

AMETAS

Good Migrations.

AMETAS White Paper Series

by Klaus Herrmann
and Michael Zapf

On Events

July 2000

Johann Wolfgang Goethe-Universität Frankfurt/Main, Germany
Department of Computer Science

www.ametas.de

The Event Mechanism

The *event system* is one of two major communication mechanisms in AMETAS. The other one, the *messaging system* allows asynchronous communication between Place Users, while the event system enables a unidirectional communication between the core components of AMETAS and Place Users. Events are generated by the AMETAS core to notify Place Users of certain conditions which often need to be processed in a timely fashion. At first glance, one might argue that sending messages to Place Users in these cases would be the better alternative in the pursuit of a truly uniform communications system. But events differ from messages in some significant ways.

Events versus Messages – What’s the Difference?

While messages are generated and deposited by one Place User and actively retrieved by another one, events are always generated by the AMETAS core and instantly forwarded to the Place Users that registered for the respective event. Place Users do not exchange events with one another. The core service that is used to generate events (we also say that events are *fired*) and performs the forwarding is the *Event Manager* while the messaging system is build around the *Post Office* as a storage facility for messages. Since events are always delivered directly, no storage is necessary for them. Summarizing, we can say that messages and events are two separate, complementary communication mechanisms. However, as we will see below, they work together to form new ways of communication.

How Are Events Structured?

AMETAS events belong to different categories that are subdivided by *Event IDs*. While categories exist for grouping related information together, IDs indicate a concrete information within a category. Currently, there are three major categories of events that cover different information domains:

- **Message Events** are fired if an action is executed on the Post Office. Such actions include depositing and deleting messages. As the White Paper on communication states, Message Events are an important supplement to the messaging system because they enable a direct delivery of a message to Place Users that are present at the time of message arriving at the PO.
- **Place Events** are fired for example if the place is scheduled to be shut down and allows all present Place Users a certain amount of time to do their cleanup or leave the place. Other Place Events are fired if errors occur within the place.
- **Place User Events** transport information concerning Place Users. If, for example, an agent arrives at a place, a corresponding event is fired and registered Place Users are notified of the arrival. This might help in certain cooperation schemes in multi-agent applications and can be used to monitor agent activity.

These categories are modeled as separate classes derived from one event superclass. Event IDs are members of the event classes.

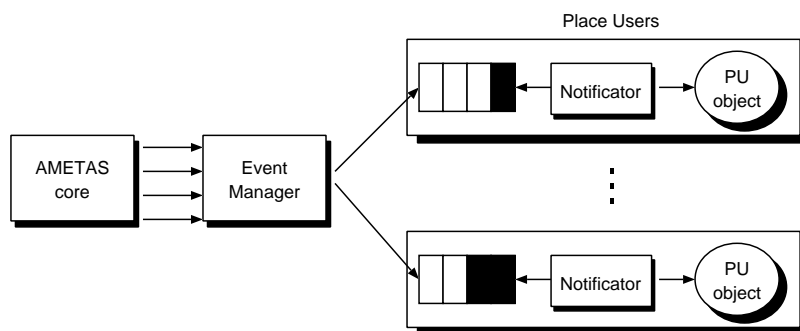


Figure 1: The flow of events

The Event Manager

As already stated, the Event Manager is the central entity of a place that controls the registration of *Event Listeners*, the filtering of incoming events and the forwarding of events to designated Place Users. An event is fired inside the AMETAS core by calling the Event Manager's `fireEvent` method. This initiates a filtering process upon which the Place Users are collected that registered for the event. For the actual notification, every Place User maintains an event queue and a separate notificator thread. The Event Manager inserts the event(s) into the Place User's event queue from where the notificator thread retrieves them in the order in which they arrived. The notificator then calls the `notifyListener` method on the Place User to actually deliver the event. This method needs to be implemented by the agent programmer to process the events according to its task and internal state. This flow of events is depicted in Figure 1.

Registration

Before a Place User can be notified via events, it has to register for a specific event or a set of events first. Each Event ID provides different options and parameters to configure the notification process. For example, a Message Event can be registered with a so-called *Message Mask* that acts as a message filter by specifying certain header fields and supplying wildcards for others, the set of messages being subject to notification can be defined this way. Place User events on the other hand need to be registered with a *Place User ID Mask* to specify the set of interesting Place Users in a similar way. After a successful registration the Event Manager will forward every event matching the registration parameters to the Place User.

Extended Event Handling

The notification scheme presented above is called *basic event handling* since the Place User only needs to implement the Java interface that provides the `notifyListener` method. After a successful registration and the generation of an event this method is called with the event as its argument.

This is a simple scheme and might lead to confusingly long `notifyListener` methods in Place Users that have to deal with many different events. Therefore, AMETAS

also provides a scheme called *extended event handling*. A class called *AMETASEventHandler* is part of the AMETAS distribution. In conjunction with the Place User extending one of the *Notifiable* classes, this class is used to give a clean structure to the handling of multiple event classes. An Event Handler provides several methods that deal with different event classes and event individual Event IDs. The programmer can put the code specific to handling a certain event in one of these separate methods and achieve a well-defined structure of his code. The *Notifiable* superclass takes care of preprocessing and casting the incoming events and handing them to the right Event Handler methods.

Event Handlers can also be subclassed to encapsulate certain aspects of event processing. Another advantages of extended event handling is the possibility of using several separate Event Handlers within the same Place User. This can be quite useful if specific parts of a Place User's code are supplied by a third party and have to define their own event handling. An example of this are *Service Proxies* that are described in the White Paper on communication.