Sensyble Workshop 2010 manuscript No.
(will be inserted by the editor)

An Architecture for Interaction Event Processing in Tabletop Systems

Simon Lehmann[1], Ralf Dörner[1], Ulrich Schwanecke[2], Johannes Luderschmidt[1], Nadia Haubner[1]

[1] RheinMain University of Applied Sciences, Wiesbaden Rüsselsheim Geisenheim, Department of Design, Computer Science and Media, Kurt-Schumacher-Ring 18, 65197 Wiesbaden, Germany
[2] RheinMain University of Applied Sciences, Wiesbaden Rüsselsheim Geisenheim, Department of Design, Computer Science and Media, Unter den Eichen 5, 65197 Wiesbaden, Germany

Received: date / Revised version: date

Abstract Interactive tabletop systems are suitable to offer a wide range of input modalities. Applications developed for this kind of interaction usually have to deal with each input devices individually. In this paper, we describe an architecture for interaction event processing suitable for the specific requirements in the context of tabletop systems. Our approach for an architecture for input processing in interactive tabletop systems is meant to help making the development of complex interactions more feasible. The key part of the architecture is based on the techniques and methods of the well established field of complex event processing (CEP), which have been applied to many other problem domains before. Our approach applies CEP to input events in interactive tabletop systems.

The architecture is build out of several layers, each responsible for handling input data or events at a specific level of abstraction. The lowest level contains the various hardware devices used for interaction. The devices and their respective input capturing modules produce all the raw input events. Above this layer, the event processing takes place. Here, all events produced in the system are received and are filtered, aggregated or transformed into higher-level events using a rule-based system. The rules are provided along with the input capturing modules, which allows for a freely configurable setup of input modules and rules. The events resulting from the processing step are then passed up to the applications, which are implemented against an API that allows to issue continuous event queries. The continuous queries are stored in the event processing system which then notifies the application about any events matching the corresponding query.

1 Introduction

Interactive tabletop systems have been in the focus of HCI researchers for about two decades and gained more interest in the past years due to the development of powerful hardware for a variety of input methods. Additionally, the key benefits associated with interactive tabletops – like ease of use, intuitiveness and support for collaborative work – make them suitable for the use in the field of ubiquitous/pervasive computing, which is getting more and more important [12].

Despite the general interest in interactive tabletops and the broad research of tools and techniques for various ways of interaction, little research is done concerning the general processing of interaction events happening throughout such a system. In contrast to the traditional input devices used with PCs, applications for tabletop systems make potential use of a wide variety of input methods. Thus, the need for processing and aggregation of input events arises, which addresses the rising complexity and interdependence of the constantly generated input events by multiple input devices.

Previous approaches to the problem of integrating multiple input channels can be classified depending on the level on which the integration takes place. Integration of different modalities can happen at a feature level, a semantic level, or somewhere in between [13]. Integration on the feature level means the fusion of raw data from multiple input devices to perform recognition on the combined data. The semantic level means a very late fusion, where the raw sensor data has already been processed through one or several classification and recognition techniques, and the resulting, higher level information is then integrated. While both approaches have their benefits, our proposed architecture will approach a more late integration, as it does not require the individual devices or even sensors to be specifically built to work together.

Even though a lot of effort has been put into the design of complex event processing technology, the general engineering of such systems is still dominated by a trial and error process [10]. This makes the design of an architecture for interaction event processing in interactive

tabletop systems even more challenging, as currently no design patterns exist.

2 Related Work

While the specific problem addressed in this paper – the general interaction event processing in interactive tabletop systems – has not been in the focus of research yet [5], similar problems can be found in the fields of multimodal user interfaces and complex event processing. Multimodal user interfaces deal with the aspects of human-computer interaction with respect to multiple different in- and output modalities. In the field of complex event processing (CEP), the processing of events in (usually large-scale) event-based systems are investigated.

The CAIP Center [11] is meant to provide a multimodal human-computer interaction system that focuses on several pre-defined in- and output modalities, such as tactile input with force-feedback, speech recognition, or gaze tracking. The proposed architecture is very focused on the specific setup described in their work and is more concerned about the application specific fusion of inputs. As many earlier works have shown, systems with multimodal inputs require a more sophisticated input processing architecture than the more traditional inputs like mouse and keyboard.

Flippo et al. created a framework for rapid development of multimodal interfaces [4]. Within their framework, they also had to deal with processing input from different devices. They used an agent-based approach for analyzing all input modalities to find out what the user actually wants to do. While this might be a general approach (like the Open Agent Architecture [9] or the Adaptive Agent Architecture [7]), their work focuses on speech input that is accompanied by other types of input to resolve ambiguous input situations. Even though their approach seems to be well suited for the specific setup centered at speech input, it is questionable if the architecture can be transferred to other kinds of setups.

The Embassi system [3] consists of two modules: a fusion component that merges speech, pointing gestures, and GUI input, and a presentation planning component that decides which modality should be used for the output. The architecture of Embassi is very well suited for setups that allow for multiple input *and* output modalities, where the output should depend on the input modalities chosen by the user. Even though other approaches more directly dealing with interactive tabletop systems, like STARS [8], indicate that the selection of output channels based on their modality is also useful in these systems, it is of lesser concern to our approach, as it only addresses the input processing. Additionally, the main problem Embassi is trying to solve is to semantically analyze the users multimodal input, which is also not in the focus of this paper.

More focused on the complex event processing is the RACED middleware [2]. It provides a complete infrastructure for the detection of complex events from several, related events. It allows the declarative description of complex events via an event definition languange. The events are then detected in the simple events produced by various sources by different service brokers connected by an overlay network. This approach also makes the system highly distributable. Even though the system is more focused on more general sensor networks or other kinds of event-based systems, it might also be applicable to interaction event processing.

An architectural approach directly focusing on tabletop systems is ZOIL and Squidy [6]. While the former is an UI toolkit for building tabletop applications, the latter deals with the actual processing of input events from various input devices. Their approach features a graph based event filter system, which processes the incoming events. While such a system works on a lower level of event processing and can be even used for the actual input tracking, it does not address the needs of more complex event processing.

3 Requirements

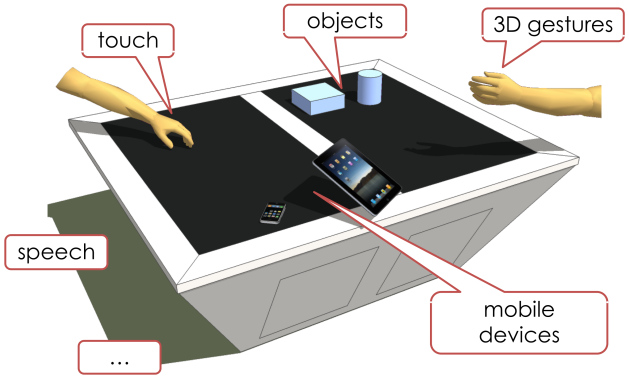

Fig. 1 Typical examples of input modalities in interactive tabletop systems.

As shown in figure 1, a typical tabletop system consists of various input modalities, such as touch, tangible user interfaces, or mobile devices. This even can be extended to three-dimensional input like gestures above and around the table or speech input. Just considering these examples of input in a tabletop system, the need for a unified interface arises. Applications need to access the different inputs and react to users interactions.

A naive approach to solve this problem would be to directly use the various APIs and SDKs provided by the devices themselves, but this quickly leads to a highly coupled system, which has to change whenever on of the input devices change. This problem might be alleviated a little by introducing some kind of wrapping layer that encapsulates the devices, but it still does not scale well

with an increasing amount of input devices. Additionally, it only shifts the coupling and the necessary changes to a different layer. In order to have a more flexible and more scalable solution, an architecture is needed, that enforces a stronger separation of the hardware and the application.

The coupling between application and devices results on the one hand from the specific event model employed by an input device, and on the other hand from the specific API used for accessing the events produced by an input device. The event model defines the structure and the properties used for communicating the input performed by a user to the application, and is usually defined by the specific characteristics of the input device or modality. Based on the event model, the device specific API is defined to access the specific event types and allows to receive events as they are produced. Changes to the device or class of devices have to be reflected in both the event model and also the API that is based on it, and thus results in changes in the applications that make use of it. This makes it also hard to extend the tabletop system to new input devices or modalities, because of the fixed event models and APIs.

Apart from the problem of tight coupling, combinations of different input devices usually demand a aggregation or otherwise pre-processing of events in order to enable more higher-level interactions. As this cannot be done in the devices themselves, it has to happen on a higher layer. If it is done inside the application, it does not allow this kind of processing to be easily reused and thus makes the development of applications for interactive tabletop systems harder. Thus, such kind of processing has to be performed on a layer between devices and applications.

An architecture for interaction event processing needs to address each one of these problems. It is required to provide

- an independent event processing layer,
- an event type independent API,
- and a unified event model.

4 Architecture Overview

Our approach to unify the integration of various input devices, and more specifically different interaction modalities, is based on the separation into several layers, which are depicted in figure 2. Before the details of the two main layers – event processing and the interaction API – are described, a short overview of the layers and their meaning is given.

Input devices are all potential devices of a interactive tabletop system. This layer does not define any kind of API or model which has to be obeyed by the devices, as it would make the implementation of this architecture impossible with existing devices.

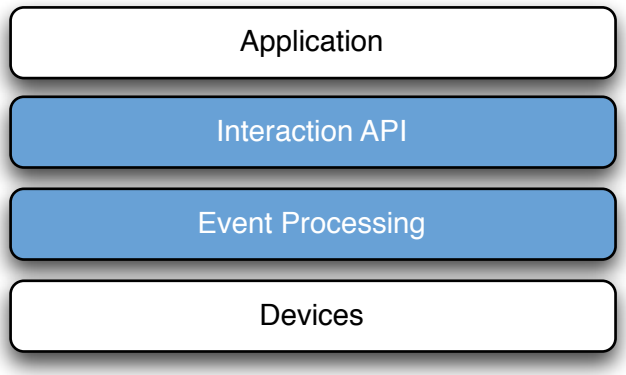

Fig. 2 Layers of the proposed architecture.

Event processing contains all the modules and operations necessary to collect the devices specific events from the available devices, convert them into the common event model used by the system, and apply any processing of the events that can be done in an application independent manner.

Interaction API allows the applications to have a fine grained access to the events produced by the event processing layer. It defines a query-based interaction API, that allows the applications to issue continuous queries agains the stream of events, which are matched against the queries' criteria, and are delivered to the respective parts of the applications.

Applications are implemented on top of the interaction API. This ensures that applications only make use of the unified event model and only access and receive events by issuing continuous queries.

5 Event Processing

The main reason to have a separate layer for processing of interaction events is to be able to process events independently of devices and applications. It also allows to make interaction event processing re-useable across different applications, and furthermore to have the processing only happen once instead of for each application. Figure 3 shows the components of the event processing layer. It consists of an event collector, which is responsible for collecting all events produced by the available input devices, and a rule engine, which further processes the events by applying the available rules.

In the proposed architecture, event processing means that incoming events are filtered, aggregated or modified based on pre-defined rules provided by input modules. Processing of events allows for example to generate higher level events from simpler, lower level kind of events. For example, two touches moving side-by-side in the same direction can be aggregated to a higher-level event representing a scrolling gesture. Furthermore,

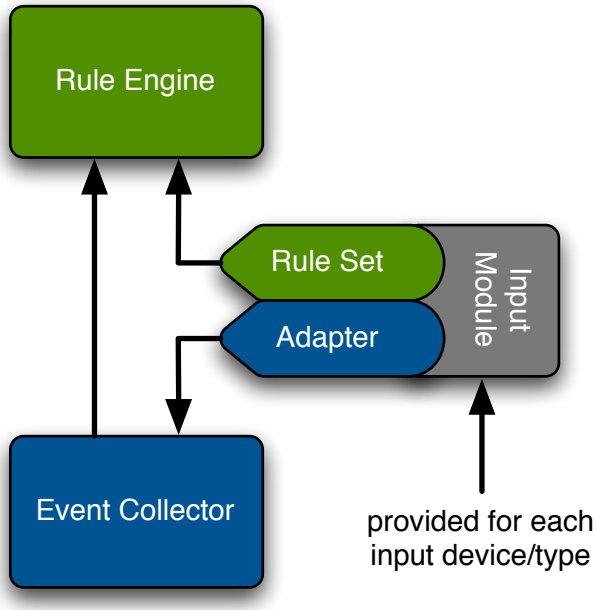

Fig. 3 Architecture of the input event processing layer.

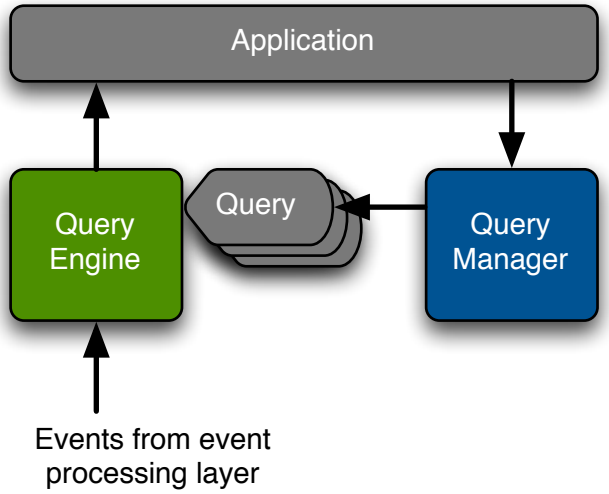

Fig. 4 Architecture of the interaction API layer.

events from different types of input devices can be combined, which allows to generate information which could not be obtained by the input devices themselves.

Input modules are specific to a certain device or class of devices and contain an event adapter and a set of rules. The event adapter works as a bridge between the device specific APIs or SDKs and the more generalized event processing. It contains the necessary functionality to retrieve the raw events from a specific input device and to feed them into the rule engine for further processing. The rule engine then makes use of all the rule sets of the available input modules, which describe how events of a certain kind of input should be processed.

6 Interaction API

To allow for a flexible and extensible access to the various events flowing through the system, the architecture provides an API based on continuous queries [1]. These queries are issued by the applications interested in certain events and are continuously matched against any events. Any event matching one or more queries is forwarded to the respective event consumers registered when the query was issued. The queries itself allow for an application defined specification of interesting events and thus make it possible for developers to formally specify any input allowed in their application. As continuous queries also allow for time based queries, they are also suited for defining application specific input patterns – usually called gestures for finger or hand interaction – like distinguishing between a long-tap and a short-tap gesture.

The benefit of this kind of API is twofold: it operates on the general event model used throughout the system and is therefore not bound to any pre-defined types of events, and it allows application developers to state more precisely which kind of events they are interested in, thus reducing the amount of event processing inside the application code. By choosing an API that is much more dynamic and open in its nature than traditional event listener concepts used by many UI toolkits, the whole architecture is made more suitable for broad and long-term adoption in interactive tabletop systems.

By having a declarative API that lets the application just state what it needs, this architecture design also allows optimizations of event processing. As the query engine responsible for the execution of queries is completely separated from the application, all applications developed against this API can benefit from future improvements made to the query engine implementation.

7 Conclusion and Future Work

The architecture presented in this paper for the input system of interactive tabletop systems is tailored to bring several benefits over the currently used approaches in this field. First and foremost, it enforces a stronger separation between input devices and the applications running on a tabletop system, especially by requiring a unified event model across different devices. Additionally, it allows for a more extensible and scalable system, which is especially important to tabletop systems, as more and more different input modalities are added to these systems. Also, the additional event processing layer relieves applications from lots of recurring pre-processing tasks and encourages the development of more application independent – and thus re-usable – event processing. Fi-

nally, the query-based API allows applications a more fine-grained access to all the numerous events occurring in the system.

As future work, a prototypical implementation of this architecture will be necessary to gain insight about the performance and behavior of a concrete implementation. As performance, and more specifically latency, is an important aspect of interactive systems, the impact of more complex event processing has to be determined.

Besides the prototypical implementation and resolution of practical issues, the aspects of distributed tabletop systems and how they might be addressed by the proposed architecture are worth further investigations. Distributed tabletop systems include setups where multiple tables are used together in one location, or where multiple tables are spread across different locations to be used in remote collaborations settings.

References

1. Chen, J., DeWitt, D. J., Tian, F., and Wang, Y. NiagaraCQ: a scalable continuous query system for Internet databases. *SIGMOD Rec. 29*, 2 (2000), 379–390.
2. Cugola, G., and Margara, A. RACED: an adaptive middleware for complex event detection. In *Proceedings of the 8th International Workshop on Adaptive and Reflective MIddleware* (2009), ACM, pp. 1–6.
3. Elting, C., Rapp, S., Möhler, G., and Strube, M. Architecture and Implementation of Multimodal Plug and Play. In *Proceedings of the 5th international conference on Multimodal interfaces* (2003), pp. 93–100.
4. Flippo, F., Krebs, A., and Marsic, I. A Framework for Rapid Development of Multimodal Interfaces. In *Proceedings of the 5th international conference on Multimodal interfaces* (2003), pp. 109–116.
5. Hinze, A., Sachs, K., and Buchmann, A. Event-based applications and enabling technologies. In *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems* (2009), pp. 1–15.
6. Jetter, H., König, W., and Reiterer, H. Understanding and designing surface computing with zoil and squidy. In *CHI09 Workshop-Multitouch and Surface Computing* (2009), Citeseer, pp. 1–5.
7. Kumar, S., and Cohen, P. R. Towards a Fault-Tolerant Multi-Agent System Architecture. In *Proceedings of the fourth international conference on Autonomous agents* (2000), ACM New York, NY, USA, pp. 459–466.
8. Magerkurth, C., Stenzel, R., Streitz, N., and Neuhold, E. A multimodal interaction framework for pervasive game applications. In *Workshop at Artificial Intelligence in Mobile System (AIMS), Fraunhofer IPSI* (2003), Citeseer.
9. Martin, D., Cheyer, A., and Moran, D. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence 13*, 1 (Jan. 1999), 91–128.
10. Paschke, A. Design Patterns for Complex Event Processing. In *Proceedings of the 2nd International Conference on Distributed Event-Based Systems* (2008).
11. Shaikh, A., Juth, S., Medl, A., Marsic, I., Kulikowski, C., and Flanagan, J. L. An Architecture for Multimodal Information Fusion. In *Proceedings of the Workshop on Perceptual User Interfaces (PUI'97)* (1997), pp. 91–93.
12. Tse, E., Greenberg, S., and Shen, C. Motivating Multimodal Interaction around Digital Tabletops. In *Video Proc. ACM CSCW Conf, Computer Supported Cooperative Work* (2006), pp. 6–7.
13. Turk, M. *Multimodal Human-Computer Interaction.* Springer US, 2005, pp. 269–283.