Chapter 3

A Model of Event Notification Services

The sciences do not try to explain,
they hardly even try to interpret,
they mainly make models.
John von Neuman

In the previous chapter, we identified requirements for future event notification systems; these requirements are used in the next chapter as criteria to analyze existing event notification systems. In this chapter, we set the ground for this analysis by defining a model for event notification services. The model supports a consistent functional analysis of differently structured event notification systems by (1) providing a general terminology for ENS, (2) identifying components of ENS, and (3) specifying the event notification process independently of a specific implementation.

We start the chapter by examining the thesis' central term of *event notification service*. We review related terminologies, their definitions and usage, and discuss the distinction to our notion of ENS. Then, our requirements regarding a model for event notification services are discussed briefly. We name the limitations of existing models[1] for ENS and, consequentially, propose a new model. Our model consists of a reference model, an event model, and an event process model. The reference model identifies the main parts of an event notification service and describes their possible cooperation. In the event model,

[1]In analyzing the limitations of models, we partly anticipate the evaluation of ENS in the next chapter. Here, we concentrate on the aspect of how to conceptually model an ENS. In the next chapter, the systems architectures and implementations are evaluated based on the theoretical foundation presented in the current chapter.

we define the entities in which the service performs actions, e.g., event and profile. In the event process model, the tasks of an ENS are described in greater detail; the event-based interactions of the ENS parts are illustrated in an event notification sequence. Different communication strategies for observation and notification are distinguished. Finally, we give a summary of the chapter.

3.1 Notion of Event Notification Services

The term 'event notification service' is not clearly defined within the research literature. A survey [RK98a] of over two hundred related publications and systems failed to elicit a globally consistent definition of *event notification services*. Therefore, we propose our own definition for this central term of our work. We distinguish the notions of event notification service and event notification system:

Definition 3.1 (Event Notification Service)
An event notification service is a service that connects mutually anonymous parties. The service acquires, filters, and delivers information about events.

Definition 3.2 (Event Notification System)
An event notification system is the implementation of a particular event notification service.

We refer to an *event notification service* as the general concept and to an *event notification system* when focusing on the implementation of the designed service. Many publications do not give an explicit definition but rather a descriptive introduction into their notion of an event notification service, calling it 'alerting service', 'publish/subscribe service', or 'push system'. Several of these terms are used synonymously to the term 'event notification service', some are only closely related. In the following survey, we briefly review the notion of these terms in the related work. We define each term, briefly clarify what common ideas and intentions lie behind the term's notion, and describe its usage and relation to the topic of this work.

Personalized System: The term *personalization* is not restricted to event notification. It encloses the expression of a client's interest in a personal profile. The profile can be defined either explicitly by the client or implicitly by data mining or analysis of client actions [FD92, SKK97]. Personalized systems operate on the application level, as discussed in Chapter 1. Several event notification services support personalized client profiles, these are personalized systems. Other forms of personalization are adaptation of web-pages to certain customers (e.g., client identification in electronic commerce), personalized web-pages [ABFK99], personalized (relevance) feedback in search-engines [Mou96], and automatic personalized filtering and classification of documents [Coh96].

Alerting Service: The term *alerting system* or *alerting service* is used as synonym to event notification system – it emphasizes the service's application. The term 'alerting service' is now widely used in the context of digital libraries, such as Springer Link Alert [Spr01]. The term 'alerting' also often refers to newsletter services via email. The original applications for alerting services are systems raising alarms in case of hazard, intrusion, or engine failure.

Publish/Subscribe Service: The *publish/subsrcibe* paradigm is an interaction model that consists both of information providers (publishers, suppliers) that publish data to the system, and of clients (subscribers, consumers) that subscribe to issues of interest within the system. The role of publish/subscribe systems is to timely send the right information to the right customer. Publish/subscribe systems are ENS that support publishers that actively send data to the system. Often, events and event-reporting messages are not distinguished. Example services are LeSubscribe [PFJ01], Elvin [SA97], Gryphon [ASS$^+$99], WebFilter [FJL$^+$01], and Scribe [RKCD01].

Event Service: The term *event service* is used by Carzaniga [Car98] as synonym to ENS, but it is more widely used to refer to the CORBA Event Servicet [OMG97b]. The CORBA event service acts on the middleware level and has lower complexity than event notification services.

Push System: The term *push system* refers to an Internet-based system that delivers content to its clients via subject-based channels [Hau99]. Examples of push-systems are Poincast [Inf02], Marimba [Mar03], and Webcanal [Web98]. Push system implementations are often middleware systems that support application-level systems.

Dissemination-based System: In the context of the *Dissemination-based Information System (DBIS)* framework by Franklin et al. [FZ97], an ENS is an *information broker* that acquires information from *data sources*, adds value, and distributes the information to *clients* (= net consumers of information). The term focuses on the distribution of documents, not the observation and filtering of events. Implementations are the DBIS system [FZ98] and the Broadcast Disk System [AFZ97].

System for Selective Dissemination of Information: A *System for Selective Dissemination of Information (SDI)* is similar to an event notification system. The phrase was introduced in the $60's$ [Sal68], it is historically the 'original' term.

Information Filtering System: Such a system is an ENS that especially deals with new or changed documents [Cal98]. These services are also referred to as document filter systems. A first approach to the clear definition of information filtering has been proposed by Belkin and Croft [BC92], a generally accepted definition of information filtering is still lacking.
Yan and García-Molina [YGM95] use 'information filtering' as synonym for ENS, which neglects the aspect of the dissemination and notification. On the other extreme, they use the term *information dissemination* that ignores the aspect of the profile filtering.

Content-based Routing System: The term refers to *Information Retrieval Systems* in which queries are routed to the available servers based on the expected relevance of the server to the query (see, e.g., [SDW$^+$94]). Coft [Cro95] uses the term 'information routing' for this concept. These services are not covered by our work.

(Event) Monitoring System: This term refers to systems that monitor certain event sources, filter events, and send notifications. In general, the event sources are passively-observing sources, such as sensors and measuring heads. The applications range from introspection or supervision of computer programs to communication in distributed systems and web-applications. In research literature, the term mostly refers to distributed program interaction, e.g., performance tuning by automated monitoring [Yan94].

Event-based Infrastructure: Event-based infrastructures support asynchronous message exchange between objects in a distributed environment. They define a middleware for interoperability of independent, heterogeneous systems and build an abstraction for program-to-program communication on the system level. Examples are JEDI [CDF98] and the CORBA Event and Notification Service [OMG97a]. We consider event-based infrastructures as ENS on middleware-level, they are discussed in greater detail in Chapter 4.

Awareness Service: The term *awareness* describes the automatic adjusting of present information of provider and client. The client is aware of all changes at the provider's side, either new objects, deleted objects, or modified objects. The focus of awareness services is the adjustment of (document-centered) repositories, not the notification about events [Swe03, DS97, CGM97]. Awareness services are, therefore, only related to ENS.

Lately, the notion of *awareness widgets* or *awareness displays* is used to describe tools (as the Tickertape [PFK$^+$98]) for communication between human beings (e.g., in institutions). The tools attempt to simulate the advantages of face-to-face situations, thus being aware of other people and their work. In this work, these systems are treated as notification systems.

Current Awareness Service: This term is not clearly defined – it is used to describe a variety of service offers. Most of the implemented services match our definition of ENS. They primarily intend to keep their clients informed about current journal literature of a specific area, such as the service for British Official Publications BOPCAS [BOP03]. The term is used synonymously to the term alerting service.

Event Handling Services: The term *Event Handling Service* describes a service defined in the Java Dynamic Management Kit (JDMK) [SUN03] for the development of software agents. Due to their different abstraction level, these services are not in the focus of this work. However, Olken [OJM98a] uses the term to refer to ENS.

After having reviewed these terms, we conclude that our notion of event notification services includes and extends the notions of alerting services, publish/subscribe services, and SDI services as discussed above.

3.2 Limitations of Existing Models

Event notification systems are, similar to other software systems, often complex entities. To support a reflection about and analysis of different approaches we shall use a model that identifies the main parts and interactions within ENS. A few models for event notification systems exist – however, we argue that a new model is required. For our purpose, a model for event notification services should include:

1. Definition of the central terms in the context of ENS.
2. Identification of the substantial parts of an ENS and their interaction.
3. Specification of the event notification process.

We analyzed both system-specific and general models for event notification services. The following drawbacks of existing models have been identified:

Lacking Independence of Implementation: Several models for event notification services have been introduced by the authors of particular system implementations, e.g., Siena [CRW01] and Yeast [KR95]. Unfortunately, these models reflect the specific implementation. For example, they neither cover active event observation nor include a generic event concept. Few independent models for event notification services have been introduced [KW95, FZ97, RW97, MF98a, SCT95]: The model by Kahn and Wilensky [KW95] focuses on the object storage, event notification is not covered. Franklin and Zdonik [FZ97] propose an infrastructure for notification services that gives a general classification of architectures of dissemination-based systems. The model focuses on the role of an event notification system as broker within an information network. The model lacks detailed definitions at the service level. The model for event-based architectures by Rosenblum and Wolf [RW97] consists of seven parts: object, event, naming, observation, time, notification, and resource model. Several features are addressed only at a high level; the model lacks specifics, such as sophisticated event compositions.

The model by Martin-Flatin [MF98a, MF98b] focuses on network management, it discusses the implementation of push- and pull- based communication with specific technologies, such as java applets and JDBC servers. The model describes the phases of communication, it therefore addresses the aforementioned third demand regarding ENS-models in an implementation-oriented manner. The event-based object model by Starovic et al. [SCT95] describes a distributed-systems programming model introducing objects, events, and constraints. The model focuses on the communication aspect. The event notification process is described in a simplified manner; specific system components are not identified. The model gives the impression that it is developed for a specific implementation.

The design models of observer pattern [GHJV95] and event notification pattern [Rie96] are naturally concerned with design issues of event observation but are independent of a concrete implementation. Both patterns support an event-based communication and decoupling of objects without explicit filter support; they may be seen as design models for simple ENS on the implementation level (for details see Section 4.2.4).

Inconsistent Terminology: On the one hand there are 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. We addressed this problem in the previous Section 3.1.

Additionally, the different models for event notification services 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 providers to communicate with multiple clients asynchronously [OMG97a]. CDF [Ell97] or Netcaster channels [Net99a] are similar to television broadcast channels. In contrast to CORBA, they comprise an active observation method on channel objects.[2]

Inadequate Event Model: In most system-specific event-based models, an event is identified by its *physical representation as message* (see, e.g., [Car98, TIB03]). This approach is not sufficient because

[2]Our evaluation of the implementations of event notification services with channels can be found in [FHS98].

it neglects unobserved events, and the problem of event observation and timestamping. 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. Other models define an event as *a state transition of an information object at a particular time*, where the state of an object is the current value of all its attributes (e.g., [OMG97a, KR95]). We follow the latter approach, because it is more appropriate for an ENS where the invokers often remain anonymous.

However, two aspects of event definitions remain unsolved: the handling of new objects and passive events. New objects: Consider the publishing of a scientific article at a specific time; the state of the object is the content of the article. But what was the state of the object before it existed? The notion of an event as a state transition of this particular object is not sufficient here. Passive events: Consider the example profile 'Notify if a sensor did not send data for more than thirty minutes' (see Example 2.1 on Page 13). Existing models of event notification services do not support this kind of profile[3], as neither is an operation performed on the information object, nor does the information object 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*.

The limitations of existing models for event notification services have been discussed in greater detail in [HF99a].

3.3 Reference Model

In the previous section, we argued for a new model of event notification services (following Requirement R1 from the previous chapter). In the next three sections, we introduce our model for ENS. Our model abstracts from distribution aspects in event notification services: Each part or the complete service may be distributed or replicated in a network. The model consists of three parts: a reference model, an event model, and an event processing model. In this section, we introduce our general reference model[4] for event notification services that identifies the main parts of such services. The event model and the event processing model are addressed in the subsequent two sections.

Within our reference model, we decompose the functionality of an event notification service, identify the main parts[5] and the data-flow between the parts. The reference model addresses the first and second demand regarding ENS models as identified in the previous section. Our reference model is presented as a diagram that shows all logical parts involved and their interaction (see Figure 3.1). In the following paragraph, we describe the parts shown in the diagram. In our reference model, we use a simplified notion of *events* and *profiles*; the terms are analyzed in greater detail in the next section.

Objects of interest are *information objects* that are located at the provider's site, optionally in an *object repository*. Information objects can be persistent (e. g., documents) or transient (e. g., measured values). Changes of these objects (creation, update, deletion) are induced by an *invoker*.

It is the responsibility of the *observer* to detect changes of single objects or in the *object repositories*.

[3]The concept of a passive event has been used as *negative event* in the active database system SAMOS [GD92].

[4]For a general introduction into model types see [BCK98].

[5]The term *part* is used here to refer to the logical units within an event notification service, the term *component* refers to software units of an event notification system. The mapping between parts and components may be one to one or more: A software component may implement a fraction of a functional part or several parts.

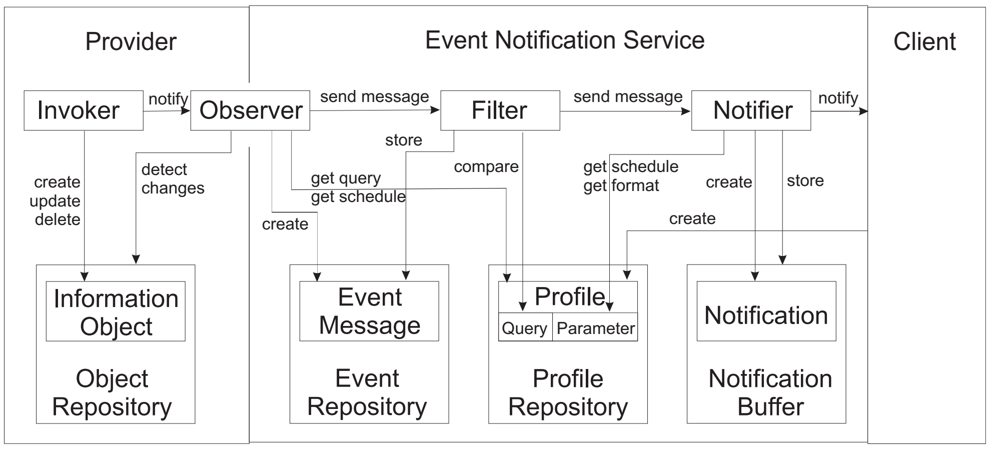


Figure 3.1: Reference model of an event notification service

If the invoker does not inform the observer about changes, the change detection is an active task of the observer (performed according to a schedule). Any change is an event. 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 an event match, the filter creates an *event message* and delivers it to the *notifier*. For the detection of *composite events*, events are stored in the *event repository*.

The *notifier* in turn checks the schedule part of the *profile* (see below). If immediate delivery is demanded, the event message is converted 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.

In client *profiles* we distinguish two parts: The *query profile* (used by the filter) specifies the events in which the client is interested. In the *parameter profile*, additional parameters are defined, such as a schedule, a notification protocol, and a notification format. These parameters may be used by the observer and the notifier.

The components of the event notification service can be (and usually are) deployed and replicated for scalability and reliability. Invoker and object repository usually reside at the provider's site. Not all information providers implement an observer; an event notification service that covers these types of providers could implement an observer as a wrapper for each provider. Alternatively, the observer can be moved to the provider's site (if allowed) and perform its tasks as an agent of the event notification service there.

3.4 Event Model

In this section, we introduce our event model for event notification services. While the reference model describes the interaction of the parts in an ENS, the event model focuses on the items on which the components of the service perform their actions: information objects, events, profiles, and notifications. The event model addresses all three demands regarding ENS model as defined in Section 3.2.

Information Object. In correspondence to other models (e.g., [CGMP99, CGM97, RW97]), we use objects to encapsulate functionalities. In our model, an *object* can be any logical entity residing within a network, such as files and processes. Hardware and human users can also participate but are represented by their software-based proxies. Each object is uniquely identifiable, e.g., by an identifier or handle [CGM97]. For example, a handle may be a URI [BFM98] or a DOI [DOI03]. Considerations of a naming model for event notification services can be found in [RW97].

The objects that are offered by *providers* of an ENS, such as journals, news-pages, or movies, we call *information objects*. We assume that objects are described by attributes. Each attribute has a domain that defines the possible attribute values. Objects have a *state*, which is given by the value of its attributes. A set of information objects offered by a provider is referenced to as *repository*. A provider can offer one or more repositories, examples are databases and sensor networks. Because repositories can also be seen as information objects, we consider a hierarchy of information objects. Additionally, information objects can also be composed of other objects, e.g., a journal consisting of articles.

Event. The central concept of an event notification service is the *event*. In contrast to states, events have no duration. Events may be state changes in databases, signals in message systems, or real-world events such as the departures and arrivals of vehicles. Formally, state transitions are caused by actions such as insertion, deletion, or change of an information object.

Definition 3.3 (Event)
An event e is the occurrence of a state transition of an information object at a certain point in time. This point in time is called (event) occurrence time $t(e)$.

Events are denoted by a lower Latin e with indices, i.e., e_1, e_2. We consider *primitive events* and *composite events*, which are formed by combining events (see Figure 3.2). Similar to a model used for Event Action Systems [KR95], we further distinguish two categories of primitive events: *time events* and *content events*. Time events represent the passage of time – the events refer to certain points in time. They do not consider a certain object (*this* particular system clock) but rather a logical abstraction, e.g., of a global clock. Time events may involve clock times, dates, and time intervals.

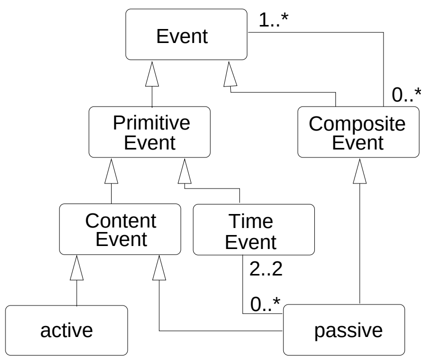


Figure 3.2: Categories of events

Content events involve changes of non-temporal objects, such as sensors. We additionally distinguish *active* and *passive content events*. Active content events are state transitions of an information object at a particular time; they are observer independent. The state of a particular information object is described by the attribute values of the object. A state transition occurs if at least one of the attributes' values is changed.

Passive content events model the fact that for a given time interval (defined by two time events), an object did not change. The profile 'Notify if the sensor did not send data for more than thirty minutes' refers to passive events. Passive events are content events as well as composite events; they have to be observed.

Composite Events are temporal combinations of events. In some models, composite events are called event patterns [CDF98, GD93, RW97]. The contributing events have to be observed, their combination has to be identified by the ENS according to the system's event algebra. The temporal combinations of events are defined by event operators, such as conjunction, sequence, and disjunction. Composite events will be studied in greater detail in Chapter 5.

Events may be reported by means of event messages. In a message, an event may be described by a collection of (attribute;value) pairs, such as the 3 pairs in the following example. The event message reports the crossing of a temperature threshold at a sensor:

$$\begin{aligned} e_{room}: \ event(&temperature = 35\ ^\circ C, \\ &room = 150, \\ &timestamp = 10:00:00) \end{aligned}$$

In the example, we omit the attribute domains, e.g., `real` for the temperature attribute. The structure of an event messages is referred to as message type[6]:

Definition 3.4 (Event Message Type)
The event message type describes the structure of an event message.

Examples for message types are XML [W3C03c] with a certain Document Type Definition [W3C03b] or (attribute;value) pairs with given attributes and value domains, as used here. Providers of event information may announce the event message types they offer to the ENS by means of advertisements. In the literature, often typed and untyped *events* are distinguished. This confuses the event messages reporting the event and the event itself[7]. Each event has a timestamp reflecting its occurrence time:

Definition 3.5 (Timestamp)
The timestamp of an event is the value of a special mandatory attribute of an event message. The time system of reference for timestamps is discrete.

Timestamps are defined within a time system based on an internal clock, while the *event occurrence time* is a point in real time defined by the occurrence of the event.[8] The timestamp is an approximation of the event occurrence time, which might not be known to the ENS. The accuracy of timestamps depends on the employed event detection and timestamping method, which are addressed in detail in Chapter 6.

Definition 3.6 (Event Space)
The set of all possible event instances known to a certain system is called the event space $\mathbb{E}$. The set of all time events is denoted $\mathbb{E}_t$. The event space is formed by the set of primitive events $\mathbb{E}_P$ and the set of composite events $\mathbb{E}_C$: $\mathbb{E} = \mathbb{E}_P \cup \mathbb{E}_C$.

The set of primitive events $\mathbb{E}_P$ contains all events as reported to the system. The set of composite events $\mathbb{E}_C$ detectable by a certain system is defined by the systems event algebra, i.e, by the systems filter and profile semantics. Note that we define a composite event *not* as set of primitive events, as

[6]In contrast to the concept of typed events as introduced, e.g., in the CORBA Notification Service [OMG99], we define the event message type as general structure of a message and do not distinguish whether this structure is predefined.

[7]The event as the occurrence of a state transition is neither typed nor untyped.

[8]We do not deal with the problems of time and relativity; we assume a common notion of continuous real time.

done, e.g., by Gehani et al. [GJ91, GJS92b]. This difference greatly influences the event composition, which is addressed in Chapter 5. Composite events are created based on an event algebra; the algebra defines temporal event composition operators. An event algebra that is suitable for an adaptive event notification system is proposed in Chapter 5. Event composition defines new event instances. The new event instances inherit the characteristics of all contributing events; the event occurrence time is defined by the composition operator. We denote the fact that a set of event instances contributes to a composite event by the $\succ$ operator:

Definition 3.7 (Composition Contribution $\succ$)
Let $e_1, ..., e_n \in \mathbb{E}$ be event instances that contribute to the composite event $e \in \mathbb{E}_C$. This relation is expressed as $\{e_1, ..., e_n\} \succ e$. The $e_1, ..., e_n$ can be primitive or composite event instances.

We distinguish *event instances* from *event classes*: An event class is a set of events specified by an event query while an event instance relates to the actual occurrence of an event. For event instances, we simply use the term *events* whenever the distinction is clear from the context. First, we define the concept of event queries and then, we introduce the notion of event classes.

Definition 3.8 (Event Query)
An event query q_{exp} is a function $q_{exp} : 2^{\mathbb{E}} \rightarrow 2^{\mathbb{E}}$ accompanied by an expression $exp : 2^{\mathbb{E}} \rightarrow \{0, 1\}$.

For simplicity, we omit the index exp for event queries and simply refer to a query q.

Definition 3.9 (Event Class)
An event class $E \subseteq \mathbb{E}$ is a set of event instances. An event class is defined by an event query q_{exp}: The event class is the set of all event instances for which the expression of the defining event query is true $E = \{e \,|\, e \in \mathbb{E} \wedge exp(\{e\})\}$.

Even though events of the same event class share some properties, they may differ in other event attributes. An event class could contain, for instance, all events that describe a temperature change in rooms of a selected building. Then, one event may be the actual change of temperature in room 150 while another event instance may refer to room 200.

Event classes are denoted by an upper Latin E with indices, i.e., E_1, E_2. The fact that an event e_i is an instance of an event class E_j is denoted *element-relationship*, i.e., $e_i \in E_j$. Event classes do not have to be distinct, $e_i \in E_j$ and $e_i \in E_k$ is possible with $E_j \neq E_k$. Event classes can also have subclasses, so that $e_i \in E_j \subset E_k$. Note that the notion of event classes and event message types in ENS differs from the terminology used in object-oriented modelling. A trace, or history, of events is defined as follows:

Definition 3.10 (Trace)
A trace $tr \subset 2^{\mathbb{E}}$ is a sequence of events $e \in \mathbb{E}$. A temporally restricted trace $tr_{t_1,t_2} \subset tr$ is a sub-trace with defined start- and end-points t_1, t_2, respectively.

The history of events that a service processes is $tr_{t_{start},\infty}$ with t_{start} being the point in time the service started observing events.[9] A trace can be seen as a (semi-ordered) list of event instances; we can use the operations commonly defined for lists. Note that the trace is the sequence of event instances that

[9]We do not explicitly distinguish the traces of different services by additional indices as would be formally necessary.

actually *are known* to the service while the event space is formed by all event instances *possibly known* to the service.

Profile. In an ENS, the events of a trace are filtered according to client profiles. Clients describe the events in which they are interested as profiles. In profiles, we distinguish two parts: The description of the events the client is to be about (*query profile*) and additional information about the client and the conditions for observation and notification (*parameter profile*). Here we focus on the query profile. If clear from the context, we refer to the query profile simply as profile.

Definition 3.11 (Profile)
A profile is an event-query that is periodically evaluated by the event notification system against the trace of events.

Within the system, the filter part evaluates the query against the event messages. Note that client profiles specify event classes. We also refer to a profile p by the event class E that this profile defines. An event class is a more general concept, whereas the client profile is specific for ENS. For simplicity, we use predicates on (attribute;value) pairs for examples of profiles. An example profile is $p_{temp} = profile(temperature\ > 35\ °C)$. This profile may be evaluated in an ENS:

Definition 3.12 (Profile Evaluation)
The application of the event query of a profile p to a given trace $tr_1 \subset \mathbb{E}$ is called profile evaluation. The result is a new trace:

$$p(tr_1) := \{e | \big(e \in \mathbb{E}_{\mathbb{P}} \wedge e \in tr_1 \wedge exp(\{e\})\big) \vee \\ \big(e \in \mathbb{E}_{\mathbb{C}} \wedge \exists n \in \mathbb{N}^+ : (e \prec \{e_1, ..., e_n\} \wedge e_i \in tr_1, \text{ for all } 1 \leq i \leq n)\big)\}$$

The evaluation of a profile on an empty trace results in an empty trace.

Before explaining the resulting trace of a profile evaluation, we introduce the concept of event–profile matching. If a profile expression is true for a certain event, then the event matches the profile. Based upon a notation used, e.g., in [Car98], we define the matching operator:

Definition 3.13 (Event–Profile Matching $\sqsubset$)
Consider the event e and a given profile p. The event e matches p, denoted $p \sqsubset e$, if the expression of the profile query is evaluated to true for the event e.

Our example event e_{room} matches the profile p_{temp} ($p_{temp} \sqsubset e_{room}$); the profile p_{temp} defines the class $E_{temp} = \{e \ \mid \ e \in \mathbb{E}, p_{temp} \sqsubset e\}$. For profiles regarding primitive events, $p_{primitive}(tr_1)$ results in a trace $tr_2 \subset tr_1$ containing all matching event instances e, $p \sqsubset e$. For composite events, $p_{composite}(tr_1)$ results in a new trace tr_2[10]. All profile evaluations regarding a certain profile p start after the profile has been defined at time t(p). Thus, for each positively evaluated event e_1, it holds implicitly that $t(e_1) > t(p)$.

[10]Note that our notion of profile matching differs from Gehani [GJS92a], where *always* $p(tr_1) \subset tr_1$.

Notification. A notification is a message reporting about events. Clients are notified according to the schedules given in their parameter profiles. Before notifications are sent, they may be edited, e.g., by removing duplicates, merging, and formatting.

3.5 Event Processing Model

After having introduced the reference model and event model in the previous sections, we now concentrate on the event processing model. This third part of our model emphasizes the cycle of event processing within an ENS. This processing model addresses the third demand regarding ENS models as defined in Section 3.2. We describe the event processing cycle for an active content event in detail; the cycles for other event categories are discussed briefly.

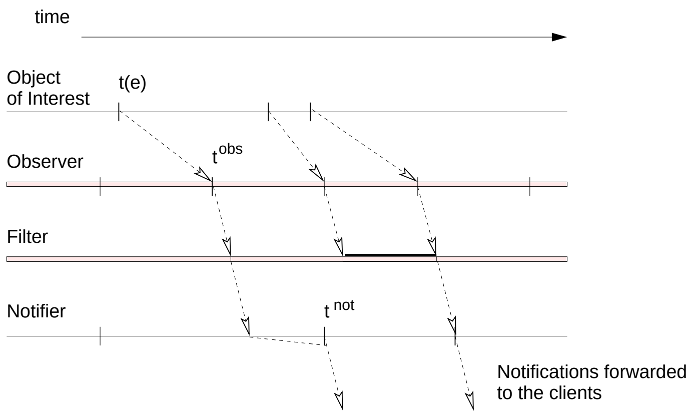


Figure 3.3: Event notification sequence for an active content event

We introduce an *event notification sequence* as the graphical representation of the temporal information flow within an event notification service. A full event notification sequence shows the data-flow from event occurrence until, eventually, a notification is sent to a client. An event notification sequence for an active content event is shown in Figure 3.3. Active content events can happen at any time; they are caused by invokers that perform actions on the information objects (objects of interest). Let $t(e)$ be the occurrence time of an event e, i.e., the time when a state transition e on an object has been caused by an invoker. An observer may 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

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

Here, the time t_0 generally denotes a start point in time, it can be different for different profiles and observers. At time $t^{obs} = t_i$ the observer registers events that occurred in the interval $(t_{i-1}, t_i]$ [11] with t_{i-1}, $t_i \in T^{obs}, i > 0$. The influence of communication delays and operational times for the actors (observer, filter, notifier) are neglected here; their influence is analyzed in detail in Chapter 6. The observer creates a message reporting the occurrence of the event and forwards it to the filter. The filter evaluates the profiles at the reported events.

[11] We employ the common mathematical reference for open intervals using parenthesis $(.)$, for closed intervals using square brackets $[.]$, and for half-open intervals using the mixed notation $(.]$ and $[.)$.

In time events, object and invoker can be seen as integrated: The system clock is an actively changing object. Time events are detected by an internal observer, such as active content events. The clock provides a discrete time signal that defines the minimal granularity Δ^{obs} for the observation. Composite events are temporal combinations of events, the contributing events have to be observed as described above. The possible combinations of events have to be evaluated by the ENS according to the system's event algebra. A composite event is indicated in the event notification sequence by the horizontal bar at the filter, which combines two events (see Figure 3.3). Passive events have no invoker, they can only be recognized by the filter. The contributing time events have to observed. Additionally, the service has to observe the referenced event class – if an event instance of this class occurs then the passive event did not occur.

If an event matches a certain profile, the filter forwards the event message to the notifier. Notifications for the clients may be buffered, checked for duplicates, merged, and delivered according to the client's profile. For a scheduled notification, clients are notified according to a certain schedule

$$T^{not} = \{t_k | t_k = t_0 + k\Delta^{not}, k \in \mathbb{N}, \Delta^{not} > 0\},$$

where $\Delta^{not} \geq \Delta^{obs}$. The events that happened in $(t_i, t_j]$, $t_i, t_j \in T^{obs}$ with $i < j$ are reported to the clients at time $t_k \in T^{not}$, where $t_{k-2} \leq t_i < t_{k-1}$ and $t_{k-1} < t_j \leq t_k$. Because this asynchronous approach can lead to a maximal notification delay of $\Delta^{not} + \Delta^{obs}$, it is recommended 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 $t_i = t_k$. This synchronization ensures that only a minimal phase shift occurs. Immediate delivery is performed at $n = 1$.

Depending on it's schedule, the notifier sends the message to the interested clients at time t^{not} (see Figure 3.3). An unscheduled notifier would cause the notification to be sent, e.g. immediately.

3.6 Communication Strategies and Modes

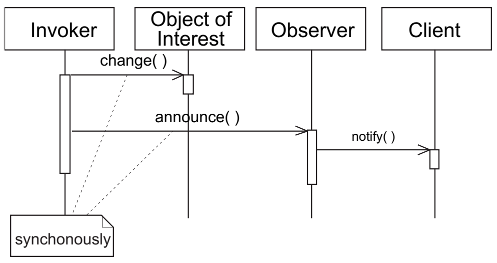
In the event process model introduced in the previous section, we also addressed communication between ENS parts: Events have to be observed at providers' sites and notifications are sent to clients. Until now, we abstracted from the used communication strategies. In this section, we discuss communication modes, specifically different observation and notification strategies. For the external communication of event notification services, two connections have to be considered: provider–service communication and service–client communication. Here, we describe the provider–service communication in detail. The principle can also be applied for the service–client communication and communication in a distributed service.

We distinguish passive/active observations that use a communication mode synchronous/asynchronous to the event occurrence. The four combinations are shown in Table 3.1. *Passive synchronous observation* can be initiated by the invoker or the information object. The invoker changes the information object and synchronously announces the state change to the observer (see Figure 3.4(a)). Passive synchronous observation can also be initiated by the information object, e.g., as in the event notification pattern [Rie96] and the observer pattern [GHJV95]. The object itself announces its state change to an observer (see Figure 3.4(b)). Both forms are facets of a single strategy, because both object and invoker

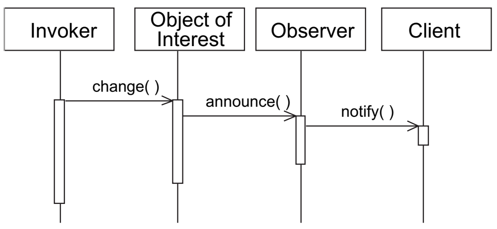
Observation	Mode	Initiator
passive	synchronous	invoker or information object
passive	asynchronous	separate observer on provider's site
active	synchronous	observer (triggered by event occurrence)
active	asynchronous	observer component of the ENS

Table 3.1: Communication strategies and modes

reside on the provider's site. We shall illustrate later that in a certain implementation, the distinction between the two modes depends on the considered abstraction level. For *passive synchronous observation*, the provider employs an observer of events on its site, which asynchronously reports the events to the ENS.



(a) Initiated by invoker



(b) Initiated by information object

Figure 3.4: Passive synchronous observation

Active synchronous observation of events is conceivable but not often employed: The observer in the ENS is triggered by the event occurrence. This triggering is already a synchronous (passive) observation in itself. Then, the observer performs a more detailed observation than the triggering can provide. *Active asynchronous observation* is performed as follows: After the invoker changes the information object, the observer verifies the object's state on a regular or irregular basis. The state change is detected by the observer after a certain delay (see Figure 3.5). The influence of different observation strategies on the accuracy of event detection is discussed in greater detail in Chapter 6.

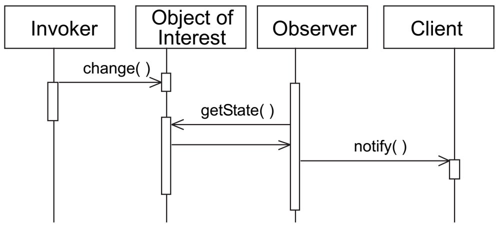


Figure 3.5: Active asynchronous observation initiated by observer

The communication strategies *passive observation* and *active observation* are often referred to as *push* and *pull*. In the *push model*, the provider informs the ENS about events. In the *pull model*, the ENS observes events at providers' sites. We argue that the identification of push and pull communication of an ENS depends on the abstraction level of the communication and the distinction of the two communication connections. To support our argumentation, we study the example of an Internet-channel

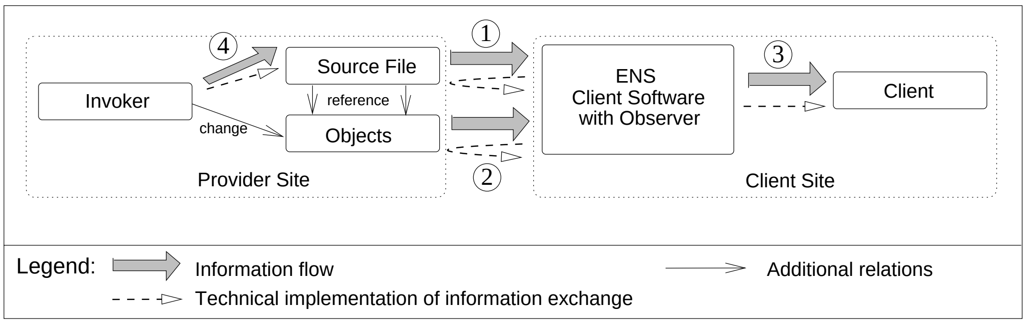


Figure 3.6: Communication at CDF channels (see Example 3.1): Arrows represent observation strategies on different abstraction levels

that uses the CDF-technique [Ell97]. This example shows how within one communication, different communication strategies are used on different levels.

Example 3.1 (Communication in CDF Channels)
Providers of a CDF channel provide a source-file that contains links to Internet-sources, clients subscribe to the channel by initially downloading the file. The ENS software resides on the client's site (see Figure 3.6). The new content of the channel is pushed to the clients. By considering the underlying techniques, one becomes aware of the fact that the client software regularly pulls the source-file and reports changes in the file to the clients. The provider either updates the file on any change of the referenced objects or the client software has to crawl for information about state changes of the referenced objects. Figure 3.6 shows the information flow (gray arrows) and the underlying technical implementation (dashed arrows) for our example. Communication is performed on several levels, the enumerations refer to the circled numbers in the figure.

1. *Application level: The provider pushes new changes in the source file to the client. Middleware level: The client-software inspects the source-file for changes (active asynchronous observation by observer, pull).*

2. *Application level: The provider pushes information about new changes in the referenced objects to the client. Middleware level: The client-software crawls for information about the referenced objects (active asynchronous observation by observer, pull).*

3. *Application and middleware level: The client is actively informed by the channel if a channel update occurs (passive synchronous observation, push).*

In the example, push communication is only used for system–client interaction. The provider–system interaction uses pull-style. This is even more important, since the provider–system interaction is performed over the network, while the system–client interaction is performed on the same site.

On application level, the source file is provided as information object. On a lower level, we identify the Internet-sources as information objects. The interaction of invoker and source file implement a rudimentary event notification service with the higher-level service as client:

4. *Middleware level: The invoker changes the referenced objects and the source-file. The source file collects the information about the changed object, it can be seen as passive observer (passive synchronous observation by invoker, push).*

The system–client interaction is asynchronous active observation (pull) as in (1).

Example 3.1 shows that push-communication can be implemented on a lower level using pull-communication and vice versa. Consequently, the identification of communication styles depends on the considered abstraction level. In this thesis, we concentrate on communication on application level for general considerations. However, for the detailed analysis of observation methods in Chapter 6, we focus on the middleware level.

3.7 Summary

In this chapter, we first identified terms and concepts that are related to the term 'event notification service'. We provided a survey of these terms, their usage and relation to the topic of this work. Then, a unified model for event notification services has been proposed. Our model consists of three parts: a reference model, an event model, and an event processing model. The reference model of ENS provides a general terminology for the description of logical parts and their interactions within event notification services. Within the event model, the items handled by the ENS are defined in detail. The tasks of an ENS and the event notification sequence have been described within our event process model. The model evolution has been published in [Hin99, HF99b, HF99a]. The model presented in this chapter is used as a basis for the analysis of related systems in the next chapter.

Already, 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 object repository and the information objects contained in the repository. In the latter case, observers need to be initialized with the states of all existing objects in the repository. They can only detect changes that occur with frequencies less or equal to the observation frequency: If two events regarding the same object happen within the same observation interval, they cannot be distinguished by the observer. The implications of these conclusions are addressed in detail in Chapter 6.