Towards Integration of User Interaction and Context Event Processing in Intelligent Living Environments

Simon Lehmann Jan Schäfer Ralf Dörner Ulrich Schwanecke
Design Computer Science Media Department
RheinMain University of Applied Sciences
Wiesbaden, Germany
{simon.lehmann,jan.schaefer,ralf.doerner,ulrich.schwanecke}@hs-rm.de

Abstract: Event processing plays a significant role in the current development of intelligent living environments. It ranges from processing of information produced by a magnitude of sensors to gain insight into the activities of the inhabitants on a more global scale, to the processing of immediate and rather short-lived events of user input on and around interactive systems embedded in common household furniture like tabletops or tablets. Based on the work conducted separately in those two fields, we found that the still evolving field of complex event processing (CEP) provides the methods and tools to handle those distinct use-cases equally. Especially the application to interactive systems, while being novel and uncommon, is well suited and further shows the broad applicability of CEP. The comparison of the two application fields shows that, even though the events occurring in them are distinguished by their intention, commonalities do exist and provide integration points. Furthermore, the integration of those applications within the context of smart homes allows to provide demand-oriented resource management, which realizes self adaptation and control.

1 Introduction

Intelligent living environments, also known as smart homes, aim at providing its residents with useful services and assistance for everyday activities. These services cover different aspects, such as health monitoring, energy management, environment control, security, communication, or productivity tasks. Social and ethical challenges aside, major technical challenges in this field of research are modeling and prediction of the activities and actions of the residents, multi-modality of interaction with various kinds of computer systems, and intelligent monitoring of systems components [HSB09].

On a technical level, these applications are all based on the processing and interpretation of context events happening throughout the system and its environment. Be it events produced by sensors for monitoring of the residents location, events coming from user input devices, or events of the internal state of the system. So looking at the way event processing is done in different areas of intelligent living environments, which are usually treated separately, allows to gain insight about the individual requirements, but also the commonalities shared by all parts of the system.

In this paper, we explore the possible points of integration of two areas of event processing

in an intelligent environment setting. We identify those areas, where each application field can benefit from the other, but also define the central characteristic which distinguishes both fields from each other, namely the intention of the processed events. Additionally, we introduce a *demand-oriented resource management* and discuss how it is made possible within that context.

We base our work on two concrete systems. The first deals with systems management. The second employs complex event processing to handle the interaction with more complex input devices of interactive tabletop systems, which is a rather novel and uncommon field of application. Even though the two systems are not interacting with each-other and were developed separately for their individual purpose, both use complex event processing methods and techniques. More specifically, both use Esper [Esp11] as an underlying technology, which provides the additional benefit of having similarities on a technical level.

In the following, we will first give a brief overview of what complex event processing is and how the general methods and techniques are provided by a concrete implementation, Esper. We then proceed to describe the two areas of application, and how they relate to the context of intelligent living environments, in more detail. Individual requirements, challenges and results are presented with each application. After that, the distinguishing characteristic of both application fields is defined in more detail, followed by a discussion of possible points of integration between those systems. Finally, we present our proposal of a demand-oriented resource management and conclude with a short summary and outlook on future work.

2 Complex Event Processing with Esper

Complex event processing means the detection, analysis, and general processing of correlated raw or simple events, which results in more abstract and meaningful complex events [Luc02]. As a very general term, it covers a large field of techniques and methods for detection of relationships or patterns of events, event abstraction, modeling of events and event hierarchies, and abstraction of the whole process of event processing. It builds upon the concept of the event driven architecture (EDA), in which loosely coupled components communicate by emitting and consuming events [LS11]. In an EDA, any notable change of state or thing that happens is considered an event, which is propagated to all components interested in them. The components evaluate the information of the events and take any necessary action, which might include generation of further events [Mic06].

The software library *Esper* provides a general purpose component for complex event processing in Java or .NET. It is based on the principle of *continuous queries* [TGNO92], where processing, filtering and dissemination of events happens by querying the engine, which continuously tries to match and process events according to the active queries. Queries are formulated using the *event processing language* (EPL). The EPL is a SQL-like declarative language, which supports selecting events from streams, conditions, sliding windows (time and length based), aggregation functions, joins, pattern matching, insertion into event streams and many other features known from relational databases (examples of

EPL queries are given in section 3.2.2).

The engine provides two mechanisms for querying events. The first allows full processing and manipulation of event streams in the engine and is available by normal EPL statements. The second method allows notification about events matching a specified *pattern*, which is expressed by the pattern syntax of the EPL. While the second method is in fact a subset of the first, it has the benefit of a reduced syntax and also remove some overhead, as they are implemented using state machines only.

From a software development point of view, Esper allows to extend its functionality in many ways (e.g. custom sliding windows or aggregation functions), which allows using it in novel contexts and use cases. It also does not define a specific event representation or how processing of events is executed, i.e. single- or multi-threaded.

3 Event Processing in Intelligent Living Environments

3.1 Home Service Platforms

With increasing integration of distributed systems in living environments, the amount of generated data that has to be processed and interpreted by home service platforms is also on the rise. Ambient sensors collect, (pre)process and forward large amounts of data generated by inhabitants and their environment including but not limited to vital signs, deduced activities, device states and environmental parameters such as temperature or humidity. Together, these values represent the *context*, which is needed to develop context-aware, adaptive applications for the platform.

3.1.1 Information Integration and Deduction

Applications in intelligent living environments rely on context information to be able to adapt dynamically to the changing user's needs or preferences. Unless an application is very simple (e.g. clock or weather display), it requires deduced information, which can be gained by combination and interpretation of events created by the platform's components (devices and software). The resulting deduced information represents more abstract information of higher value for application development (e.g. a recognized inhabitant activity). If the platform is able to produce and provide this abstract knowledge for applications, it removes the burden from applications to calculate it individually.

The more heterogeneous the systems in an intelligent living environment are, the harder it becomes to interpret and relate the events generated by them. This can be solved by manually mapping input and output of each to be integrated system to each different system or application. A much more elegant approach to solve this problem is to integrate all context producing and consuming components into a common context model, a *context ontology*, which is managed by the service platform. This allows to semantically integrate and prepare information from arbitrary sources for consumption of context data by applications.

An approach for this context model has already been presented in [Sch10], which uses the *Web Ontology Language* (OWL) [Gro09] for ontology modeling. Here, the sub-ontologies of each system contributing to the context model have to be based and integrated with a common basic ontology.

A common context model offers additional benefits. Just like arbitrary context providers or consumers, *context processors* such as CEP engines can use the information stored in the model to deduce high-level information easily, which itself can become part of the model and, thus, be consumed by components of the system.

3.1.2 Information Filtering and Extraction

Creating and managing a context model is not cheap in terms of required processing power, as incoming and outgoing information has to be imported and exported constantly, which always requires information mapping of some kind (e.g. from OWL/XML to some transport protocol or to Java). Some sensors or applications produce large amounts of raw data, from which only the *right* information is required. As the raw data should not become part of the context model (to prevent bloat and unnecessary computational effort), this data has to be preprocessed. Often, the producing component is able to do this itself, but this is not always possible. Then, a CEP engine is used as *context preprocessor*. Instead of relying on information from the context model as input, it receives input streams directly from the data sources. In this case, only the processed output becomes part of the model and, thus, can be used by other applications.

3.2 Interactive Tabletop Systems

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 [TGS06].

Despite the general interest in interactive tabletops and the broad research of tools and techniques for various ways of interaction, the general processing of interaction events happening throughout such a system has not received much attention. Traditionally, input events are handled by some abstraction layer of the operating system, which provides an event dispatching mechanism for user interface toolkits. Events are typically processed by an event loop, i.e. a continuously running process which takes any occurring event as their input and publishes them to one or multiple subscribers. Besides the routing of events from their source to the appropriate user interface components, no additional processing takes place. Any complex event processing which extracts higher level information from events are provided by specialized frameworks on the application level, e.g. for gesture detection.

In contrast to the input devices used with PCs (e.g. keyboard and mouse), applications for interactive tabletop systems make potential use of a wide variety of input methods. Additionally, also other interactive systems do make use of more input devices as their use cases increase. 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. Based on a previously described architectural approach [LDS+10], we propose a unified tabletop input layer (UTIL) to provide a middleware for processing of interaction events in tabletop systems. At its core it employs Esper for all event processing and querying tasks. The general architecture is implemented in large parts by the Esper engine, with additional components specifically built for the purpose of processing interaction events.

The collection of events from the various event sources is implemented by device dependent collectors, one per input device – an input device being any programmatically accessible source of input events, which does not necessarily correspond to a physical device. The collection process is handled individually for each device, though most use some OSC/TUIO-based protocol for communication. The collectors are responsible for producing the raw event objects, which are then fed into the processing engine. The rules for processing of simple input-events sent by the collectors to complex events are realized as EPL queries that put new or combined events into the event stream.

The interactive applications running on the tabletop system subscribe to events through a centralized query manager, which is provided by the middleware. Applications can connect to it and use it transparently to issue event queries. The application queries are based on the EPL pattern language, which are a subset of the full EPL. They allow for full specification of patterns of events and the issuer of the pattern gets notified whenever an event matches the given criteria. Application queries are fed into the same engine instance responsible for the processing of the rule queries described earlier. These are two design decisions which are related to each other: Giving application developers the ability to use full EPL queries would have two drawbacks while providing only little benefit of enabling applications to use the full potential of the event processing engine. The drawbacks, however, are two-fold. First, writing full EPL queries is much more complex than EPL patterns, thus imposing additional complexity on the application developers. Second, it also enables applications to directly interfere with the processing of the events, which is shared by all applications running concurrently. Thus, giving application developers only a limited subset of the full EPL for their queries, it enforces a true read-only access to the processing system and relieves them of quite some complexity which should not be needed on the application layer.

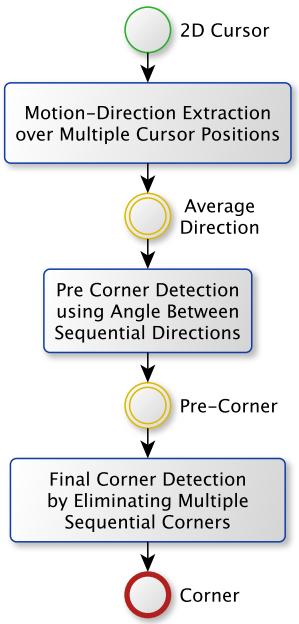
3.2.1 Processing of Spatial Data

Interactive systems are usually comprised of one or several input devices or sensors to allow users of the system to manipulate artifacts in the system. Most of this input deals with spatial information such as the position of a device itself (e.g. a mouse), body-parts of the user (e.g. fingers or hands) or other objects. When processing events mostly consisting of such spatial data, tools specifically designed for this purpose are required. Processing

events based on two or three dimensional spatial data has mostly been done in the area of geo-information processing [Lip09], but also in multimedia communication systems [GYJO10]. Esper itself only supports time or scalar value based processing of events, i.e. events can be processed depending on their temporal relations or based on the amount of events or other scalar valued attributes. Thus, we had to provide a set of single-row functions, aggregation functions, and data window views has to be provided as extensions to Esper to support processing of spatial data in two or three dimensional space.

3.2.2 Gesture Detection

The main purpose of UTIL is to find patterns and correlations of inputs produced by the users. Simple examples include the detection of a two-finger scroll gesture or a multi-finger pinching gesture. These use only one input device and thus also one modality. More advanced examples are correlation of input from different devices, such as multitouch and three-dimensional hand tracking above the table for determining which touches belong to which hand.



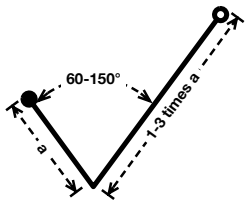

```
INSERT INTO AverageDirectionEvent
SELECT ISTREAM * as cursor,
               avgDirection(position) as dir
FROM Cursor2DEvent.win:length(LENGTH)
GROUP BY cursorId

INSERT INTO PreCornerEvent
SELECT cursor.cursorId AS cursorId,
       first(cursor.pos) AS pos,
       first(dir) AS firstDir,
       last(dir) AS lastDir,
       (last(dir)).angle(first(dir)) AS angle
FROM CursorDirectionEvent.win:length(LENGTH)
GROUP BY cursor.cursorId
HAVING leaving()

INSERT INTO CornerEvent
SELECT cursorId, pos, angle
FROM PreCornerEvent
MATCH_RECOGNIZE (
    PARTITION BY cursorId
    MEASURES A[0].cursorId AS cursorId,
             avgPosition(B.pos) AS pos,
             (CASE WHEN avg(B.angle) < 0
              THEN min(B.angle)
              ELSE max(B.angle) END) AS angle
    PATTERN (A B+ A)
    DEFINE
        A AS abs(A.angle) < ANGLE_THRESHOLD,
        B AS abs(B.angle) >= ANGLE_THRESHOLD
)
```

Figure 1: EPL queries used for detecting corners in 2D strokes drawn with a finger.

A more complex example for interaction event processing, is a two dimensional touch gesture detection based on strokes drawn on the surface of a table. It should be noted that, while the main goal of UTIL is to enhance processing of events from multiple input devices, it also allows to provide complex event processing for a single input device.

The general algorithm behind this gesture detection is based on the ideas presented in [WX10] and [WEH08]. First, corners in strokes drawn with a finger on the surface are detected. Based on the corners, shapes comprised of multiple corners can be detected. As shown in figure 1, the corner detection is fully implemented using only EPL queries. First, the average motion-direction is computed over a window of multiple cursor positions. Second, the angle between sequential directions is determined. Finally, corners are detected by looking for a sequence consisting of an angle below a threshold, followed by one or more angles above the threshold, and terminated by an angle below the threshold. Every time such a sequence is found, the average position and greatest angle is used as the position and angle of the final corner. Additionally, the start and end of each stroke is also considered as a corner.

The actual gestures are described using the match-recognize feature of Esper (which is also used in the final step of the corner detection), which allows to specify regular expressions of events. An example of this is shown in figure 2. The detected gestures are translated into gesture events, consisting of the average position of all corners that make up the gesture and an identifier of the gesture.




```
INSERT INTO GestureEvent
SELECT cursorId, "YES" AS type, pos
FROM CornerEvent
MATCH_RECOGNIZE (
    PARTITION BY cursorId
    MEASURES A.cursorId AS cursorId, B.pos AS pos
    PATTERN (A B C)
    DEFINE
        A AS A.angle = 0,
        B AS B.angle BETWEEN 60 AND 150,
        C AS (C.angle = 0) AND ((B.pos).distance(C.pos)
        BETWEEN (1 * (A.pos).distance(B.pos)) AND
                (3 * (A.pos).distance(B.pos)))
)
```

Figure 2: EPL query used for detecting a ‘Yes‘ gesture based on the previously detected corners.

4 Intentional versus Non-Intentional Events

The two fields of application for event processing presented here share many principal characteristics. Of course, all general characteristics of event driven architectures and event processing systems are found. However, they also share more specific traits, like the presence of multiple, heterogeneous event producing devices, the general need for integration and filtering of those events to deduce and extract higher level information, or the overall application area of dealing with human activities. Especially the last characteristic provides several opportunities for integration of both systems.

Considering all the commonalities, the question arises where to draw the line between event processing for home service platforms and interactive (tabletop) systems. It could be argued that they are actually the same, as user input in an interactive tabletop system can be considered just as another sensor like temperature or vital signs. Vice versa, the

context information collected from ambient sensors could be eventually interpreted as just another input to an interactive system (using a very broad definition of that term).

However, not all characteristics are identical. The two fields of application can be distinguished by the intention of processed events: on the one hand, events like 'opening a door' or 'walking across a room' are considered non-intentional with respect to the home service platform, which only observes these events through ambient sensors. The actions behind the events were not performed to trigger some behavior of the system and a response to that action is usually not its primary purpose. While anything that happens because of non-intentional events may possibly be expected by the residents, this is only expected by habit. On the other hand, events like 'touching a display surface', 'pressing a key', or 'saying a defined command' are considered intentional with respect to the home service platform. The actions are performed just for the purpose of interacting with the system and a reaction is expected. Moreover, the feedback is usually expected to be instantaneously and directly observable. Anything happening only after a certain delay[1] is then considered to be non-responding and may even confuse the users.

5 Points of Integration

The two distinct systems provide one the one hand context events, and direct interaction events on the other. As the comparison in the previous section shows, the separation of both systems should not be easily removed. However, there are potential points of integration, which can be identified.

Applications running on one or more interactive tabletops or surfaces in the living environment could benefit from additional information about the context in which they are used. For example, knowing that a person has entered the room could lead to activation of the interface next to him or her. This means turning the context events into direct interaction events. In addition to the use as simple input events, the context events can of course also be used in aggregation and creation of complex input events. Moreover, the context events can be employed in the realization of multimodal interfaces, where selection of the input modality for interaction might depend on the context in which the application is used. Of course, care must be taken not to interpret to many of the context events as direct interaction, because the users might not have intended this meaning and might not know why the interactive system behaves in a certain way.

In the other direction, the direct interaction events produced by the input devices themselves and also the complex events derived later could be fed into the context event processing system. They provide very fine grained information about the current activity of a person and thus can greatly support otherwise more coarsely captured data about the context. For example, the information that someone is currently interacting with the tabletop system in the living room can be used to simply infer that there is someone in the living room and how long they have probably been there. Also, as direct interaction with system usually requires the main focus of a user, it can be inferred that the person is busy working

[1] delays of up to a few hundred milliseconds are usually acceptable, depending on the kind of interaction

or some other task and in case of elderly people, that they are alive and well. Similarly to the use of context events as input to the interactive system, simply feeding all input events also into the context event processing system is probably not useful. Especially as the interaction events are very fine grained, it might be advisable to apply rate limiting[2] and aggregation on the events.

In the following section, we outline how the integration of the two systems allows for self adaptation by managing the resources based on the demand for events.

6 Demand-Oriented Resource Management

Home service platforms in smart homes have to provide as many services as needed and, at the same time, as few as possible. The former results from the requirement, that residents expect all services to be available when they need them, especially when it comes to interactive systems (as discussed above, delayed or even no response of the system is usually not acceptable in this context). The latter results from the limited resources (processing time, memory, bandwidth, energy, etc.) and the demand for reduced overall energy consumption. To provide a solution to this problem, usage of resources has to be managed based on *demand*. While this is not a novel concept in itself, we introduce how it can specifically be applied to intelligent living environments.

In *demand-oriented resource management*, demand is driven by the event consumers, which are the applications and components interested in the available context and interaction information. As most services in smart homes are eventually provided to the people living there, the demand for context and interaction information depends on the interaction of residents with the system. For example, if no one is currently using an interaction device (such as an interactive tabletop system), it would be sensible to turn the device off as well as turning off any services used by it. Vice versa, if someone enters a room, at least basic input detection of the interactive system has to be turned on again, in order to allow the potential user to start interaction. Also, all intermediary processing of events should be gradually started or stopped, depending on the need for it. This interdependence of the home service platform and the interactive systems are the reason why a demand-oriented resource management in an intelligent living environment requires integration of both systems.

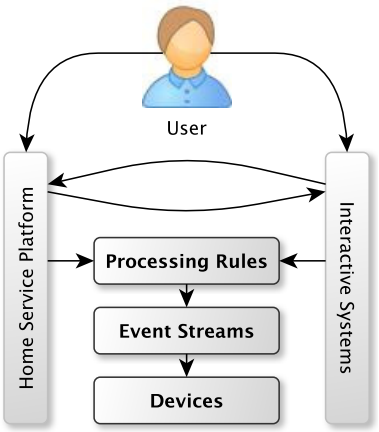


Figure 3: Overview of the demand chain in a demand-oriented resource management for intelligent living environments.

[2] Input events are generated at rates of hundreds to a few thousand events per second, depending on the number of devices and people involved. For example, two input devices with a sampling rate of 60 Hz and two people actively using multitouch gestures with two fingers each, will generate 480 events per second

In the context of complex event processing, managing resources means to manage devices, event sources/streams and processing rules. Figure 3 gives a general overview of the managed components and how each component is driven by the demand from other components. Processing rules, which translate to continuous queries in Esper, are the most specific components. They realize a specific processing step, operate on one or several event streams and may produce events for other rules or consumers. While the event processing engine tries to optimize execution, some resources are still consumed in the form of processing time and memory. As the most specific components, processing rules would be the first candidates for deactivation in case no demand is detected. Event sources or streams are more general, as they are used by a multitude of processing rules or consumers. The streams do usually not require much processing time, but the events have to be stored and possibly sent to other nodes for processing. When all processing rules or consumers interested in a specific event stream have been deactivated, the stream itself would be disabled, which means that no storage or transfer of events occurs. Finally, devices are the coarsest level to manage. They host all event generation, processing or consumption and the main resource consumed by them is energy. As all the previously named components are running on one or more devices, they can be deactivated or shut down, whenever all components currently running on them are deactivated.

These components are disabled or deactivated whenever no demand for the individual components is detected. When demand for certain information is detected again, the process of reactivation is done in reverse and only activates those components which are necessary to fulfill the demand. The management of event processing in this way results in a self adapting system, which efficiently uses resources based on the interaction and activities of the users.

7 Conclusion and Future Work

In this paper, we presented two important application areas for event processing in intelligent living environments and explored their characteristics and commonalities within that context. The potential points of integration indicate, that the two application areas can benefit from each other when information is shared between them. On the other hand, we also showed, that even though many similarities exist, each application area deals with a different kind of events, which are characterized by the intention behind them. Furthermore, we propose a *demand-oriented resource management* in intelligent living environments and outline how it can be realized by integration of user interaction and context event processing.

Based on this, several challenges have to be addressed and more work is required to realize a fully integrated system capable of providing both context and user interaction information and maintaining a self adapting operation. The integration of interaction and context information requires further analysis of their individual characteristics, in order to identify which information can and should be shared and how this information should be interpreted by the other system. Furthermore, the analysis of dependencies in the event processing systems requires to instrument the whole system in an appropriate way. Also, the

demand-oriented resource management requires a reliable way to detect demand based on the user interaction. As the user interaction itself can only be detected when input devices, input event sources and input event processing rules are active, it has to be examined how efficient resource management can be realized even though interdependencies like those exist.

References

[Esp11] EsperTech Inc. ESPER - An Event Stream Processing and Event Correlation Engine (Version 4.4.0). `http://esper.codehaus.org`, October 2011.

[Gro09] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (W3C Recommendation). `http://www.w3.org/TR/owl2-overview/`, October 2009.

[GYJO10] Mingyan Gao, Xiaoyan Yang, Ramesh Jain, and Beng C Ooi. Spatio-temporal event stream processing in multimedia communication systems. In *Scientific and Statistical Database Management*, pages 602–620. Springer, 2010.

[HSB09] Annika Hinze, Kai Sachs, and Alejandro Buchmann. Event-based applications and enabling technologies. In *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, pages 1–15, 2009.

[LDS+10] Simon Lehmann, Ralf Dörner, Ulrich Schwanecke, Johannes Luderschmidt, and Nadia Haubner. An Architecture for Interaction Event Processing in Tabletop Systems. In Ralf Dörner and Detlef Krömker, editors, *Self Integrating Systems for Better Living Environments: First Workshop, Sensyble 2010*, pages 15–19. Shaker Aachen, November 2010.

[Lip09] Michael Lippautz. *Location-Aware Complex Event Processing in Mobile Environments*. PhD thesis, Salzburg University of Applied Sciences, 2009.

[LS11] David Luckham and Roy Schulte, editors. *Event Processing Glossary - Version 2.0*. Event Processing Technical Society, July 2011. `http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2-0/`.

[Luc02] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman, Amsterdam, 2002.

[Mic06] BM Michelson. Event-driven architecture overview. *Patricia Seybold Group*, 2006.

[Sch10] Jan Schaefer. Towards a Platform for Self-Organizing AAL Applications. In Ralf Dörner and Detlef Krömker, editors, *Self Integrating Systems for Better Living Environments: First Workshop, Sensyble 2010*, pages 109–116. Shaker Aachen, November 2010.

[TGNO92] Douglas Terry, David Goldberg, David Nichols, and Brian Oki. Continuous queries over append-only databases. *ACM SIGMOD Record*, 21(2):321–330, June 1992.

[TGS06] Edward Tse, Saul Greenberg, and Chia Shen. Motivating Multimodal Interaction around Digital Tabletops. In *Video Proc. ACM CSCW Conf, Computer Supported Cooperative Work*, pages 6–7, 2006.

[WEH08] Aaron Wolin, B Eoff, and T Hammond. ShortStraw: A Simple and Effective Corner Finder for Polylines. *5th Annual Workshop on SketchBased Interfaces and Modeling*, pages 33–40, 2008.

[WX10] Lin Weiguo and Jin Xin. A sketch recognition algorithm for Pen-based Human-Computer interaction. In *International Conference on Computer Application and System Modeling (ICCASM), 2010*, volume 2, pages 248–251. IEEE, 2010.