Events, contexts, and incidents
Architecture concepts
White Paper on foundations of autonomous management

Dr. Wolfgang Thronicke
Siemens AG

C-LAB Short Report

Vol. 3 (2009) No. 1

Cooperative Computing & Communication Laboratory

ISSN 1614-1172

C-LAB Short Report

Herausgegeben von
Published by

Dr. Wolfgang Kern, Siemens AG
Prof. Dr. Franz-Josef Rammig, Universität Paderborn

ISSN 1614-1172

C-LAB
Fürstenallee 11
33102 Paderborn

fon:	+49 5251 60 60 60
fax:	+49 5251 60 60 66
email:	info@c-lab.de
Internet:	www.c-lab.de

Reactive and proactive systems rely on efficient handling of internal data and the timely notification of changes in components. This paper explains the concepts for the processing of such notifications in software systems and components and presents a concept of managing higher level state information as contexts and their relation to incidents which are the basis for intelligent autonomous systems and advanced management functionality.

Dr. Wolfgang Thronicke
Siemens AG

Motivation

Situation- and context-awareness are common keywords to describe actual systems which offer autonomous functionality for the user by ‘understanding’ what is happening and how to react appropriately. However, the term context itself is quite overloaded and often mixed with the term events.

In the German OSAMI project one aspect to be researched and developed is a sophisticated management component which should (semi-)autonomously react on certain occurrences during system operation. In order to design this software for the conceptual level a closer investigation of the relation of events to contexts and higher-level incidents needed to be made and is elaborated in the following sections.

Events

Every modern software system requires a means to communicate data from one component – the producer – to others – the consumers. Despite of having the consumers constantly check the source component for new data (polling), the source pushes new information to components which have registered or subscribed to this information. This information is called an 'event' which is more generally a message sent with the payload of event data contained.

This typical pattern is common for user interface toolkits, where user input like mouse movements, or keyboard use causes events which are propagated to processing components. In short: something happens and an event is triggered and sent to the recipients.

Systems built on this paradigm are called event-driven or event-based. Especially when the triggered events are not caused by deliberate human interaction this systems are called 'reactive'.

In fact, event processing is message processing. A typical pattern (see Figure 1) is that of EventProvider and EventListener which describes the call of a routine in the listening component1. Events usually carry the information about the cause of the event and the data involved, e.g. a mouse click and the actual mouse coordinates.

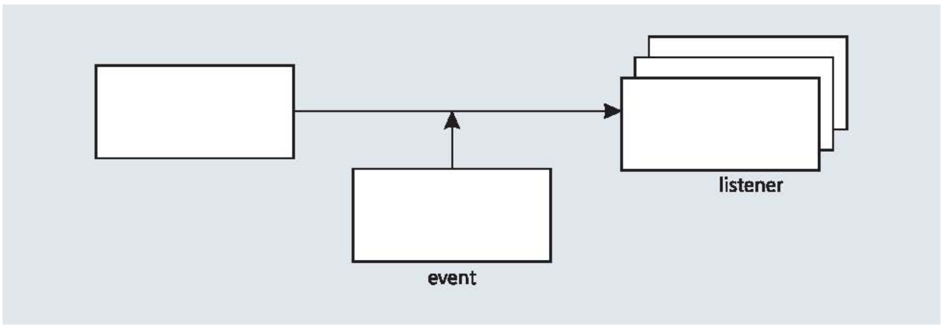

Figure 1: Event Listener Pattern

[1] In programming languages like JAVA or C# this is denoted by implementing a listener interface which requires to implement the event handler.

Context & Incident

Since an event is only a singular occurrence it is quite complicated to talk about the situational context or the state context of an application or system only based on events.

Context events

For our purposes a context can be defined as the set which contains of a set of events and derived data which are represented as 4-tuples (time, source, type, data) of context-events2. The time records the actual time of occurrence, source describes who has produced this context, and type defines the meaning of the data. The term context processing describes mechanisms which take such contexts and introduce new derived contexts with a more specialized meaning. A simple example is the processing of a geo-coordinate context into the context 'at-home'.

During the life-time of the system (or program run) context-events are generated starting from t0 to tend. The ordered by-time context-events form the context-trail or context-trace of the system. However, it is usually not the complete trace which is required to derive certain conclusions about a system, especially when actions are to be derived from such these conclusions. A context is called relevant if it is sufficient to make such a statement.

The current context of system can be defined as partial set with context-events belonging to a defined timeframe backwards from the current time. This describes the current view of the system about its state. The current full-typed context contains the current context augmented by the most recent context-events from types not in the current context.

[2] The introduction context-events is necessary in order not to mix a context (which is a set) with the single item (created by events or processing).

Context patterns

The problem of deriving conclusions from contexts is to define a means to constrain the amount of contexts to be evaluated from the context-trail. The set of all relevant contexts to derive a statement S specify the context-patterns for statement S. So an identifying process can focus on comparing the current context of a system with the context-patterns for S.

Most times context-patterns are described in a high-level description like: ‘Given the set of temperature sensors of the house watch for a rise of temperature over 40 degrees from the same sensor in the last 5 minutes.’ The formal expression of this pattern is of course more complex and harder to understand. However, this kind of description is usually enough to implement the pattern in a search engine for context patterns.

Incidents

When state of the system can be described on a high level as set of all current statements found out from the current context, statements can remain valid until they are renewed by a context pattern identification process or they expire after a certain time. From a formal approach it is convenient to define a statement having the possible values of true, false, or unknown. Thus, an expired or not yet computed statement is simply ‘unknown’.

Every reevaluation of a statement may change the current state of a system. All states which have to be acted upon are called incidental. Thus, the change from a non-incidental state to an incidental one is called an incident. From a system point of view it is the incidents which signal a system failure or a management or maintenance issue or any other action. Of course, this seems to be like an event by itself but an incident carries a semantic meaning because it contains the distilled contextual information and the mapping to the work environment and use-case of the system.

Architecture for Context and Incident Processing

The advantage of the separation of events, contexts, and incidents becomes evident, when a flexible, and modular system for processing contexts and managing incidents has to be conceived: Sensors, and their device driver usually build the proprietary layer which is closely connected to the operating system. Thus, the events they throw and the data provided by them use specific protocols which have to be mapped into standard data-types available. Sometimes a device itself does not throw an event but can only be polled, so there has to be a specific event generator component. In order to enable a unified processing across different event sources the context processing layer serves as abstraction from the partly not homogenous events as provides a common interface for notification and processing. A specific component the ‘context store‘ serves as persistent or transient memory for such context-events. With this standardization the incident detection and processing is decoupled from individual events and event-sources can be formulated as generic common part of the system. Figure 2 gives a top-level view of this architecture. Context handlers can directly work on the context store and read and write contexts. For instance, high-level contexts can be introduced which are computed from other contexts.

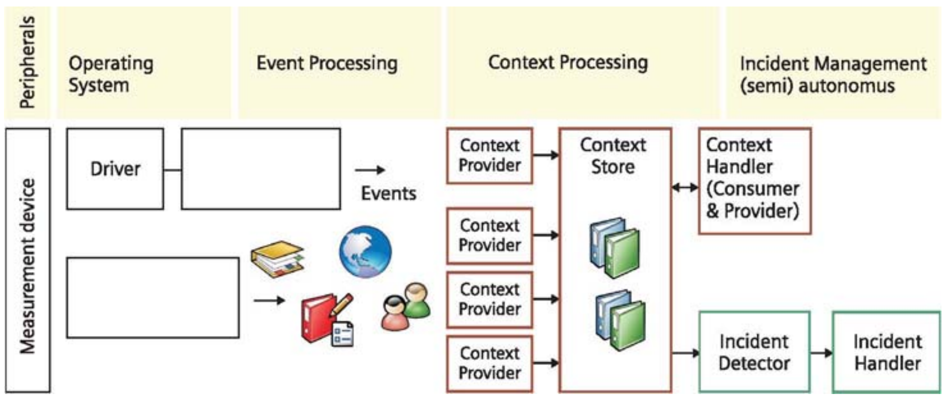

Figure 2: Top-level architecture

In this sense the incident detector is also a specialized context handler which searches for specific context-patterns to identify incidents which are then processed by the incident handler.

The incident handler can serve several purposes: In a simple realization it could simply send alerts about incidents to a remote system, or – in a more sophisticated scenario – autonomously react on this by triggering suitable management and control functions.