A software architecture for intelligent facility management based on complex event processing

Ralf Vandenhouten, Ralph Holland-Moritz

Zusammenfassung

Dieser Beitrag präsentiert einen Architekturvorschlag für ein intelligentes Management-System, das Daten mit Hilfe von künstlicher Intelligenz auswertet. Die Architektur bedient sich dabei des Complex Event Processing, um eine hohe Flexibilität bei der Verknüpfung der einzelnen Komponenten zu erreichen. Die Komponenten und deren Zusammenspiel werden am Beispiel eines Gebäudemanagementsystems illustriert.

Abstract

This article presents an architectural suggestion for an intelligent management system which evaluates data using artificial intelligence. The architecture uses complex event processing in order to gain high flexibility when connecting the individual components. As an illustration of the components and their interaction a facility management system is used as an example.

1 Intelligent Management System

An intelligent management system can be divided into four groups of components. These groups are input components, intelligent components, data converters and reporting components as it can be seen in Figure 1. A facility management system is used as an example of such a system. In this environment the input components are sensors which provide input data for the intelligent system. Sensors are defined as technical components measuring one or more physical or chemical properties or material type of goods in their environment. The measured property is called a feature or attribute of a sensor and has a specific value at a given time.

Fig. 1: Components in a simple architecture

The reporting components, forming the output of the intelligent management system, are generated reports with status information or schematic representation like ground plans of the building with connected state indicators. The reporting component is the interface for the facility manager or security agent, giving him an overview of the events inside the building or location.

Between input and output components there is an interlayer consisting of intelligent components. These intelligent components evaluate the input data coming from the events and generate output data as result of the evaluation process with the help of their knowledge. The intelligent components can be divided into two types depending on how the knowledge is stored or gained. These two types are the learning components and the knowledge-based components.

A learning component is able to gain knowledge from changing findings and to draw conclusions from simulation or normal operation. The learning component is applicable for event sequences which are unknown or not describable.

Under knowledge-based components we understand systems which evaluate new statements with the help of a knowledge base. The knowledge base is formed out of statements which are defined as rules, logic statements or semantic connections. With the help of this component changes in the operational application can be evaluated by means of a rule base, defined by a so called expert. For evaluating the statements of the knowledge-base and a subsequent conclusion a rule interpreter is used which interprets the facts by means of predefined rules. Knowledge-based components work well for the reasoning of events by means of a knowledge base consisting of elementary rules. It can be used where states of the system are known a priori.

Different types of intelligent components need different types of input data and provide different types of output data. Some intelligent components have their own event memory, others don't. The ones without memory need a system state copy every time they are executed. To connect the input data from the input components with the intelligent systems and the output data from the intelligent systems with the output components we need data converters for input and output conversion.

2 Requirements

In an intelligent management system with different intelligent components there are requirements to ensure flexibility and other non-functional requirements.

The first non-functional requirement is the requirement of parallel processing. Under parallel processing the parallel execution of the intelligent components is understood. That means that the components should work independently from each other. In an intelligent management system the focus is on time dependant events which have to be evaluated time-critically. Because of their different behavior the intelligent components can evaluate events in different speed. Parallel processing therefore means that one component is independently from another in the sense of not having to wait for the other's termination.

When dealing with different intelligent components it is previously unknown how these components work together. It cannot be determined whether component »A« and »B« evaluate before »C« and if »C« uses the output of »A« and »B« or whether they evaluate parallel or in another way. Thus the intelligent components cannot be connected in a fixed way. This means that a loose coupling is needed for connecting input and output of the intelligent components.

In order to be not restricted to specific intelligent components and to be able to further develop the intelligent system or extend the system with new intelligent input or output components a requirement of the system is to be easily extensible. This means that the system should be extendable by other components with little effort and without changing basic components.

3 Solution

For connecting different components together with loose coupling a possibility is to use the observer pattern. With this behavioral pattern there are two central types of objects called subject and observer. In the example of a facility management system the subjects would be sensor events and the observers would be the intelligent components. Another form of this pattern is known as publish-subscribe which focuses on a publisher which would be the event generator and on the subscriber which would be again the intelligent component (Gamma et. al. 2004). To facilitate the subject from the publisher to the subscriber a middleware is needed for registering the subscriptions and for receiving and forwarding or distributing the subjects.

For realizing the loose coupling in the form of publish and subscribe two related approaches arise: message-orientated and event-based middleware. With message-orientated middleware (MOM) the communication between the different components is made by abstract messages with the help of an interlayer (Heinzl et. al 2005). For ensuring a defined access an interface definition was defined called Java Messaging Service (JMS). Known implementations are Apache ActiveMQ, JBoss Messaging and OpenJMS. With JMS we have to kinds of roles a client can play. This is the message producer and the message receiver. A server called JMS-Provider acts as a broker between the clients. To achieve a loose coupling between the clients the delivery of the clients works with queues or topics under which a message can be published or subscribed.

Message-based solutions are usually complete middleware systems. Middleware is often designed for the operation of a distributed system on a high abstraction layer and thus generates additional overhead in communication. In addition they offer more features which are beyond the pure communication (Heinzl et. al 2005). According to this the available implementations of JMS are heavyweight. For allowing a lightweight implementation a lower abstraction layer for the message communication has to be seeked for, besides the resignation of additional unnecessary features. As a solution there is the communication on a lower abstract level by means of events instead of messages.

The simplest solution to realize the communication by means of events is to implement the observer pattern with the help of regular Java structures like event queues. This solution means additional implementa-

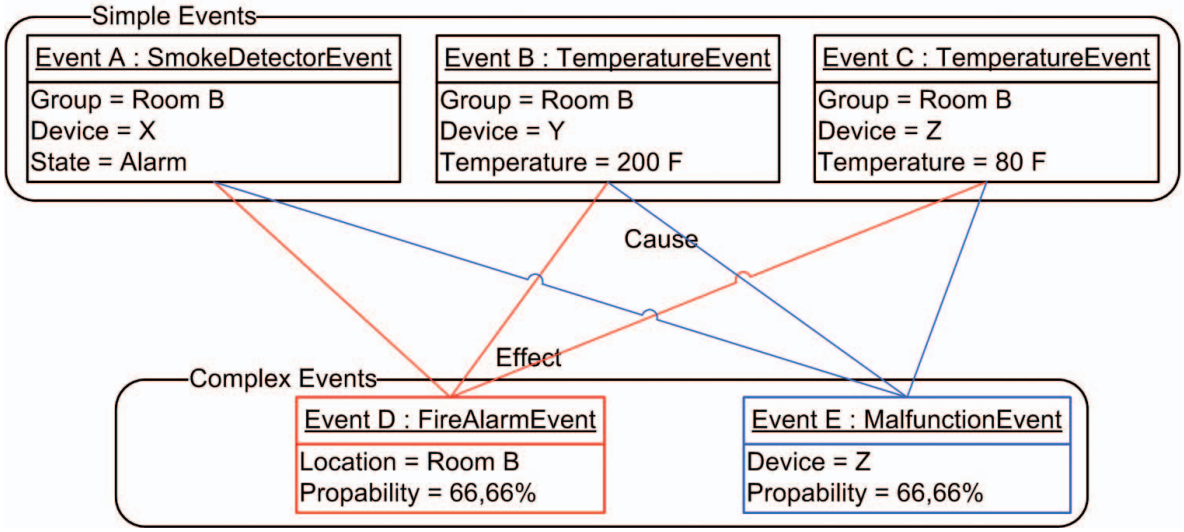

Fig. 2: Relationship between simple and complex events

tion cost and involves the danger of error sources. In particular, the observer pattern is inadequate for messaging in distributed systems with an RPC middleware like R-OSGi (Vandenhouten et. al. 2009). An alternative solution of using events for the communication between local components is to use an event-orientated architecture also known as Event-Driven Architecture (EDA). Connected with EDA is the software technology for the realization of EDAs called Complex Event Processing (CEP). As CEP, besides the possibility to communicate by means of events, offers additional features which support further aspects of the intelligent system CEP is the preferred choice for the communication layer of the intelligent system.

3.1 Event-Driven Architecture and Complex Event Processing

The Event-Driven Architecture (EDA) represents a software architecture pattern where events take center stage. These events can occur inside or outside the system and are published to all interested parties of the system by a proper mechanism (Bruns et. al. 2010).

With the centric treatment of events EDA decisively differs from the Service-Orientated Architecture (SOA) where functions take center stages which are offered as services. In this case one or more distributed applications communicate with help of provided services which forms a SOA (Barry 2011). Despite the different approaches or rather because of this fact both architectures can be combined. Thus an EDA can be integrated into a SOA (Charles et. al. 2010).

Besides EDA, which represents a general concept as event-orientated design pattern, the Complex Event Processing (CEP) exists as a concrete technology for event processing which can be considered a core element of an EDA.

The event processing which exists since more than 50 years is a subject of the computer engineering which gains more and more importance. Started as a method for changing threads in a processor by means of special events, events in the event processing are considered more general today. Thus in today's event processing more abstract events like sending a message or changes on the financial market like a purchase or sale of a share are processed. The basis of the today's event processing forms the discrete event simulation of the early 1950s. The idea was to simulate hardware control systems or natural phenomena with the help of a simulation language. In this connection events where generated which effected the generation of new events. After the simulation the emerged event chain had to be analyzed, which matches the today's idea of event processing. A second area, which decisively forwarded the development of the event processing, is the development of the packet-orientated network technology in the late 1960s. The resulting TCP/IP protocol and the OSI model which divides communication in differently abstracted layer plays a central role in the development of the complex event processing (Luckham 2007).

Events: In event processing events can represent any kind of occurring activities. Thus the »Event Processing Technical Society« (EPTS), which is a community with an interest on event processing, defines an event as follows: »Anything that happens, or is contemplated as happening.« (Luckham et. al. 2008) As examples the EPTS cites the following:

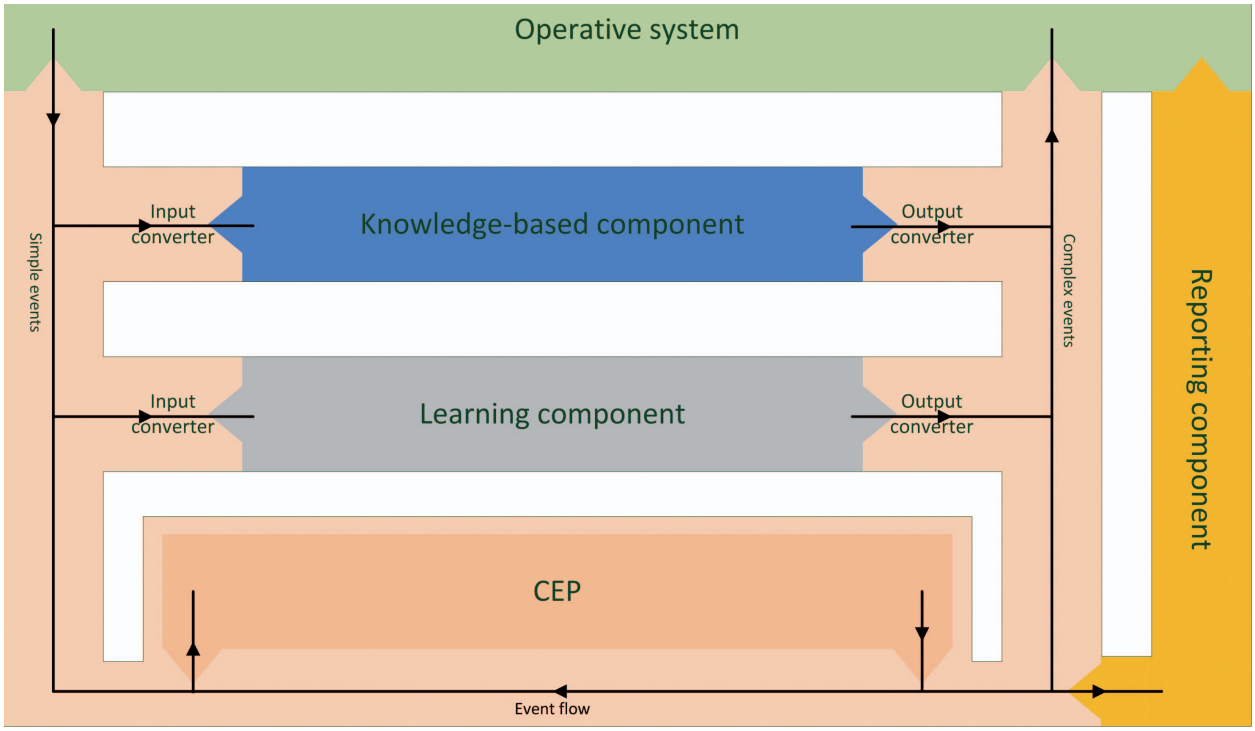

Fig. 3: Example architecture using CEP

- A financial trade
- An airplane lands
- A sensor outputs a reading
- A change of state in a database or finite state machine
- A key stroke
- A natural occurrence such as an earthquake

A definition of a mapping of a real event in the information technology as so-called event object, event message or event tuple is defined by the EPTS as follows: »An object that represents, encodes or records an event, generally for the purpose of computer processing.« (Luckham 2007)

Another definition of an event in the information technology is given by David C. Luckham: »An event is an object that is a record of an activity in a system. The event signifies the activity. An event may be related to other events.« (Luckham 2001)

3.2 Complex Events

According to Luckham an event can be associated with another event. When these events are merged the resulting abstract event is called a complex event. David C. Luckham defines a complex event as follows: »A complex Event is an aggregation of other events, called its members.« (Luckham 2001)

Events can be associated in different kinds. The most important types of relationships are the time, the causality and the aggregation (Luckham 2001).

The time-dependent relationship of events means a relationship of events related to their temporal occurrence. Thus different events can occur with a temporal offset or together. The causal relationship between events means the dependency of an event to another event that occurred before. If a subsequent event occurs because of other events then these events are causally associated. When different sub events are merged then the relationship between the events is called aggregation. The resulting aggregated event forms a complex event which represents an abstract image of the sub events. An example of aggregated complex events can be seen in Figure 2.

3.3 CEP as middleware

Because of the fact that Complex Event Processing allows the distribution of events to a registered receiver CEP will be used as middleware or rather communication channel for the single intelligent components of the intelligent system. Thus the components, respectively their input converters, register themselves as receivers of elementary and complex events. After the evaluation of the incoming events the components publish the evaluation results as complex events to the CEP system. Thus a steady flow of events at different abstraction layers arises. Through the loose coupling of the components by CEP the event flow through the single intelligent components is not limited as you can see

in Figure 3. This way a component can use the result of another component and the resulting event can also be used as input of a third component and so on.

For reporting the results of the evaluation which exist as events in the CEP system the presentation component registers itself as a receiver of the events to be displayed. Through the steady event flow in the CEP system in form of elementary and abstract or complex events, respectively, a real-time view of the complex state of the intelligent system through all abstraction layers is possible.

CEP used as middleware meets the requirement stated in Chapter II. It offers parallel processing of events through its event processing engine because parallel processing of many events is a basic need of CEP in general. Loose coupling is realized with CEP acting like a central anonymisation layer between the different components of the intelligent system. The only entities that are known to the components are events which act as communication channel between the single components and the CEP engine itself. Figure 4 illustrates the relationship between two components with the help of CEP.

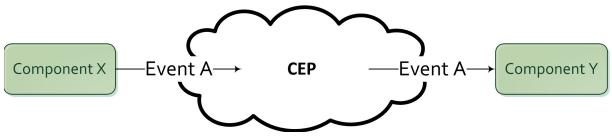

Fig. 4: Loose coupling through CEP

3.4 Other roles of CEP

As already mentioned the different intelligent components need and produce different types of data. Thus input and output converters are needed for every single intelligent component. These converters could be implemented by means of programmatic structures. With its capabilities to merge, group and aggregate events in dependency of time or other attributes CEP is predestinated for event conversion. This way converters can be implemented in CEP once and be reused by different components.

Besides other features CEP offers methods of evaluating events. When simple events are associated to each other in the form of generating complex events then the resulting event can be seen as a conclusion from the simple events. Thus CEP can be used as intelligent component reasoning from simple to complex events.

4 Conclusion

An architecture of an intelligent management system has been presented that consists of input components, intelligent components, and reporting components, where the components are connected via converters. CEP can be employed as the backbone of such an architecture, supporting the components by providing various types of event related services. It allows for loose coupling, extensibility and parallel processing. With its event processing capabilities it can also be used to implement converters for the different types of intelligent components and act itself as an intelligent component.

Biblography

Barry, D. K. (2011): Service-oriented architecture (SOA) definition, http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html, cited: 05.09.2011.

Bruns, R., Dunkel, J. (2010): Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse. Springer Verlag, Berlin, Germany.

Charles, O., Schalk, M., Hollunder, B. (2010): CEP meets SOA: Komplexe Ereignisse bringen Mehrwert in SOA-Infrastrukturen. In: OBJEKTspektrum, 2010, vol. 5, 28-33.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (2004): Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software. Addison-Weslay, Munich, Germany.

Gualtieri, M., Rymer, J. R. (2009): The Forrester Wave™: Complex Event Processing (CEP) Platforms, Q3 2009, Forrester Research, Inc.

Heinzl, S., Mathes, M. (2005): Middleware in Java: Leifaden zum Entwurf verteilter Anwendungen – Implementierung von verteilten Systemen über JMS – Verteilte Objekte über RMI und CORBA. Friedr. Vieweg & Sohn, Wiesbaden, Germany.

Luckham, D. C. (2001): The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Weslay, Boston, MA, USA.

Luckham, D. C. (2007): A Short History of Complex Event Processing. Part 1: Beginnings, http://complexevents.com/wp-content/uploads/2008/02/1-a-short-history-of-cep-part-1.pdf, cited: 05.09.2011.

Luckham, D. C., Schulte, R. (2008): Event Processing Glossary –Version 1.1, http://complexevents.com/wp-content/uploads/2008/08/epts-glossary-v11.pdf, cited: 05.09.2011.

Vandenhouten, R., Kistel, T. (2009): Aus der Entfernung – Verteilte Dienste mit R-OSGi. In: iX Magazin für Professionelle Informationstechnik, 2009, vol. 9, 142–146.

Authors

Prof. Dr. rer. nat. Ralf Vandenhouten
Fachgebiet Telematik
Fachbereich Ingenieurwesen/Wirtschaftsingenieurwesen
Technische Hochschule Wildau [FH]
T +49 3375 508-359
ralf.vandenhouten@th-wildau.de

Ralph Holland-Moritz, M. Eng.
Fachgebiet Telematik
Fachbereich Ingenieurwesen/Wirtschaftsingenieurwesen
Technische Hochschule Wildau [FH]
T +49 3375 508-616
ralph.holland-moritz@th-wildau.de