Logic-Based Modeling of Event-Oriented Architectures for the Development of Application Systems

Matthias Fischer[1], Erich Ortner[1], Joachim Sternhuber[1]

[1]Technische Universität Darmstadt, Development of Application Systems,
Hochschulstraße 1,
64289 Darmstadt, Germany
[fischer, ortner, sternhuber]@winf.tu-darmstadt.de

Abstract

In this paper we propose a new paradigm concerning the schematic consideration of events in opposite to things. Starting from that paradigm, a new approach to a logic-based, language-critically motivated modeling-approach and modeling-language is derived. Further, some examples for logics which can be used in our approach are presented and exemplified. These logics can be used in a modular way. Finally we present an event-oriented architecture for application systems, which is applicable for the implementation of application systems designed by usage of the event-oriented approach presented in this paper.

1. Introduction

While events are becoming ever important concerning design of software-products and event-driven architectures (EDA) are on the rise, a standardized modeling-language that is capable to model events in an easy and clear manner is still lacking. (Ortner and Schneider 2008) presented the main features of a logic-based language-approach for an event-oriented modeling-language. This paper picks up that language-logical approach for event-modeling and event-applications, exemplifies it and depicts the advantages of that approach in opposite to previous approaches.

The Event Processing Glossary of the Event Processing Technical Society (EPTS, www.epts.com) defines an event as "anything that happens, or is contemplated as happening".

This rather general definition is to help harmonizing the manifold and not standardized definitions for "event" in context of complex event processing (CEP).

In this paper we define an *event* as "anything that happened, is happening, will happen or might happen". That definition does not tie the concept of an event to occurrences in a computer-program. Rather, it includes occurrences in physical world, like the receipt of an order in distance selling or the completion of a product. We follow (Sowa 2002) by stating that an event is identified by its position in time and space and by the ontological classification given by Sowa. A prerequisite for an event to take place at a certain position in time is that it has a starting- and an ending-point in time. That is why an event is a special occurrence that has a certain starting-point and an ending-point in time. Especially concerning the ex ante-view, our understanding of an event differs from the prevalent view on events in event processing as a message about an occurrence which already took place (Luckham 2007).

In many approaches events are understood in terms of such a message. A single or several messages are computed and if their pattern matches a certain condition, a given action is executed. This pattern is called ECA (event – condition – action) schema.

This kind of ex-post consideration may be adequate and useful for a lot of tasks, but by striving for the goal of a holistic modeling-approach for an event-application, the ECA-schema soon meets its limits. (Anicic et al. 2008) propose the adoption of a context, thereby expanding the ECA-schema to an ECCA-schema (event – context – condition – action), what improves the situation insofar, that the execution of an action due to events does not only depend on a universal condition (rule) any more, but in addition on the actual circumstances of the system, like the information, that the same action has been executed recently due to a similar event-pattern, but did not lead to the results expected and is therefore useless at the current situation. That may prevent the system from doing the same action over and over without gaining any useful results. In this way the consideration of a context improves the power of the body of rules in an event-oriented application.

Instead of introducing a separate context like (Anicic et al. 2008), we consider *things* analogue to an object-oriented representation of a system-state.

A *complex event* is considered as an event consisting of several other events. These may take place after each other, or (partly) at the same time or even totally decoupled with respect to time.

An event is called *atomic* or *elementary*, if it does not consist of other events. A *composite* event is a complex event which is constructed by connecting events using logics.

2. Events and object-orientation

Our approach brings another technological innovation with respect to modeling and implementing software-systems. Current object-oriented programming reconstructs things as objects.
Occurrences like incidents or events are generated and implemented as (the execution of) methods of an object. In this way every occurrence - and with that every event, which presumes that something happens – is bound to a thing. In object-orientation an object reconstructs a thing that may have occurrences at its disposal. A reconstruction of an occurrence without binding it to a thing is not possible in line with object-oriented modeling and programming.
We use the terms "object" and "item" synonymously. Both occurrences and things are considered as objects. Occurrences can be events, actions, processes and so on, like our definition of an event says. That is how our approach differs significantly from the object-oriented approach.
The introduction of an event as defined above is centric to our approach. In opposite to object-orientation, an event can be modeled as a first-class instance without the necessity of binding it to a thing. This is quite important, since our approach takes events into account that will or might happen, an ex ante view on events.
In part five of this paper we will see that events can be used to model consequences of the occurrence of a pattern of events, which includes the rules in which conditions are formulated and the resulting actions.
An event itself can own and use objects (e.g. occurrences), so called event-things. To be clear: we do not want to abolish the modeling of things as objects. Things as objects (and independent from events) will do their part in event-oriented applications as well. That is why we propose to put event-entities on the same level as thing-entities, both reconstructed as objects.
It is reasonable to use things to model the organizational structure of a system and to use occurrences or events to model the operational structure of a system. The reconstruction of things and events on the same level is shown in the rational classification of objects in figure 1.

3. Events in a classification of objects

In figure one, on the lowest level of the classification, there are occurrences and things on an equal footing. An event is a special kind of occurrence. Further, one can see, that both (possibly equipped with qualities and connected with other components or objects) are objects. Further, both things and occurrences can be *carriers* of certain *qualities*. There are qualities that refer to a certain singular object and qualities that accord to the concept of an object, which means that all objects that are embraced by that concept per definition do have that quality.
A *component* is a singular occurrence or thing equipped with qualities.

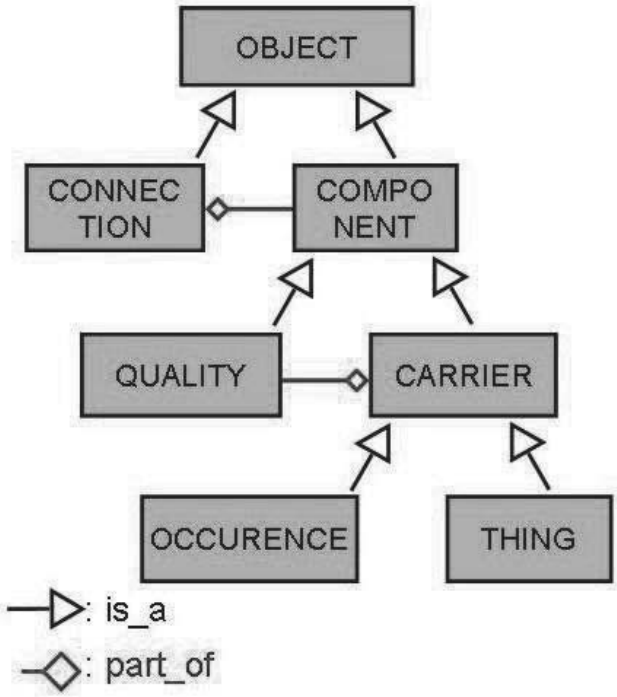

Figure 1: A rational classification of objects

If a component is self-contained, it is considered as an *object* without being connected to other components or objects. If that is not the case, it can be connected with other components.
Connections enable us to construct new objects from components, which in this context may be other components – occurrences or things equipped with qualities – or even other objects. This makes a nested construction of objects possible.
There are several different types of connections, which can be divided into classifications and compositions (Horrocks 2005). Compositions can further be divided into aggregation (cover aggregation, Codd 1979), which are used to reconstruct part/whole relationships, and connexion (Cartesian aggregation, Codd 1979), that are used to reconstruct connections that are concepts on their own, like the concept "father" which is a connection between two persons. Connections via classification involve on the one hand subsumption, which are important with respect to the consideration of different language-levels (see part four of this paper) and inclusion. Inclusions are used to reconstruct species/genus-relationships. Inclusion empowers us with

abilities similar to those of inheritance in object-oriented programming- and modeling-languages. Besides the mentioned possibilities to connect components, they can be connected using logic, which is described in part five of this paper. Following the EPTS glossary, we call events that are constructed by connecting other events using logics as *composite events*.

4. The concept "event"

In our language-critical approach, the distinction between things and events becomes apparent by the existence of two different types of concepts. On the one hand there are thing-concepts and on the other hand there are event-concepts. According to (Frege and Beaney 1997), a concept is a function whose value is always a truth-value. If the concept embraces the given object (thing or occurrence), that truth-value is "true" and "false" otherwise.

Considering the concept "dog", its function is "dog(x)" and its truth-value depends on x being an object that is a dog or not. Dog(Fido) is true, if Fido is a dog and false otherwise. On a language-level, this fact is either expressed by the sentence "Fido is a dog" or by the negative sentence "Fido is not a dog."

Analogical, there are functions whose values are always a truth-value for event-concepts. The truth-value of the function "Writing a paper(x)" is true, if the occurrence x is an instance of the concept "Writing a paper" and false otherwise. The difference between the functions of thing- and event-concepts is the way they are reconstructed on the language-level. Considering two events A and B, with A being embraced by the concept "Writing a paper" und B not, the sentences that are reconstructed are "Event A does occur as Writing of a paper" und "Event B does not occur as Writing of a paper".

At this point the existence of two different language-levels becomes apparent: the schema- and the instance-level. On the *schema-level* there are concepts, which possess an intension and an extension. The intension of a concept is the schema that describes and defines an object (thing or event). The extension is the set of all objects (instances) embraced by that concept. This two-layered language allows us to reconstruct events similarly to things in an object-oriented programming-language. By defining schemas of events as a concept, the universal aspects of an event are modeled, e.g. that the result of an event "Writing a paper" always is a thing, namely the written paper.

Singular aspects are reconstructed via the instances of the concept. Such singular aspects may be the actual point in time, an event started or ended.

So, there is a strict distinction between the schema of a concept and the instances of a concept. In the same way, we differentiate between the schema and the instances of qualities and connections. Note, that the instances of a quality or a connection cannot be present without any carrier respectively component.

We mentioned earlier, that things may be possessed by an event, respectively be created by an event. Just as well, one can imagine events that lead to the destruction of things, like a fire in a depot or the erasure of entries in a database. Further, events can be connected to things like an event taking place in a building or room. But still, we have to keep in mind that while two objects, no matter what kind of, are connected; they still are two independent distinct objects or components. They do not have to be connected in a universal way. The event "dance" may take place in a disco, but it does not have to. A dance can take place in a gymnasium or outdoors as well.

That is why we propose the separate modeling of events and things and their schemas. Singular instances may be connected in a certain manner, it may even not be possible to decouple these instances (Sowa 2002), but their concepts and schemas should be separated. With this kind of modeling, the obligatory connection between things and occurrences can be overcome.

Methods of objects that represent an action that can be taken by or with the object should be modeled as independent events and the respective components should be connected using logics. Using this approach of connecting events and things, it is possible to add more semantics to the model. This approach enables a holistic view and modeling of an application system, concerning the modeling of "real world"-events as well as the modeling and design of software. Further, the separation of events and things allows us to reuse events and actions (even taking methods of an object-oriented object into account) more easily, since they do not have to be bound to a certain object.

Eventually we have to state, that the broadly used view on an event as a message respectively a trigger for something is narrower than our proposal, but we explicitly do not exclude it from our model, since our approach can easily be used to model that kind of events, if the shown advantages are not needed. But our approach does not restrict the modeler to that kind of events, in addition it allows to model services, procedures, commercial transactions etc. The model is rich in semantics and can be designed in a differentiated way.

5. Logics to connect and evaluate events

Figure 1 shows that complex events are generated by connecting events. Besides the connections shown in part three of this paper, that yield complex events, events can be logically connected. Logical connections between events can for example be of a temporal nature, like event A immediately taking place after event B, or both event A and event B taking place at the same time. In addition to a temporal aspect, two events might for instance be

connected with respect to the location where they take place. In this way a composite event may consist of two temporally linked events that take place at the same location or maybe two certain different locations that might for example be close to each other.
In addition, more different logical connections of events are possible, depending on the logics applied when modeling the events. Events connected by logics are called composite events.
With aid of logics, events can be connected in a way that allows complex analysis in terms of "complex event processing". The proposed modeling-language is organized in a modular way, so the user can choose which logics to use and does not have to care about all available logic-modules.
The logics considered in our approach go beyond classical Boolean logic, in the sense that not only truth-values like "true" and "false" are distinguished, but in addition modalities like "necessary", "possible" and so on.
In (Ortner and Schneider 2008) temporal and modal logic for describing events and their relationships have already been introduced. Temporal logic allows us to take points in time, durations and concepts like concurrence into account when modeling events. The modal-logics-module enables us to grant attributes to events like "necessary" and "possible" (Kamlah and Lorenzen 1984).
Both, the temporal- and the modal-logics-module, have been introduced. In addition to these, we now introduce further logic-modules

5.1 Normative logic

Besides events that are necessary for other events to take place, there can be events that are not obligatory for a complex event to occur but may be desirable or forbidden. Normative logic deals with attributes like "bidden" and "forbidden" (Lorenzen 1984).
Take a look at an example: After the event "order" took place, the event "delivery" is aspired. Certain events have to occur respectively have to have already occurred to make the event "delivery" reachable. These might be the event "production" of the ordered goods, the availability of a delivery truck and a driver, and maybe even the event "payment". One can think of a scenario, in which all these events are necessary for the event "delivery" to become reachable. In addition to these events, the company policy demands a quality inspection before the goods are delivered to the customer. If the framework given by the used temporal logic allows the event "quality inspection" to take place and leave enough time for the event "delivery" to take place on time, the event "quality inspection" will be executed.
But if there is not enough time and if the event "quality inspection" would take place, the delivery would be late, the negative possible (unnecessary) event "quality inspection" may be omitted.
Reasoning according to an example like this one can be automated by using the architecture proposed in part six of this paper.

5.2 Practical logic

In our example for the usage of normative logic, we used a term which belongs naturally to practical logics (Lorenzen 1985). We stated that the event "delivery" would be reachable respectively not reachable under certain circumstances. Practical logic is engaged with attributes like "reachable" and "avoidable". At this point one can see how our approach takes future into account.
One can picture a scenario opposite to the one we used describing normative logic, in which events are "necessary" to reach another event but did not yet take place. In the context of a supply chain, the production of goods might be initiated just when the goods are ordered or if the stock falls below a certain amount of them. In this case, the system reasons that for the event "delivery" to be reachable, the event "production" has to be initiated first. This reasoning uses events, logics, but also things like a stock etc. to decide, what to do. Taking the current circumstances into account, according to (Anicic et al. 2008) the system can reason, that due to the low stock level there is need to initiate the event "production" to make the event "delivery" reachable.

5.3 Topological logic

A further logic relevant for the treatment of events is the topological or positional logic (Kremer and Mints 2005), which gets used in our approach with respect to its importance in terms of a locative logic (location-logic). This kind of logic is especially important when considering events, since an event (which takes place at the occurrence-level) is always singular in time and space. It enables us to relate an event with a location (source) and to relate the location of an event with the locations of other events. A location can be given in terms of coordinates or by a discrete value, like a room number or the identification of a building.
In our example, locative logic can be used to model a distance selling company that owns several storages and possibly more than one production facility. If an "order" event takes place at a certain location and the stock in the storage next to the event's location is too low to satisfy the order, it has to be reasoned, what option to take. One option might be to initiate a "production" event close to the storage with low stock; another might be a delivery of the desired goods from a distant storage to the one having insufficient resources. It might even be possible to take the goods for the customer from a different storage.

6. An event-oriented architecture

In this section we present an architecture that forms the basis of the presented modeling-concept. It is depicted in figure number two.

The lowest tier of the architecture is the *integration tier*, which offers integration of external systems like databases. The task of this layer is to encapsulate objects (both events and things) from external systems in objects that conform to the presented object classification, either things or events. The layer contains data-resource-adapters, which make connection to databases and other systems that offer things-schemas and make information about thing-instances available. It further contains application-resource-adapters that are able to connect to other applications and thereby makes services and applications available as event-schemas and event-applications.

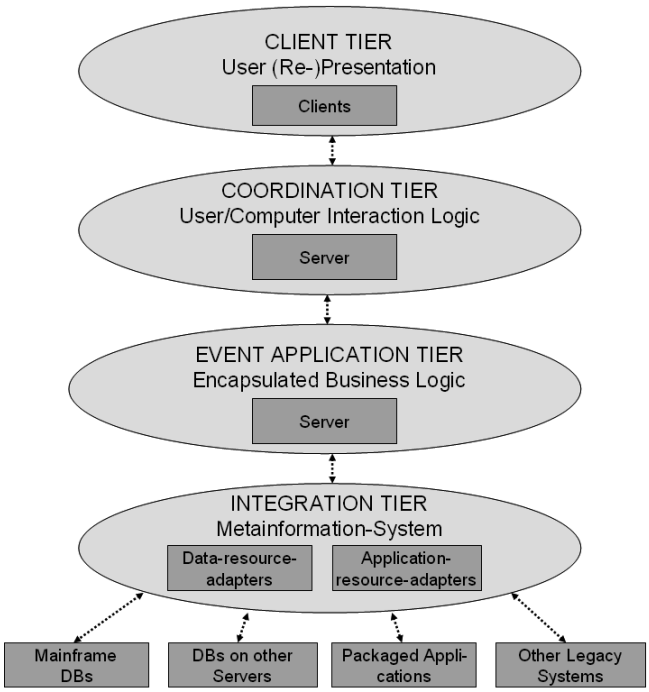

Figure 2: An event-oriented architecture

On the next higher level is the *event application tier*. In this layer the applications of the event-schemas are executed, which is why the business logic is located in this layer. The business-logic is contained in event-applications (of complex or atomic nature) which are coordinated, controlled and orchestrated by the coordination tier.

The next layer is the coordination tier. Here the event- and thing-schemas are located, as well as the connections between singular events and things. The instances of events, thinks, qualities and connections are managed by this layer by a higher-level-language to be able to coordinate and control them. That is why this tier is responsible for controlling operations in an event-oriented system. Logical connections and rules are deposited in this layer, the reasoning is done in this layer and if necessary new events respectively event-applications are initiated by this layer.

If there are any logical connections between sub-events of a complex event, for instance temporal or modal ones, this tier decides automatically, what has to be done further and initiates, waits for or skips events or event-applications. If the system is not able to do all the reasoning automatically, for example because of a lack of rules, the system commits the decision to be made with respect to the current event- and event-application situation to a user by using the client tier for interaction. That may happen since we do not just use Boolean, but in addition more complex logic-combinations. Their examination might yield a result that is not automatically decidable, for the result might not just be a “true” or a “false”-value.

The top-layer of the architecture is the client tier. This tier is responsible for tasks like interaction (both with users and other automated systems) and data presentation. The quite complex topic “interaction” is presented in a separate paper (Fischer, Ortner, and Sternhuber 2009) in a context of a deeper investigation of the architecture.

As depicted in figure two, the coordination tier is located on a (single) server, while the event application tier may be located on a server or it may be distributed. The client tier may be located on a server and interact with users using interfaces for interaction with software-systems or a web-interface for interaction with human users. A further possibility is the creation of “heavy clients”, computer programs which have to be installed and configured on an independent client.

The example used throughout the paper can now be applied to the presented event-oriented architecture.

On the coordination tier there is a schema of an event “order”, which contains all universal qualities and properties of an order. The schemas of the events “production”, “quality inspection”, “delivery”, “storage” and “stock-level” are stored here as well, the latter two being thing-schemas. Further, there are logical rules for each event, describing, which events are necessary, bidden or forbidden for an event e.g. “delivery” to be reachable and to be able to take place.

On the event application tier the applications as instances of these schemas are located. An actually happening order is located in this tier as a supporting application containing a specific set of qualities, like the order date, a date of delivery, the number of ordered goods and so on.

When an order takes place, the coordination tier checks which logical rules are available for an occurrence of an “order”-event. In our example the coordination tier checks, if the stocks are high enough to satisfy the order. Such information about things is on the one hand available on the coordination tier as schemas which describe the universal aspects of things and on the other hand instances of these schemas are managed on the event application tier.

Information concerning things and events that is not needed to be present in memory all the time can be accessed using the integration tier by requesting this information from a database or any other external system connected to the architecture.
Is the desired date of delivery far enough in the future, the event "quality inspection" is initiated with all corresponding event-applications. Otherwise it is omitted.
Finally the event "delivery" and all its event-applications may take place.
The main benefit of the proposed architecture is the possibility of a top-down modeling of events. In opposite to a lot of architectures that are widespread at the moment (Luckham 2007), the goal of our approach is not to look for events that can be observed, but to model events in the creation-process of an application. This way, there is no need to aggregate low-level-events into complex events by the use of agents, but complex events can immediately be modeled and integrated into the application. By the proposed unified coordination-layer, these events can be orchestrated. That is why the cause of an event can be determined easily, since new events are not created by an independent agent but by the coordination layer. If the coordination layer decides to create a new event, it knows why that decision has been made. This way causality is easy to control. Of course, this control over causality is limited by the boundaries of the architecture. Concerning events entering the systems or leaving the systems, we are faced with the usual problems concerning causality.
The main disadvantage of the proposed architecture is that it is not able to perform real-time-processing.

7. Conclusion

In this paper we presented a new approach to modeling of events using logics. The application of some logic-modules has been presented and exemplified. We further presented an event-oriented architecture that in addition to modeling is an approach to productive usage of the presented paradigm in real-world scenarios.
The approaches presented are still under development and need further elaboration. The results of that work will be published successively.
Many scenarios described in our examples can – at least in parts – be modeled and solved with currently available methods of complex event processing. The advantage of our holistic approach is the possibility to model the whole systems by using events and logic-modules. There is no need to use different languages for modeling events, rules, conditions or actions; everything can be done in a single language. Further this consistent formulation enables automatic reasoning at the coordination tier in a standardized manner. This way causality can be controlled easily.
In this tier it is possible to add new event-schemas and logic referring to them without any need for change in the rest of the application system.
The goal of our architecture approach is the extension of currently available approaches by the possibility of top-down development of event-oriented applications.

References

Anicic, D.; Sen, S.; Stojanovic, N.; Ma, J.; and Schmidt, K.-U. 2008. Contextualised Event-Triggered Reactivity With Similarity Search. In Proceedings of the 1st iCEP08 Workshop on Complex Event Processing for the Future Internet, Vienna, Austria.: CEUR-WS.org/Vol-412/paper6.pdf

Codd, E. F. 1979. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.*, 4(4): 397–434.

Fischer, M.; Ortner, E.; and Sternhuber, J. 2008. Interaction in an Event-Based Context. An Architectural Concept (Draft). Forthcoming.

Frege, G.; and Beaney, M. 1997. *The Frege reader*. Oxford: Blackwell (Blackwell readers).

Horrocks, I. 2005. Applications of Description Logics. State of the Art and Research Challenges. In *Conceptual structures: common semantics for sharing knowledge. 13th International Conference on Conceptual Structures,* 78–90 Berlin, Germany: Springer

Kamlah, W.; and Lorenzen, P. 1984. *Logical propaedeutic. Pre-school of reasonable discourse*. Lanham, MD: University Press of America.

Kremer, P.; and Mints, G. 2005. Dynamic topological logic. In *Annals of Pure and Applied Logic* 131: 133–158.

Lorenzen, P. 1984. *Normative logic and ethics*. Mannheim, Germany: Bibl. Inst.

Lorenzen, P. 1985. *Grundbegriffe technischer und politischer Kultur*. Frankfurt am Main, Germany: Suhrkamp

Luckham, D. 2007. *The power of events. An introduction to complex event processing in distributed enterprise systems*. Boston, Mass.: Addison-Wesley.

Ortner, E.; and Schneider, T. 2008. Temporal and Modal Logic Based Event Languages for the Development of Reactive Application Systems. In *Proceedings of the 1st iCEP08 Workshop on Complex Event Processing for the Future Internet*, Vienna, Austria.: CEUR-WS.org/Vol-412/paper5.pdf

Sowa, J. F. 2002. *Knowledge representation: Logical, philosophical, and computational foundations*. Pacific Grove, Calif.: Brooks/Cole