given time interval an object did not change. Examples are also given in Section 3.

Profile: A profile is the formally defined information need of a client. Each profile consists of two parts, the description of the events the client wants to get notified about (*query-profile*) and the conditions for the notification (*meta-profile*).

Within the *query-profile*, the clients specify the events they are interested in.

For time events, we can distinguish between events given as points (t_e), as intervals ($[t_{start}, \infty)$, $(\infty, t_{end}]$, or $[t_{start}, t_{end}]$) or as frequencies (e. g., weekly). Here the events are given as absolute values in time. These points, intervals, or frequencies can also be given in relation to another event, e. g., "X weeks after the conference" (relative). Other time events can be formed by using combinations. Relative time events and combinations can be seen as forms of event patterns, which are described later. For time events, client and server need to define a reference (e. g., a common time zone).

For the object events, clients have to define the objects, the attribute values they are interested in and the state transistion to observe on the object. Additionally, the repository of the objects has to be defined (by giving the repository identity in the profile or indirectly by subscribing to services from special suppliers). The definition of the object repository is independent from the object itself since the same object can reside as duplicate on different repositories. Identification of objects can be done by giving

1. the object handles,
2. metadata about the objects,
3. values of attributes of the objects.

For example, if clients define their interest by giving similar objects ("Notify about all objects similar to THIS one"), this can be seen as a handling of (2) and/or (3). The subscription to subject-based Internet-channels is covered by (2), subscription to Internet-sites (favorites) refers to (1). For composed objects, the level of the object and the concerned attributes within the hierarchy should also be given (e.g. consider the journal-example).

For active object events, clients have to define the objects as described above, further attribute values they are interested in and the type of state transition to be observed on the object. Possible state transitions are the occurrence of a new object, the change of an object, or the removal of an object. The change of an object can concern the structure (changing set of attributes, changing range of attributes), or the values of the attributes. Additionally, clients can be interested in the number of values, if it changes or in different changes of the value itself. Conceivable are values that change from point or interval to point or interval (e. g., "Notify, if value X is no longer in $[y, z]$", change from interval to interval).

Passive events need to specify the objects and their attributes (see active events), and constraints as time or counters.

In addition to primitive events (time and object events), clients can specify event patterns. Event patterns are combinations of events. A pattern can include any number of events combined with unary, binary, n-ary-operators. Conceivable operators are, for instance, the sequence operator or Boolean operators in combination with time constrains. The sequence operator reflects the temporal order of the events (e.g. implemented in Siena [Car98]).

Examples of query-profiles are:

- *"Notify, when in an issue of journal X an article about topic Z is published":*

  object event, object identification by metadata, state transition: new object appears),

- *"Notify, when in database X the value of attribute Y is larger than Z":*

  object event, object identification by handle, state transition: change of object, change: attribute value from interval to interval),

- *"Notify, if the temperature in room X is constant for the time period Y":*

  passive event, object identification by handle

- *"Notify, if file X is changed after message Y appeared":*

  event pattern of two object events, sequence operator, object identification by handle

- *"Notify, if two articles with title X appear in different journals":*

  event pattern of two object events, Boolean AND with unlimited time period, object identification by meta-data

The following items have to be defined within a *meta-profile*:

1. Content of notification (e. g., object-handle, object itself, meta-data describing the object/event and/or their number),
2. Structure of notification (number of events reported in the message, instructions for the merging of notifications, ranking mechanism),
3. Notification-protocol (e. g. e-mail, desktop-icon, download),
4. Time-policy for event detection (frequency of observations),
5. Time-policy for notification such as scheduled (e. g., daily) or event-dependent (e. g., on n events or depending on event-attributes, such as "X weeks before the conference").

Observer: Observers detect events. The event observation can be triggered by the invoker, by a time policy for observation or by a profile (for passive events). Observers can be part of the alerting service or reside on suppliers side. A MediAS can be distributed so that it employs several observers.

Notification: A notification is a message reporting about events. Clients are notified according to the time-policy given in their profiles. Notifications created by observers have to be evaluated to discover patterns or duplicates. Before sending notifications, depending on the profile, they have to be edited (e. g., duplicate removal, merging, formating) in order to ensure that clients get only one notification at the time.

5.2 Model

An alert relation describes a connection between repositories of information objects, clients, and events the client wants to get notified about. A supplier can provide various repositories. Suppliers inform clients about possible events by means of advertisements. Clients define their interest using profiles, where they specify the events on the repositories they are interested in.
The *alert relation* A is a relation of a set of *repositories* R, a set of *clients* C, and a set of *profiles* P (We employ the notion of the relational algebra.):

$$A \subset R \times C \times P. \tag{1}$$

Clients may *register* or *unregister* for certain events happening on certain repositories, for instance by subscribing to an event channel, such as CDF-channels. The corresponding tuples $\{[r, c, p]\}$, where r denotes the repository, c the client, and p the profile regarding an event on r, are inserted into, or deleted from, the relation. An event channel E is a projection of the alert relation onto the repositories and the profiles:

$$E = \Pi_{R,P}(A). \tag{2}$$

Within a projection duplicate tuples, are deleted. According to this model, a CORBA channel is only a communication medium and not covered by the definition of a channel given above. Another conclusion is, that clients that are interested in the same events at the same repository are subscribed for the same event channel, as in CDF-channels or mailing lists.

Events can happen at any time, they are caused by invokers that perform actions on the repository. Let t_e be the time a state transition e in repository r has been caused by an invoker. An observer can learn of events in two ways. Either an observer is notified by the invoker at time $t^{obs} \geq t_e$. Or the observer proceeds according to a time schedule

$$T^{obs} = \{t_i | t_i = t_0 + i\delta_{obs}, i \in \mathbb{N}_0, \delta_{obs} > 0\}. \tag{3}$$

The observer registers all state changes on r in the interval $(t_{i-1}, t_i]$, $t_{i-1}, t_i \in T^{obs}$, at time $t^{obs} = t_i$. The observer creates a message reporting the occurrence of the event and forwards it to the filter (see architecture, Section 4). The profile filter compares the reported events to the profiles. With Carzaniga [Car98], $p \sqsubset e$ denotes that an event e matches a profile p. Notifications to the following clients must be produced:

$$C_{i-1,i}(r, e) = \Pi_C(A_{i-1,i}(r, e)), \tag{4}$$

where

$$A_{i-1,i}(r, e) = \sigma_{(A.R=r)\wedge(A.P \sqsubset e)}(A) \tag{5}$$

is the set of alert relations affected by events e in r at $t_e \in (t_{i-1}, t_i]$.

Passive events have no invoker, they can just be recognized by the filter. A profile for a passive event consists of the object description and the interval during which the object should not change. Let $O_{i,j}$ be the set of objects in r that did not change in the time interval $(t_i, t_j]$, $t_i, t_j \in T^{obs}$, $i < j$. Let p be the profile that triggers a filter by its time-policy in $(t_i, t_j]$. Assume that $O_{i,j}(p) \subset O_{i,j}$ is the set of objects that is covered by the passive event specification in p. The clients that must be notified are $C_{i,j}(r, p) = \Pi_C(A_{i,j}(r, p))$, where $A_{i,j}(r, p) = \sigma_{(A.R=r)}(A \ltimes O_{i,j}(p))$.

Objects can be composed of other objects, e. g., journals that consist of articles. The matching of the compound objects can require further data inquiries or calculations. For example, databases of digital libraries often do not contain the full-text of journals but references to the suppliers (e. g., the publishing houses) and, therefore, need to query the suppliers database.

Notifications for the clients are buffered. Before the delivery, they are checked for duplicates and then merged and delivered according to the client's profile.

For a scheduled profile, clients want to get notified according to a schedule

$$T^{not} = \{t_k | t_k = t_0 + k\delta_{not}, k \in \mathbb{N}_0, \delta_{not} > 0\}, \tag{6}$$

where $\delta_{not} \geq \delta_{obs}$. The time t_0 here denotes generally a start point in time, it can be different for different profiles and observers. The condition for the cycle durations ensures that notifications are sent only after the observer noticed events. (Otherwise clients could be misled by the impression that no event happen if only the observer did not recognize it by that time.) The events that happened in $(t_i, t_j]$, $t_i, t_j \in T^{obs}$, $i < j$ affect a set of channels $E_{k-1,k}$, with $t_{k-1}, t_k \in T^{not}$, where $t_{k-1} < t_j \leq t_k$, $t_{k-2} \leq t_i < t_{k-1}$, and therefore $E_{k-1,k} = \Pi_{R,P}(A_{i,j})$.

Since this asynchronous approach can lead to a maximal notification delay of $\delta_{not} + \delta_{obs}$, it is recommendable to synchronize observation and notification, so that $\delta_{not} = n\delta_{obs}, n \in \mathbb{N}$ and $\exists t_i \in T^{obs}, t_k \in T^{not}$ so that $t_i = t_k$ (i.e. that enshures that the phase shift equals zero). Immediate delivery is performed by $n = 1$.

Notifications on events are sent at the time the notifier gets the event messages. In notifications on n events, the notifications are merged and then delivered. In notifications on the n^{th} event, the first $(n-1)$ notifications are omitted. Notifications depending on events are handled similarly. The time of notification is triggered by attribute values of the object ("Send notifications X weeks before the conference Y starts"). In the *push model* of alerting, the supplier is the active part. It has to inform all customers that have registered for event channels in which the supplier's repositories participates. As described in Section 2, an alerting service can act as supplier. In the *pull model*, the customers regularly check the event channels for which they are registered.

Several conclusions can be drawn from the model:

- The observers either have to be informed by the invokers or they have to be aware of the state of the repository (and the objects contained in the repository). In the latter case, observers, therefore, need to be initialized with the states of all existing objects in the repository. They can only detect changes with frequencies less or equal to the observation frequency. Otherwise, observers can miss events within their observation interval. If two events regarding the same object happen within the same observation interval, these events can weaken or intensify each other. For example, if the events that a value of an attribute increases and decreases happen shortly one after another, these two events neutralize each other (e.g. a stock ticker with to low observation frequency would miss some stock value changes).

- For passive events, filters initially have to know about the existence of objects, and, therefore, they have to be initialized with all existing objects.

- For the recognition of changed objects, different update strategies have to be considered [CGM97]. *Versioning:* The old object remains, as a previous version. The handle of the new object is different from the one of the old object. *In-Place-Update:* The old object is removed, the handle of the new object is unchanged. *Shadow Updates:* The old object is removed, and the new object has a new handle.

5.3 Example

In this section, we show exemplary the application of the model to an alerting service for digital libraries (see Section 3). The repositories for our service are the journal repositories $r1$ and $r2$ at the servers of two publishing houses. Clients are students and research assistants at the Freie Universität Berlin, who define the journal articles they are interested in in their profiles. An instance of the alert relation A is shown in the Table 1. The instances of event channels are then [$r2$, "articles in journal X"], [$r2$, "articles by author Gray"], and [$r1$, "articles by author Brown"].

R	C	P
$r2$	faensen	"articles by author Gray"
$r2$	faensen	"articles in journal X"
$r1$	hinze	"articles by author Brown"
$r2$	hinze	"articles in journal X"

Table 1: Instance of an alert relation

Let us assume, that the observer is scheduled daily, starting at 1/1/99, 12:00am . The profiles are scheduled weekly, starting 1/1/99, 12:00am synchronous to the observer. The parameters for the alerting service are then:

$\delta_{obs} = 24h$

$t_0 = 1/1/99;12{:}00\text{am}, t_0 \in T^{obs}$

$\delta_{not} = 168h$

$t_0 = 1/1/99;12{:}00\text{am}, t_0 \in T^{not}$

$n = \frac{\delta_{not}}{\delta_{obs}} = 7.$

Let us consider the following event: On t_e =1/2/99, 7:00am, an issue of journal X (=information object) is published at $r2$, the journal contains, besides others, an article by Gray. The event is observed the same day ($t^{obs} = t_1$ =1/2/99, 12:00am) and reported to the filter. We therefore define t_0 =1/1/99; 12:00am and t_1 =1/2/99; 12:00am. The affected alert relation is $A_{0,1}$=$\{$[$r2$, faensen, "articles by author Gray"], [$r2$, faensen, "articles in journal X"], [$r2$, hinze, "articles in journal X"]$\}$.

The clients that have to be notified are $C_{0,1} = \Pi_C A_{0,1}$=$\{$[faensen],[hinze]$\}$. They are then notified on $t^{not} = t_1$ =1/8/99; 12:00am, so the delay is $t_e - t^{not} = 149h$, the maximal possible delay with this configuration is $\delta_{obs} + \delta_{not} = 192h$. Client Faensen gets the whole journal with all articles, the article by author Gray has been eliminated by the duplicate handling component (This can be specified in the meta-profile). The journal is a composed object that consists of objects at different levels. Since she defined the filter for articles, client Hinze does not get the whole composed object but the article.

6 Mediating Alerting Service

In this section, we motivate and propose the use of a mediating alerting service. Consider the application domain of a digital library. For full coverage of scientific publications, users have to register at a variety of suppliers ($n : m$ cardinality), some even unknown to the user, with different interfaces for profile definition. Notification format and protocol are heterogeneous, and many suppliers do not offer an alerting service. In addition the notifications of different providers cannot be combined.

As solution to the problems mentioned above and pointed out in previous sections, and as an implementation of both the architectural and the event model, we propose a *Mediating Alerting Service (MediAS)*.

In MediAS, multiple alerting services cooperate in a hierarchical and parallel manner. That means that alerting services are clients (or suppliers) for other alerting services. An alerting service can also cover multiple suppliers employing multiple observers. In turn, one observer can deliver event messages to multiple filters.

MediAS integrates the view to the information suppliers. Its *observers* serve as wrappers for the suppliers' interfaces. For suppliers that implement their own alerting services the observer acts as a *client* and is notified of all changes in the suppliers' repositories. Other suppliers are queried regularly by the observer that wraps the suppliers' interfaces.

Profiles are stored in MediAS' repository. If profiles of different clients are overlapping (as it is expected in a digital library environment, e. g., profiles owned by library users of a working group or university department), several instances of MediAS can cooperate hierarchically, which improves scalability. A MediAS which is the client of a different MediAS submits an integration of its profiles to its supplier. That ensures that the client MediAS will be notified of all information objects its clients are interested in.

Problems arise when an alerting service is notified by other alerting services whose coverages are overlapping. It can happen that notifications of the same event are delivered more than once. The MediAS must take this into account and filter out duplicate events. A similar problem can be caused by observers covering the same repository or suppliers providing similar objects (e. g., two digital libraries that offer the same journals).

In addition to the problems that are well examined in the area of non-sequential processing when having parallel tasks, the duplication of the event repository forces replication to ensure reliable detection of event patterns.

7 Related Work

Event services and *event action systems* are related systems to be considered for the definition of model and architecture. *Event services* support the asynchronous message exchange between objects. They depend on an event-based infrastructure (such as JEDI [CDF98]). Examples are event notification services (such as SIENA [CRW98, CDRW98, Car98]), SIFT [YGM95], the CORBA Event Service [OMG97] or the TINA Notification Service [Con96]. Alerting Services can be built upon event services. This is especially useful in the handling of asynchronous events. Other examples for event-based infrastructures underly remote monitoring and control systems [HICD+98]; event-based communication is also used in distributed programming [SCT95] and distributed systems [MSS97].

Event action systems are software systems in which events occurring in the environment of the system trigger actions. The reaction is performed according to some action specification defined by the users. The triggered actions may generate other events, which trigger other actions, and so on. The actions are, in contrast to common event-based infrastructures, relevant to human beings rather than notifications to other software components [Car98]. The alerting services considered here are closer to event-action systems than to common event-based infrastructures. An example of an event-action system is Yeast [KR95]. Yeast is a client-server system in which distributed clients register event-action specifications with a centralized server, which performs event detection and specification management. Each specification consists of an event pattern that is of interest to the client's application plus an action that is to be executed in response to the occurrence of an associated event pattern.

There is a class of infrastructures that we do not consider here. These infrastructures do not realize a notification service, though they are published as such, e. g., the Java Distributed Event Specification. An investigation of these frameworks can be found, e. g., in [Car98]. Also, we do not consider local event-based procedures in operating systems, such as IPCs and interrupt handling.

Active databases [CM93, CM94, GJS92a, GJS92b, GJS93] can serve as triggers for alerting services. The underlying model is based on ECA-rules (Event-Condition-Action), that can also be applied to alerting services, where the conditions are defined in the profiles and the action is sending the notification. (The notion of events used in active databases can be seen as a subset of the event classes described in our unified model.) Here, active databases are of limited use, they cannot implement a complex alerting service. They cover the repository, the observer, and the filter facility, but the complex handling of scheduled notification profiles are beyond the means of active databases. Alerting Services can be implemented based on active databases.

A system-independent design framework for scalable event notification services has been proposed by Rosenblum and Wolf [RW97]. We have shown the restrictions of this model in Section 3. However, due to its independence of specialized applications, it serves as a basis for our model for alerting services.

In the context of the DBIS (Dissemination Based Information System) framework by Franklin and Zdonik [FZ97], an alerting service like MediAS is an *Information Broker*, that acquires information from *Data Sources*, add value and distribute the information to *Clients* (= net consumers of information).

Salamander [MJS97] is a wide-area network dissemination substrate to support push-based applications. It supports a publish/subscribe paradigm where applications (*invokers* in our model) publish data attributed with string-based meta-data. Clients subscribe to data by supplying a query *(profile)* that consists of similar string attributes. A *Channel* in the notion of Salamander is built by matching the string attributes of suppliers and clients.

The Simple Digital Library Interoperability Protocol (SDLIP) from Stanford University [SDL] is designed for distributed information retrieval in a digital library environment. It supports making suppliers' heterogeneity transparent to the clients by mapping queries to their interface via wrappers. As query results are not necessarily available immediately, SDLIP offers a way to deliver objects asynchronously *peu à peu*. Alerting on new or changed objects is not focus of SDLIP. However, when starting an asynchronous search, a client can specify the address of an object that implements an interface which will receive the delivery. The server would submit new documents to that "receiver". One only needs to make the query long-lasting. In our terminology such a query is a *query-profile*.

8 Conclusion and Outlook

In this paper, a general structure and architecture for an integrative and scalable alerting service has been defined. We proposed a general event-based model for alerting services. Moreover the paper introduced a general structure for profile definitions for notification services and motivated the use of a Mediating Alerting Service (MediAS).

1. *Expanded event-based model:* Based on the requirements of scenarios for alerting services, we introduced an expanded model for Internet-scale alerting services. The model covers various scenarios, whereas existing models for notification services apply only for a restricted range of scenarios (as shown in Section 3). The model offers a general terminology for the description of alerting service components. It extends the notion of events to cover states of non-existing objects. In addition, it introduces the notion of composed objects and the notion of passive events. It can be uses as a basis for the comparison of different systems.

2. *General profile structure:* In this paper, we defined the general structure of profiles independently from the profile definition language. Several languages and protocols have been proposed [BK97b, SDL]. One of our next steps will be the definition of a general profile definition language for bibliographic alerting services.

3. *MediAS:* We proposed the use of a mediating alerting service to overcome the drawbacks of the various existing implementations. Disadvantages of existing alerting services are, among others, the need for the clients to define their profile many times in many different ways, duplicate notifications cannot be avoided, and event patterns can be recognized only in a very restricted manner. In a MediAS, parts of the service can work distributed in hierarchical or parallel order. Further research will support the decision which part of the service should be duplicated and in which order the cooperating services are to be arranged. The merging of the differently ranked results is another question we are investigating currently. MediAS is under development at the Freie Universität Berlin.

This paper resulted from the first phase of the Hermes project, that is part of the German Digital Library Project of the German Federal Ministry for Education and Research (BMBF) Global Info. The aim of the project is the creation of an infrastructure that supports the user in the paradigm shift from retrieval and browsing to profile-filtered notification.

References

[BK97a] S. Brandt and A. Kristensen. Keryx: Internet notification service for dynamic web applications. (slide presentation), presented to W3C available at `http://keryxsoft.hpl.hp.com/w3c/`, September 1997.

[BK97b] S. Brandt and A. Kristensen. Web push as an Internet notification service. accompanying paper to a talk held at W3C Workshop on Push Technology, available at `http://keryxsoft.hpl.hp.com/doc/ins.html`, September 1997.

[Car98] A. Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, Milano, Italy, December 1998.

[CDF98] G. Cugola, E. Di Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *Proceedings of the 20th International Conference On Software Engineering (ICSE98)*, Kyoto, Japan, April 1998. available at `http://ftp.elet.polimi.it/people/cugola/abstract.html`.

[CDRW98] A. Carzaniga, E. Di Nitto, D. S. Rosenblum, and A. L. Wolf. Issues in supporting event-based architectural styles. In *3^{rd} International Software Architecture Workshop*, Orlando, FL. USA, November 1998.

[CGM97] A. Crespo and H. García-Molina. Awareness services for digital libraries. In C. Peters and C. Thanos, editors, *Research and Advanced Technology for Digital Libraries. First European Conference, ECDL '97, Pisa, Italy, 1-3 September*, volume 1324 of *Lecture Notes in Computer Science*, pages 147–171. Springer, 1997.

[CM93] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. Technical Report UF-CIS-TR-93-007, University of Florida, Gainesville, Department of Computer and Information Sciences, March 1993.

[CM94] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Knowledge and Data Engineering Journal*, 14:1–26, 1994.

[Con96] Telecommunications Information Networking Architecture (T.I.N.A) Consortium. Tina notification service description. available at `ftp://ftp.omg.org/pub/docs/telecom/96-07-02.ps`, July 1996.

[CRW98] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design of a scalable event notification service: Interface and architecture. Technical Report CU-CS-863-98, Department of Computer Science, University of Colorado, August 1998.

[Ell97] C. Ellermann. Channel Definition Format (CDF). Technical report, W3C, Microsoft, 1997. submitted to the W3C on 09 March 97, available at `http://w3.org/tr/NOTE-CDFsubmit.html`.

[FHS98] D. Faensen, A. Hinze, and H. Schweppe. Alerting in a digital library environment – do channels meet the requirements? In C. Peters and C. Thanos, editors, *Research and Advanced Technology for Digital Libraries. Second European Conference, ECDL '98, Heraklion, Greek*, number 1513 in Lecture Notes in Computer Science, pages 643–644. Springer Verlag, 1998.

[FZ97] M. Franklin and S. Zdonik. A framework for scalable dissemination-based systems. In *Proceedings of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA '97)*, volume 32 of *SIGPLAN Notices*, Atlanta, Georgia, Oct 5-9 1997. available at `http://www.cs.umd.edu/projects/bdisk/oopsla97.ps`.

[GJS92a] N. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases. In *Proceedings of the 18th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Vancouver*, August 1992.

[GJS92b] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Event specification in an active object-oriented database. In Michael Stonebraker, editor, *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2-5, 1992*, pages 81–90. ACM Press, 1992.

[GJS93] N. Gehani, H. V. Jagadish, and O. Shmueli. COMPOSE: A system for composite specification and detection. *Lecture Notes in Computer Science*, 759, 1993.

[HICD+98] E. N. Hanson, C. I-Cheng, R. Dastur, K. Engel, V. Ramaswamy, W. Tan, and C. Xu. A flexible and recoverable client/server database event notification system. *The VLDB Journal*, 7:12–24, 1998.

[KR95] B. Krishnamurthy and D. S. Rosenblum. Yeast: A general purpose event-action system. *Transactions on Software Engineering*, 21(10), October 1995.

[LPR98] L. Liu, C. Pu, and K. Richine. Distributed query scheduling service: An architecture and its implementation. *International Journal of Cooperative Information Systems (IJCIS)*, 7(2&3), 1998.

[LPT99] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, (Special issue on Web Technologies), January 1999.

[LPTH99] L. Liu, C. Pu, W. Tang, and W. Han. Conquer: A Continual Query System for Update Monitoring in the WWW. *International Journal of Computer Systems, Science and Engineering*, (Special issue on Web semantics), 1999.

[LRW97] C. Low, J. Randell, and M. Wray. Self-Describing Data Representation (SDR). internet draft (work in progress), Hewlett Packard Laboratories, October 1997. available at `http://keryxsoft.hpl.hp.com/documents/sdr/draft-low-sdr-00.txt`.

[MJS97] G. R. Malan, F. Jahanian, and S. Subramanian. Salamander: A push-based distribution substrate for internet applications. In *USENIX Symposium on Internet Technologies and Systems, Monterey, California, December 8-11, 1997*, volume 32, 1997. available at `http://www.eecs.umich.edu/~rmalan/publications/mjsUsits97.ps.gz`.

[MSS97] M. Mansouri-Samani and M. Sloman. Gem: A generalised event monitoring language for distributed systems. *IEE/IOP/BSC Distributed Engineering Journal*, 4(2), Feb 1997.

[Net] Netscape Netcaster. `http://developer.netscape.com/software/netcast.html`.

[Odl95] A. M. Odlyzko. Tragic loss or good riddance? The impending demise of traditional scholary journals. *International Journal of Human-Computer Studies*, 42:71–122, 1995.

[OMG97] OMG. *CORBAservices: Common Object Services Specification*. Object Management Group, November 1997. available at `http://www.omg.org/corba/sectran1.htm`.

[PL98] C. Pu and L. Liu. Update Monitoring: The CQ project. In *Proceedings of the 2nd International Conference on Worldwide Computing and Its Applications - WWCA'98,Tsukuba, Japan*, volume 1368 of *Lecture Notes in Computer Science*, pages 396–411, 1998.

[RW97] D. S. Rosenblum and A. L. Wolf. A design framework for internet-scale event observation and notification. In Springer, editor, *Proceedings of the 6th European Software Engineering Conference*, volume 1301 of *Lecture Notes in Computer Science*, pages 344–360, Berlin, 1997. available at `http://www.cs.colorado.edu/users/alw/AvailablePubs.html`.

[SCT95] G. Starovic, V. Cahill, and B. Tangney. An event-based object model for distributed programming. In *Proceedings of the Workshop on Simulation and Interaction in Virtual Environments*, pages 172–177, Iowa, 1995.

[SDL] The Simple Digital Library Interoperability Protocol (SDLIP). available at `http://www-diglib.stanford.edu:8080/~testbed/doc/SDLIP/sdlip.htm`.

[TIB] TIBCO. `http://www.tibco.com/`.

[YGM95] T. W. Yan and H. García-Molina. SIFT - a tool for wide-area information dissemination. In *USENIX 1995 Technical Conference on UNIX and Advanced Computing Systems, Conference Proceedings*, pages 177–186, New Orleans, Louisiana, January 1995. USENIX Association, Berkeley, CA, USA.