Smart Interaction Management: An Interaction Approach for Smart Meeting Rooms

Axel Radloff*, Anke Lehmann*, Oliver Staadt* and Heidrun Schumann*
*Institute of Computer Science
University of Rostock, 18051 Rostock, Germany
Email: {axel.radloff, anke.lehmann, oliver.staadt, heidrun.schumann}@uni-rostock.de

***Abstract*—Understanding information and gaining insights about information usually means interacting with the displayed information. Utilizing the new capabilities of smart meeting rooms, visual outputs of different information representation applications are presented according to the user's needs.**

In this paper we present *smart interaction management*. This interaction approach enables the users to interact with all displayed views, utilizing the novel capabilities of these environments and besides, to traditionally interact with applications using her local device. We further show two use cases demonstrating typical applications of our approach in such multi-display environments: (1) to modify the arrangement and layout of views and (2) to interact with the displayed information within a view.

I. Introduction

Smart meeting rooms are heterogeneous multi-display environments that aim to present a lot of information simultaneously to support users in discussing and making decisions. They typically consist of ad-hoc device assemblies (e.g., multiple sensors, display surfaces, software infrastructure) and provide proactive assistance to users. A typical role in such settings is the presenter role: one presenter conveys information to the audience or leads a discussion with the audience.

Sharing and discussing information usually means presenting this information to all participants. To support this information presentation a smart view management was developed [1]. The main idea was to combine the visual outputs of different applications, that generate information representing views, instead of intertwining the underlying software systems. Hence, personal devices and applications can be used. Through the smart view management, views of different devices and applications can be assigned to different display surfaces of a smart meeting room, as well as combined and arranged on only one display surface. This is achieved by a smart display management that is based on information about the users (i.e., positions, interests, and view directions).

However, exploring information or understanding information and its insights usually means interacting with this information [2][3][4]. Users must be able to interact with the displayed information, especially in the presenter scenario where one person conveys information to an audience. Here, the presenter has to be able to modify any view to convey the insights of the displayed information. It is unfavorable if the presenter has to switch between different devices to modify presented information or ask for modification by another user.

Fig. 1. User interaction with smart views in the smart meeting room.

Moreover, to highlight information or to set the focus to a certain view, the presentation layout should be modifiable interactively.

In this paper, we present the *smart interaction management*, an approach enabling the user to interact with the views and the presented information (see Figure 1). The smart view management serves as basic concept for the view presentation and the connection to the view generating applications.

The main contribution of our work is an interaction approach that enables the user to interact with all displayed information as well as to modify the layout of the information displays in the smart meeting room regardless of the application generating this information display and regardless of the used interaction hardware.

In Section II we give an overview of the related work. Our new interaction approach is based on the smart view management that is briefly introduced in Section III. Afterwards, in Section IV we present the smart interaction management, our new interaction approach to enable the users to globally interact with all displayed views and the presented information, regardless of the interaction device. Furthermore, in Section V we demonstrate the utility of our approach with two characteristic examples. We conclude and give an outlook on future work in Section VI.

II. Related Work

A challenge in smart meeting rooms is to support convenient user interaction when working with distributed displayed information. In recent work multiple display environments were investigated where user are able to interact across heterogeneous display surfaces.

At Stanford University the iRoom was managed with their developed iROS middleware [5]. To listen to user interactions a client was running on each physical machine by using an event-driven framework (Event Heap [6]). The communication method provided a publish/subscribe mechanism by which multiple systems can act as consumers and generators of system events. PointRight [7] was developed to provide a seamless interaction between multiple display surfaces in the iRoom by connecting all available screens to a combined virtual display surface.

Another approach is the Gaia project [8] which included an operating system for environments with heterogeneous devices, display surfaces and an application model. The application model provided interfaces to map developed applications to particular physical space without strict knowledge of the underlying structure. In context of Gaia, Clicky [9] was developed allowing users to control a set of displays connected to different devices by using a single input device. Furthermore the input device was associated with a user for Clicky-aware applications.

Clicky as well as PointRight used a logical mapping between screen positions and connections between display surfaces. Alternatively, a physical mapping between the screens in a multi-display environment was applied by the Universal Interaction Controller (UIC) [10]. The UIC utilized a trackable PDA to point to a display surface. Afterwards, the objects on the selected display surface can be modified on the handheld device screen. All modifications were automatically sent to the physical machine associated with the selected display surface. However, the interaction was bound to the handheld device. Furthermore, the user must split his attention between the input device screen and the selected display surface.

More approaches are dealing with interaction pointer movement in heterogeneous multi-display environments (Perspective Cursor [11], Deskotheque [12], etc.).

Tiled-wall displays are a further use of multiple displays to present information. With the increasing use of these displays flexible interactive applications are required where interaction spans the surface of multiple devices.

Shared Substances [13] is a distributed application model that replicates and mounts shared subtrees (e.g., scene graph, input device) for remote access. Their interaction model listened to events of the input devices and mapped the changes in application nodes that are mounted or replicated.

An architecture that supports indirect multi-touch input (iPad, iPaper, etc.) to interact with an ultra-scale wall display is HIPerFace [14]. This architecture intercepted input device events and forwards them to the applications via pre-defined plugins or APIs.

The various interaction approaches of recent work are not suitable to the constraints of our smart meeting room. Here, visual outputs of different information presentation applications are combined and distributed across different displays. That means, multiple views on one display surface are not necessarily associated with a single specific view generating application. Furthermore, our environment supports ad-hoc usage of personal devices (e.g., laptops, smartphones, tablets) and thus, available interaction devices are varying over time.

In the next section, we shortly describe the smart view management, which serves as basis for our interaction approach (smart interaction management), described in Section IV.

III. Smart View Management

Smart view management was developed to arrange and present the visual outputs of different applications in heterogeneous ad-hoc ensembles with dynamically varying environmental properties. This is achieved by combining the visual outputs of the applications instead of intertwining the systems themselves. Therefore, an abstract term of "view" was defined, describing everything displayable (e.g., visualization, video, slides, document). These views are combined to be presented in a smart meeting room.

Two options were developed to generate views: the ad-hoc and the prepared option.

Ad-hoc: The ad-hoc option was designed to generate views from unmodified or proprietary applications. A view grabber is started by the user, positioned on the screen and activated. The containing content is the resulting view that is streamed via network.

Prepared: The prepared option is based on a lightweight API. Here, by the use of the API the application generates the view itself. That allows adapting the entire content that should be displayed.

Smart view management consists of three building blocks realizing the combination of visual outputs of different applications: (1) interactive view package generation, (2) smart display mapping, and (3) smart view layout [1].

Interactive view package generation

View packages are assemblies of views. They are defined interactively via a lightweight graphical user interface enabling the user to specify characteristics of the views and the view packages they belong to.

Smart display mapping

Smart display mapping automatically assigns the views to the available display surfaces based on their belonging to view packages. For this purpose the view packages and their characteristics are considered. Furthermore, specific properties of the users (position and view direction) and properties of the displays (position, size, resolution, etc.) are used to assign the views to the different display surfaces.

Smart view layout
With smart view layout, multiple views can be arranged on a single display surface. A spring force based layout algorithm is used for the arrangement of the views. Furthermore, pressure-based resizing is used to efficiently utilize the available display space. These force- and pressure-based algorithms are then recalculated iteratively to guarantee a dynamic adaptation of the layout to the current situation (i.e., users and displays properties). Thereby, dynamic occlusion handling is also provided. Thus, display space currently occluded by the presenter is considered to not display views there.

Smart view management [1] only provides implicit interaction.

Smart view layout and smart display mapping are based on user's properties. Thus, the user is able to perform an interaction by changing these properties (e.g., changing position or gaze direction). Nevertheless, these interactions are not easily reasonable for the users and change all displayed views. Hence, they cannot be performed to modify a specific view without affect other views. For instance, the presenter wants to move a certain view to a display surface of her choice. Hence, she changes her position by moving to another display surface. This position change causes the view to "follow" her, but also the other views displayed in the smart meeting room.

Modification of displayed information is not supported by smart view management. For such interaction the physical machine where the view generating application is running, has to be used. For example, the presenter wants to highlight information by setting a certain filter in the visualization. Therefore, the presenter has to move to the laptop where the application is running, performs the interaction, and finally moves back to the presentation position. However, this causes the presentation layout to adjust because of the position change of the presenter. Furthermore, the "flow" of presentation is interrupted and, thus, the conveying of information.

Summarizing this discussion in smart view management two basic interactions are conceivable: (1) modifying the presentation layout and (2) modifying the displayed information.

Modifying the presentation layout means that the presentation of the views is changed. This includes the assignment of the views to the displays (e.g., the presenter wants to drag a view onto the focus display surface) and the layout of the views on one display surface (e.g., the presenter wants to magnify a view for highlighting). Here, access to the view generating application is not necessary. Hence, we call it *application-independent interaction*.

In contrast, modifying the displayed information means to change the displayed content. Here, the interaction concerns the view generating application. For example, the presenter wants to highlight information by applying a filter in the visualization. Here, the access to the view generating application is essential, so we call it *application-dependent interaction*.

IV. Smart Interaction Management

In the following section we present the smart interaction management. This interaction approach enables an application-dependent and application-independent interaction with all displayed views in a smart meeting room, regardless of the view generating application and regardless of the interaction hardware.

Thus, the presenter is able to focus on the conveying of knowledge and the interaction is integrated into the presentation process.

A. General Design

For seamless interaction with all displayed information we make specific design decisions to obtain a smart interaction management.

The main idea of our interaction approach is to decouple the interaction hardware from the interpretation of interaction. Therefore, we subdivide interaction in physical and logical interaction.

Physical interaction is described as any interaction that is performed by a user with input and output devices (e.g., pressing a mouse button). Interaction that user performs with the system is called *logical interaction*. Logical interaction can be subdivided into lexical, syntactic and semantic levels [15]. The *syntactic level* describes the order and structure of interaction elements (e.g., grammar of speech systems, layout of graphical user interface elements). The *lexical level* describes the representation of interaction elements (e.g., look of visual buttons). The *semantic level* describes the effect of interaction on the internal data structure and specifies which information is interpreted.

In the following we talk about semantic interaction when we mean logical interaction at the semantic level and syntactic interaction if we mean logical interaction at the syntactic level.

Besides the separation of physical and logical interaction, the smart interaction management should support an interaction with all displayed views regardless of the type of interaction (application-dependent/application-independent) and regardless of the view generating application. Therefore, we have to solve three basic problems:

- gathering of physical interaction and connection to the smart meeting room
- assignment of interaction
- semantic interpretation of interaction

Smart interaction management consists of three components dealing with these problems: (1) the *interaction grabber*, (2) the *interaction mapper* and (3) the *interaction handler*.

In the following we describe the general interaction process of smart interaction management. We refer to Figure 2 that illustrates the basic process. In the following subsections we describe the functionality of the components in detail.

The first problem, gathering of physical interactions is encountered by the *interaction grabber*. This component runs on the physical machine to which the interaction hardware is connected. Here, physical interaction generated by the

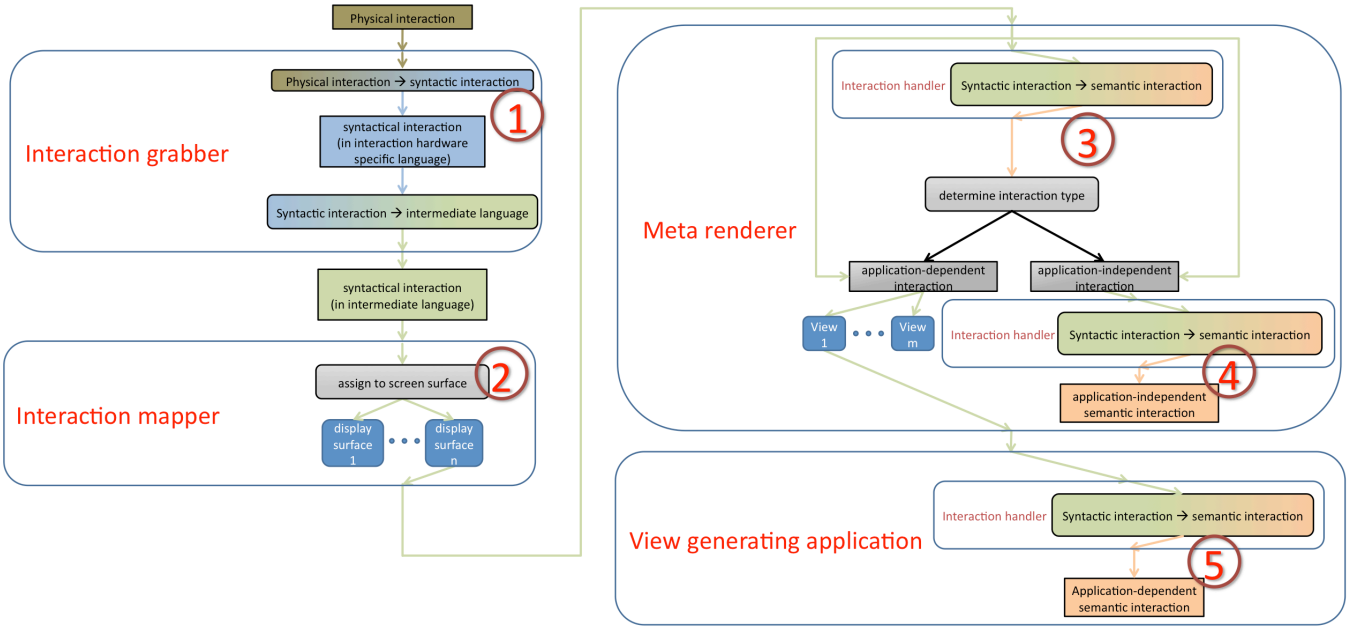

Fig. 2. The interaction process of the smart interaction management.

interaction hardware is converted into syntactic interaction by the hardware driver (see Figure 2, mark 1). Furthermore, the interaction grabber realizes the connection between the interaction hardware and the smart interaction management. In summary, the input of interaction grabber is a physical interaction, whereas the output is a syntactic interaction.

The next step is to assign the syntactic interaction. Therefore, the *interaction mapper* determines the display surface where the interaction is performed and transfers the syntactic interaction to the meta renderer (see Figure 2, mark 2). The meta renderer handles the view rendering for this display surface and is part of the smart view management.

The *interaction handler* is used to map syntactic interaction into semantic interaction. Before interpreting syntactic interaction, the type of interaction has to be determined (see Figure 2, mark 3). This is done in the meta renderer. If an application-independent interaction should be performed, the semantic interpretation has to take place at the meta renderer (see Figure 2, mark 4). Whereas, if an application-dependent interaction should be performed, the interaction has to be transferred to the view generating application and is interpreted there (see Figure 2, mark 5). The result of the interaction handler is the semantic interaction encoded in the interaction language used by the view generating application (i.e., application-dependent interaction) or the meta renderer (i.e., application-independent interaction).

This interaction process described above, includes the complete flow of the interactions in the smart interaction management. Interaction is gathered as physical interaction, translated into syntactical interaction and finally mapped to semantic interaction.

However, a crucial choice is the strategy for the mapping of syntactic interaction to semantic interaction. Here, two approaches are conceivable: (1) a direct mapping or (2) a multi-step mapping.

1) Direct mapping means that the syntactic interaction generated by the interaction grabber and encoded in the interaction language of the certain interaction hardware are transferred to the interaction handler. Here, the syntactic interaction is mapped to semantic interaction.
2) Multi-step mapping means that the syntactic interaction of the interaction hardware is mapped to an intermediate language in the interaction grabber. Then, the syntactic interaction in intermediate language is transferred to the interaction handler. There, the interaction in intermediate language is mapped to the semantic interaction.

Both approaches have their advantages and disadvantages. For *direct mapping*, the realization of the interaction grabber is much easier. Hence, supporting a new interaction device requires little implementation effort. However, at the target application, where the interaction should be executed, the mapping of syntactic interaction of any interaction hardware to the system specific semantic interaction has to be implemented. This is a significant implementation effort for all systems connected to the smart interaction management.

Alternatively, using *multi-step mapping*, an intermediate language causes an additional implementation effort at the interaction grabber, because the syntactic interaction of the interaction hardware has to be mapped to the intermediate language. In this case, a mapping from syntactic interaction in intermediate language to semantic interaction has to be implemented only once in the interaction handler. Hence, the implementation effort for the different applications is obviously reduced.

To generate views even from proprietary applications the *view grabber* of the smart view management is used (see Section III). By using an intermediate language the mapping to semantic interaction can be included into the view grabber and thus, even unmodified applications are usable for application-dependent interactions.

Based on the characteristics of the two approaches, we decided to use a multi-step mapping. Hence, the integration of new interaction hardware is more flexible and it does not require a huge implementation effort. Moreover, by using a view grabber even proprietary applications are usable.

In the following subsections, we discuss in detail how these components of the smart interaction management are designed. Furthermore, in the use case section (see Section V) we show an actual application of the described components.

B. Interaction Grabber

As presented above the interaction grabber gathers the interaction by mapping physical interaction to syntactic interaction. We apply the multi-step mapping approach that uses an intermediate language.

The requirement on such intermediate language is that the functionality of any interaction hardware can be mapped to the intermediate language.

For the intermediate language two options are conceivable: (1) designing an abstract language or (2) using a concrete language.

The advantage of an *abstract intermediate language* is that the language can be designed generically and thus, it is extensible at will. However, when implementing the connection of a new application to the smart interaction management knowledge about the structure and specification of the abstract intermediate language have to be provided.

In contrast, using a capable *concrete intermediate language* has some advantages as well. If an application already defines its semantic interaction using this language, the syntactic interaction can be directly mapped to semantic interaction without further interpretation. Furthermore, the internal structure of the language is already defined and no proprietary specification is required for the mapping at the applications. However, the language is not extensible. This disadvantage can be compensated by using modifiers. Modifiers allow to map a syntactic interaction to different semantic interactions depending on current interaction mode, that is enabled by the modifier.

In our smart interaction management, we use an existing language. For this purpose, we choose the syntactic interaction of *mouse and keyboard* as intermediate language.

On one hand, this intermediate language is often already used in the applications to realize the interaction, especially in standard desktop applications. Thus, a direct mapping without further translation can be performed. On the other hand, mouse and keyboard have a high cardinality and hence, they are powerful enough to map a multitude of different interaction hardware functionality.

For example, a spatial interaction device provides six degrees of freedom (DOF). In contrast, the mouse only has two DOF. To map the other four DOF, keyboard modifiers (e.g., [Alt], [Crtl]) are used.

However, the mapping of syntactic interaction of an interaction device is not necessarily obvious. For this reason, a mapping preset is defined during the realization of the interaction grabber. This mapping can be changed interactively by the user via user interface, similar to [14].

As a result of the interaction grabber, the interaction performed by the presenter is represented as syntactic interaction encoded in mouse and keyboard language. Furthermore, this interaction is made available to the smart interaction management. In the next step the display surface is determined by the interaction mapper.

C. Interaction Mapper

The presenter's interaction is encoded in intermediate language (mouse and keyboard language), after mapping physical interaction to syntactical interaction by the interaction grabber.

However, the aim of an interaction is to manipulate a view (position, size, display assignment, displayed content, etc.), regardless of application-dependent or application-independent interaction. That means an interaction is performed in relation to a target view and a target display surface where the view is presented. Hence, the interaction has to be mapped to the intended display surface. This is achieved by the *interaction mapper*.

Therefore, the absolute position and orientation of the display surfaces in the smart meeting room has to be known. Based on this knowledge, two options for the mapping are conceivable: (1) relative positioning or (2) absolute positioning.

Relative positioning means that from a certain point in the room a 2D projection of the spatial display surface positions is performed, similar to [5], describing the adjacent display surfaces.

In contrast, *absolute positioning* is based on additional sensor data. By attaching markers to the interaction hardware, its absolute position can be determined. Thus, by ray casting the pointing direction the target display surface is determined, similar to [10].

The interaction mapper supports both mapping options. When tracking markers are attached to the interaction hardware for the absolute positioning this option is used. In case the absolute positioning is not available the relative positioning is used.

As a result, the interaction mapper identifies the target display surface where the interaction is performed. The syntactic interaction encoded in intermediate language is transferred to the meta renderer, that is responsible for the presentation of the views on this display surface. In the next step, the type of interaction (application-dependent/application-independent) is determined by the meta renderer and interpreted by the interaction handler.

TABLE I
MAPPING OF SYNTACTIC INTERACTION IN INTERMEDIATE LANGUAGE TO SEMANTIC INTERACTION.

semantic interaction	**intermediate language**
select view	mouse motion + pointer hovers over a view
move view	mouse motion + left mouse button pressed on selected view
scale view	mouse wheel up/down on selected view
	y-axis mouse motion + middle mouse button pressed on selected view

D. Interaction Handler

The interaction handler is the last step in the interaction process. After describing the syntactic interaction in intermediate language (interaction grabber) and mapping the interaction to the target display surface (interaction mapper), the interaction handler maps the syntactic interaction to semantic interaction.

In smart interaction management we distinguish between the two types of interaction, application-independent and application-dependent. The determination of the interaction type is realized in the meta renderer (see Figure 2, mark 3). By default, the interaction is handled as application-independent interaction. To perform an application-dependent interaction the presenter has explicitly to switch to this type of interaction by using a dedicated trigger.

For the purpose of an **application-independent interaction** an instance of the interaction handler is integrated in the meta renderer. Here, a mapping of syntactic interaction in intermediate language to semantic interaction of the meta renderer is defined (see Figure 2, mark 4). An example of such mapping, to modify the layout, is given in Table I.

On the other hand, if an **application-dependent interaction** is performed, the target view is identified based on the view's position on the display surface. Then the syntactic interaction is transferred to the view generating application. Here, a further instance of the interaction handler maps syntactic interaction to semantic interaction of the specific application. Therefore, the mapping has to be particularly defined to cover the application's capabilities.

Nevertheless, when multiple users interact simultaneously, the assignment of permissions for interaction is a known problem. Several papers discuss this issue in different environments (e.g., [16][17][18]). Furthermore, this problem has an important social component, how people work together in which constellations. However, this is not the scope of this paper.

V. USE CASE

In this section we present use cases for both types of interaction within the smart interaction management: (1) application-independent interaction and (2) application-dependent interaction.

Fig. 3. Application-independent interaction performed by a presenter: select view (top), move selected view (middle) and scale selected view (bottom).

First, we demonstrate the use of a Nintendo Wii remote controller (Wiimote) as interaction hardware for an application-independent interaction. Second, we show the use of mouse and keyboard for an application-dependent interaction. In these use cases, CGV (Coordinated Graph Visualization) [19] is applied as view generating application.

The use cases are described according to the interaction process illustrated in Figure 2. In both use cases the *interaction mapper* uses the relative positioning.

A. Application-independent interaction

As a use case for application-independent interaction we show the use of a Wiimote to modify the view layout and the assignment of views to the display surfaces.

To connect the Wiimote to the physical machine, where the interaction grabber is running, a Bluetooth driver is used. Furthermore, the physical interaction determined by the driver is translated into syntactic interaction using a third party library. In our case we use Java with the "WiiuseJ" API[1]. This API generates events representing the syntactic interaction.

The next step is the mapping of syntactic interaction to the intermediate language (mouse and keyboard). Therefore, the interaction capabilities of the Wiimote have to be investigated.

[1]http://code.google.com/p/wiiusej/

Fig. 4. Application-dependent interaction performed by a presenter: select a polyline in the upper view of the display surface (highlighted in the scaled subimage).

TABLE II
MAPPING OF WIIMOTE'S SYNTACTIC INTERACTION TO INTERMEDIATE LANGUAGE (MOUSE AND KEYBOARD).

syntactic interaction	**intermediate language**
infrared sensor motion	mouse motion
"A" button	left mouse button
"B" button	middle mouse button
"+" button	mouse wheel up
"–" button	mouse wheel down
accelerator y-axis and "B" button is pressed	y-axis mouse motion
"home" button	[insert]-key

The Wiimote has a variety of different sensors detecting different kinds of movements.

First, it includes buttons with binary states (pressed or not pressed). Second, it has sensors detecting the acceleration and the roll-pitch-yaw angles. And third, through the infrared sensor a two-dimensional movement (x,y) is detected.

Table II presents the Wiimote's mapping of syntactic interaction to to interaction in intermediate language, i.e., mouse and keyboard interaction (see discussion in Section IV-B). This mapping is extensible according to the supported interaction.

Afterwards the mapping of the syntactic interaction in intermediate language to semantic interaction is done using the *interaction handler*. In this use case, an application-independent interaction is performed and thus, the interaction handler of the meta renderer is responsible for this mapping.

Here, the current semantic interactions are supported: (1) select a view, (2) move a view and (3) scale a view (see Table I).

With this mapping, an application-independent interaction with a Wiimote can be performed to modify the layout of all views displayed in our smart meeting room (see Figure 3).

B. Application-dependent interaction

As a use case for application-dependent interaction we show the use of classical mouse and keyboard as interaction devices to interact with content of a view generated by CGV [19]. CGV is powerful visualization system, which displays multiple views to present information through different visualization techniques. Moreover, any performed interaction is coordinated across all representations.

Even for mouse and keyboard an interaction grabber has to be realized. These devices are standard interaction devices for personal computers (e.g., laptop, netbook). This also means that any interaction performed with these input devices are interpreted and executed by the used computer. However, the syntactic interaction generated by these devices is used for the smart interaction management. Therefore, the semantic interpretation has to be disabled on the personal computer.

For this interaction grabber a window is created to provide the syntactic interaction to the smart interaction management. If the mouse pointer reaches the side of a window, the pointer is automatically moved to the opposite side, to keep the mouse within the window. The resulting syntactic interaction is already given in intermediate language and, thus, no further mapping is required.

For example, the presenter wants to highlight a subset of a certain dataset in a displayed view generated by CGV.

Therefore, the presenter has to use the trigger to enable application-dependent interaction by pressing the [insert]-key. The activation of this trigger is a syntactic interaction and has to be semantically interpreted by the interaction handler of the meta renderer (see Figure 2, mark 3).

The target view is then determined by using the knowledge of the meta renderer (i.e., position and size of displayed views). Then, the syntactic interaction is send to CGV. There, an interaction handler is implemented to map syntactic interaction in mouse and keyboard language to semantic interaction

of CGV.

Referring to the example, the presenter selects a polyline in the view displaying parallel coordinates (see Figure 4). Beneficially, through the coordination of interaction in CGV the selected data is highlighted in all views presenting the same dataset.

VI. Conclusion and Future Work

To understand information and to gain insights about this information, the interaction with this information is necessary. In this paper we presented the *smart interaction management*, an approach that enables users to interact with all views displayed in the smart meeting room. Thereby, the user is able to modify information displayed in the view as well as the layout of the presented views, regardless of the used interaction hardware and the view generating application.

An open issue for future work is a concept for an appropriate permission system for collaborative interaction. On the one hand, existing approaches can be adapted to our interaction approach. On the other hand, our smart interaction management enables the development of new permission systems, e.g., by distinguish between interactions performed at local devices and interactions performed using the smart interaction management. Furthermore, specific permission systems considering user properties (e.g., role, domain knowledge, experience) are conceivable.

Acknowledgment

This work was supported by a grant of the German National Research Foundation (DFG), Graduate School 1424 MuSAMA. We thank Valerius Weigandt for his implementation work.

References

[1] A. Radloff, M. Luboschik, and H. Schumann, "Smart views in smart environments," in *Smart Graphics - 11th International Symposium, SG 2011, Bremen, Germany, July 18-20, 2011. Proceedings*, L. Dickmann, G. Volkmann, R. Malaka, S. Boll, A. Krüger, and P. Olivier, Eds., vol. 6815. Springer, 2011, pp. 1–12.

[2] R. Kosara, H. Hauser, and D. Gresh, "An interaction view on information visualization," *State-of-the-Art Report. Proceedings of EUROGRAPHICS*, 2003.

[3] J. Thomas and K. Cook, *Illuminating the path: The research and development agenda for visual analytics*. IEEE, 2005, vol. 54.

[4] J. Yi, Y. ah Kang, J. Stasko, and J. Jacko, "Toward a deeper understanding of the role of interaction in information visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, no. 6, pp. 1224–1231, 2007.

[5] B. Johanson, G. Hutchins, and T. Winograd, "Pointright: A system for pointer/keyboard redirection among multiple displays and machines," Tech. Rep., 2000.

[6] B. Johanson and A. Fox, "The event heap: A coordination infrastructure for interactive workspaces," in *4th IEEE Workshop Mobile Computer Sys- tems and Applications (WMCSA 2002)*. Piscataway, N.J.: IEEE Press, 2002, pp. 83–93.

[7] B. Johanson, A. Fox, and T. Winograd, "The interactive workspaces project: Experiences with ubiquitous computing rooms," *IEEE Pervasive Computing*, vol. 1, pp. 67–74, April 2002.

[8] M. Roman and R. H. Campbell, "Gaia: enabling active spaces," in *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, ser. EW 9. New York, NY, USA: ACM, 2000, pp. 229–234.

[9] C. R. Andrews, G. Sampemane, A. Weiler, and R. H. Campbell, "Clicky: User-centric input for active spaces," University of Illinois, Tech. Rep., 2004.

[10] H. Slay and B. Thomas, "Interaction and visualisation across multiple displays in ubiquitous computing environments," in *Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, ser. AFRIGRAPH '06. New York, NY, USA: ACM, 2006, pp. 75–84.

[11] M. A. Nacenta, S. Sallam, B. Champoux, S. Subramanian, and C. Gutwin, "Perspective cursor: Perspective-based interaction for multi-display environments," in *CHI 2006*, 2006, pp. 289–298.

[12] M. Waldner, C. Pirchheim, E. Kruijff, and D. Schmalstieg, "Automatic configuration of spatially consistent mouse pointer navigation in multi-display environments," in *Proceedings of the 15th international conference on Intelligent user interfaces*, ser. IUI '10. New York, NY, USA: ACM, 2010, pp. 397–400.

[13] T. Gjerlufsen, C. N. Klokmose, J. Eagan, C. Pillias, and M. Beaudouin-Lafon, "Shared substance: developing flexible multi-surface applications." in *CHI*, D. S. Tan, S. Amershi, B. Begole, W. A. Kellogg, and M. Tungare, Eds. ACM, 2011, pp. 3383–3392.

[14] N. Weibel, R. Oda, F. Kuester, A. Satyanarayan, S. Yamaoka, W. G. Griswold, A. Lazar, K.-U. Doerr, and J. D. Hollan, "Hiperface: a multi-channel architecture to explore multimodal interactions with ultra-scale wall displays," in *ICSE '11: Proceedings of the 33rd International Conference on Software Engineering*. New York, NY, USA: ACM, 2011.

[15] G. de Melo, "Modellbasierte entwicklung von interaktionsanwendungen," Ph.D. dissertation, University Ulm, 2010.

[16] A. Bryden, G. Phillips Jr., Y. Griguer, J. Moxon, and M. Gleicher, "Improving collaborative visualization of structural biology," in *7th International Symposium on Visual Computing*, ser. Lecture Notes in Computer Science. Springer, 2011.

[17] C. Forlines and R. Lilien, "Adapting a single-user, single-display molecular visualization application for use in a multi-user, multi-display environment," in *Proceedings of the working conference on Advanced visual interfaces*, ser. AVI '08. New York, NY, USA: ACM, 2008, pp. 367–371.

[18] S. Su, R. Loftin, D. Chen, Y. Fang, and C. Lin, "Distributed collaborative virtual environment: Paulingworld," in *Proceedings of the 10th International Conference on Artificial Reality and Telexistence*, 2000, pp. 112–117.

[19] C. Tominski, J. Abello, and H. Schumann, "Cgv – an interactive graph visualization system," *Computers & Graphics*, vol. 33, no. 6, 2009.