glueTK : A Framework For Multi-Modal, Multi-Display Human-Machine-Interaction

Florian van de Camp
Fraunhofer IOSB
Karlsruhe, Germany
florian.vandecamp@iosb.fraunhofer.de

Rainer Stiefelhagen
Karlsruhe Institute of Technology
Karlsruhe, Germany
rainer.stiefelhagen@kit.edu

ABSTRACT
As new input modalities allow interaction not only in front of a single display, but enable interaction in the whole room, application developers face new challenges. They have to handle many new input modalities, each with its own interface and requirements for pre-processing, deal with multiple displays, and applications that are distributed across multiple machines. We present glueTK, a framework that abstracts from the complexities of these input modalities, allows the design of interfaces for a wide range of display sizes, and makes the distribution across multiple machines transparent to the developer as well as the user. With an example application we demonstrate the wide range of input modalities glueTK can support and the functionality this enables. GlueTK moves away from the focus on point and touch like input modalities, enabling the design of applications tailored towards interactive rooms instead of the traditional desktop environment.

Author Keywords
Application frameworks;multi-modal interfaces

ACM Classification Keywords
H.5.2 Information interfaces and presentation: User Interfaces. - Graphical user interfaces.

General Terms
Design, Human Factors

INTRODUCTION
A wide range of new input modalities are becoming mature enough to be used in real world applications. The introduction of the Kinect has shown, that there is much interest in new input technology. It has also shown, that the focus of interaction broadens from thinking just in terms of display coordinates, to the human in front of the display. Other computer vision technology offers an even wider spectrum of information about humans that can be used for interaction. For an application developer however, it is still hard to make use of all this information. Current frameworks are mostly WIMP based and therefore do not offer interfaces for modalities other than mouse and keyboard. In many cases this has led to using touch like input modalities to emulate mouse input, which severely limits the real potential of this new generation of input modalities that can provide richer and more natural ways of interaction. Post-WIMP frameworks have been proposed, but they take care of handling almost exclusively touch like input modalities.

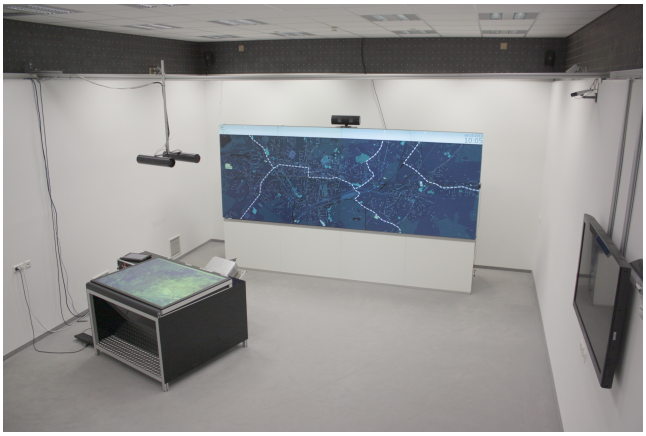

Figure 1. Room setup with back projection video wall and digital table

When Xerox created the Star with the mouse as an input modality, they completely rethought the user interface [15]. Unlike previous text based interfaces they tailored the interface towards making use of the potential of a computer mouse. This example shows, for all new input modalities, input (handling new input modalities) and output (display of content) can not be considered independent of each other.
GlueTK is a framework for handling both, input and output, which allows the output to be tailored towards the properties of the input modalities and the input modalities to be improved by utilizing knowledge about what is currently being displayed. In addition, glueTK abstracts away from the fact that applications are distributed across multiple machines and displays by introducing a network transparent signal and slot system.
The requirements for such a framework pose many challenges. A large variety of input modalities has to be handled, which are connected in completely different ways and provide not only touch like data but completely orthogonal input data as well. Since many input modalities are concerned with persons in the whole room rather than in front of a single

IUI'13, March 19–22, 2013, Santa Monica, CA, USA.

display, multiple display devices with different aspect ratios, resolutions and sizes, ranging from smartphones to display walls of many meters in size have to be handled. And because multiple displays are involved, the framework has to abstract from the fact that multiple machines and applications are involved; neither the application developer nor the user wants to worry about this. The framework has to allow for transparent communication and data transfer across applications and machines. All components of the framework have to meet high performance requirements as the input needs to trigger instant visual feedback and allow for low latency interaction. The latency of the whole interaction chain from input handling to display update has to be minimal to allow for fluent, natural interaction.
The remainder of the paper is structured as follows: First we will discuss related work, followed by a description of glueTK's architecture and its components. Using an example application for crisis control rooms we give an insight to what kind of applications can be created using glueTK. A final conclusion will both summarize and give an outlook on the future plans for glueTK.

RELATED WORK

There have been several approaches to handle touch and touch-like input modalities. Especially for table top displays, multi touch has gained much popularity and several frameworks have been created with such a display in mind [7, 26, 11, 16, 24]. While these frameworks deal with the many different technologies enabling multi touch they are mostly limited to touch and multi touch like modalities. As the table top display is the focus for these frameworks, they do not support inter display and inter machine interaction.
Other projects move away from a single display to allow interaction in a room. The intelligent room project [5] is one, that aims at providing an intelligent room, without any displays, that users can interact with in a star-trek like fashion. The system integrates multiple input modalities by utilizing "agent-based layers" in which each agent offers an abstraction of the component it wraps, so the application developer doesn't have to deal with it in detail. One of the main goals of the interactive workspace project by Johanson et al. [14] is the design and use of rooms that contain one or more large displays. The use of these is not tailored towards the multiple modalities and display sizes for the most part, but towards using readily available technology such as websites and traditional WIMP applications. They implemented an "Event Heap" that is used as a central entity to distribute events between all machines and interfaces. A specific application of multiple input modalities for collaborative work on multiple devices has been presented by Bragdon et al. in [4]. Instead of presenting a generic approach to implementing such applications, they focus on the combination of touch and pointing as well as the social acceptability of gestures in an office setting.
Krumm et al. [17] present one of the few systems that go beyond touch-like modalities for interaction. In their system, they use a multi person tracker to control certain functionality of an intelligent room. However their focus is the tracking system itself that has to cope with the challenges faced in such an environment and not the use or integration of the tracking system for interaction. Fernandes et al. [8] describe a middleware framework that allows the use of multiple modalities that can easily be exchanged if they offer overlapping capabilities.

There has been a lot of work on fusing multiple input modalities [9, 23, 18]. There are frameworks to accomplish the fusion of multiple input modalities in a generic fashion such as MUDRA [13]. The focus is the fusion itself and therefore only touch-like modalities or naturally related modalities such as speech are considered in most cases. Most of the work on multi modal input ignores the question of the corresponding user interface. However as Oviatt [20] points out, output is part of a multi modal interaction. Melchior et al. [19] describe a toolkit that deals with many issues of multi display interaction, especially data and widget transfer between displays. While supporting multiple displays, devices and platforms, for input and output they rely on the traditional WIMP paradigm and also focus on working on a single device at a time. The ROSS API [28] is a toolkit that allows communication between multiple devices to exchange input and output data. The main focus is a nested structure to organize devices and modalities within a room. One of the few examples of frameworks that include dedicated output is PyMT [12]. They state that current GUI toolkits are WIMP based and therefore not built for the new range of input modalities. Rendering user interfaces tailored towards multi touch interaction is a part of PyMT. This allows them to build applications that take into account the features multi touch has to offer compared to a mouse, and also deal with its properties. PyMT is build for touch like input modalities and while the control of input and output would allow for it, they do not make use of the knowledge about what is displayed on the screen to improve the input modalities.
While most frameworks only focus on the input modalities, mostly ignoring the display of user interfaces and the actual interaction with displays, glueTK provides one coherent framework to handle input as well as create user interfaces and drive multiple display devices. GlueTK supports not only touch and pointing like modalities and speech, which are rather common, but also enables the use of information from systems like person tracking, head pose estimation and face identification for interaction.

GLUETK

In the following sections we will describe the glueTK architecture and how it deals with the challenges involved in creating multi-display applications utilizing a large variety of input modalities. One of the main challenges is making all different input modalities easily available, as they come with a wide variety of interfaces, data representations, and they require different kinds of pre-processing. Displays vary in physical size, aspect ratio, and resolution, this is both challenging from a performance perspective but also from a design perspective as user interface elements have to be tailored towards these different configurations. Since many new input modalities, like pointing gestures, are not necessarily bound to a single display, glueTK applications can span across multiple displays that are driven by different machines. Because

of this necessary separation, glueTK has to deal with communication and data transfer across machines to make this distribution transparent to the application developer and the end user. GlueTK is implemented in C++ and uses the Clutter toolkit [1] for rendering. To seamlessly integrate the multi modal and multi display aspects of glueTK, Clutter is completely wrapped and invisible to the developer. For network communication we use a middleware based on the publish/subscribe communication paradigm, developed in our research group. GlueTK is split into two separate libraries: glueInput and glueOutput. GlueInput provides a clean interface to input modalities and takes care of all pre-processing and adaptation so the developer only has to deal with one coherent interface. GlueOutput offers an easy way to build efficient interfaces for a large variety of display types, from smartphones to high resolution video walls, connected to one or multiple machines in a network. In many cases, it is not an option to rewrite existing applications using glueTK. In these cases, glueInput can be used without glueOutput to integrate new input modalities into an existing application.

The contributions of this paper are:

- A network transparent signal and slot system
- A flexible and simple interface to modalities beyond mouselike input and context aware preprocessing
- A generic framework for creating cross machine and cross display user aware applications
- A way to enable the use of new modalities in existing applications without limitation of the modalities potentials

GLUETK ARCHITECTURE

As an introduction to the components that make up glueTK and how they work together, Figure 2 illustrates the architecture with some exemplary components. Along with the following description, this is intended to give an overview of the architecture before the components will be described in more detail.

The input modalities are displayed on the left. FaceID, a face recognition system, which outputs progress information about the identification process, Persontracker, a multi person tracking system which generates a list of coordinates of persons in a room, a pointing gesture recognition system, and a gyro mouse, which outputs relative display coordinates. While FaceID, Persontracker and pointing recognition send their information via a local network, the gyro mouse has a USB interface. For each of those input modalities there is a corresponding event handler on the right. The goal of the event handlers is to gather data from the input modalities, do pre-processing and add functionality. All event handlers push the final information as signals to the event manager. Besides managing all event handlers and being the central point of communication for all input data, the event manager also takes care of subscribing to network streams. This way, if multiple event handlers need the same data, there is no need to subscribe to them separately in each event handler. Context handlers subscribe to signals of other event handlers and generate additional information. The 3D pointing directions of the pointing system, for example, are mapped to displays in the room this way.

Because many displays can be used in a glueTK application that are driven by different machines, intra machine as well as inter machine communication and data transfer are important aspects of glueTK. To make the differentiation easier, a glueTK application consisting of several applications in the traditional sense is from here on called “glue application”. All applications that make up such a glue application, communicate using a network transparent signal and slot system, which also takes care of the communication within each application. The proxy event handler takes care of distributing local signals via the network to all other applications and making the received signals available locally. This way, within a glue application there is no difference between a signal from within the same application and a signal from a different application running on a different machine.

To distribute these signals to the right slots of the right widgets, the signal manager keeps track of connected signals and slots using a mapping table. Every time a signal comes in, either from a local widget or from the event manager, the signal manager checks if any slots are connected to it and calls the corresponding slots if that is the case. If only glueInput is used, the signal manager constitutes the interface to any third party software. If both glueInput and glueOutput are used, a derived signal manager also initializes the display and manages the rendering of all widgets.

Widgets are interface elements created from basic building blocks provided by glueTK. A widget can offer slots that can be connected to any signal, be it for communication with other widgets, e.g. a button click triggering the display of an image, or for reacting to data from input modalities, e.g. moving the widget to the display position provided by the gyro mouse, to create a cursor.

GLUE INPUT

The input layer takes care of getting information from input modalities to the application developer. This way, instead of having to deal with a multitude of different interfaces to get the input data, all information is provided via one coherent interface.

Event manager

The event manager is the entity between the event handlers and the user interface. By having one central component for communication between input and output layer, many duplications can be avoided. It allows for a single, clean interface between the layers, which makes it easy to separate them. To provide input modalities with context information, glueTK applications always provide information about the state of the user interface elements (position, size, state etc.) to the event manager where event handlers can access this information.

If glueInput is used to connect modalities to an existing application, the application developer can provide context information at run time which is made available to the event handlers to exploit this information. For delivering input data from any modality to the user interface, glueTK allows for similar flexibility. All input data is provided as signals which can be mapped to functions as will be shown below. This way, the developer has a coherent way of accessing all data,

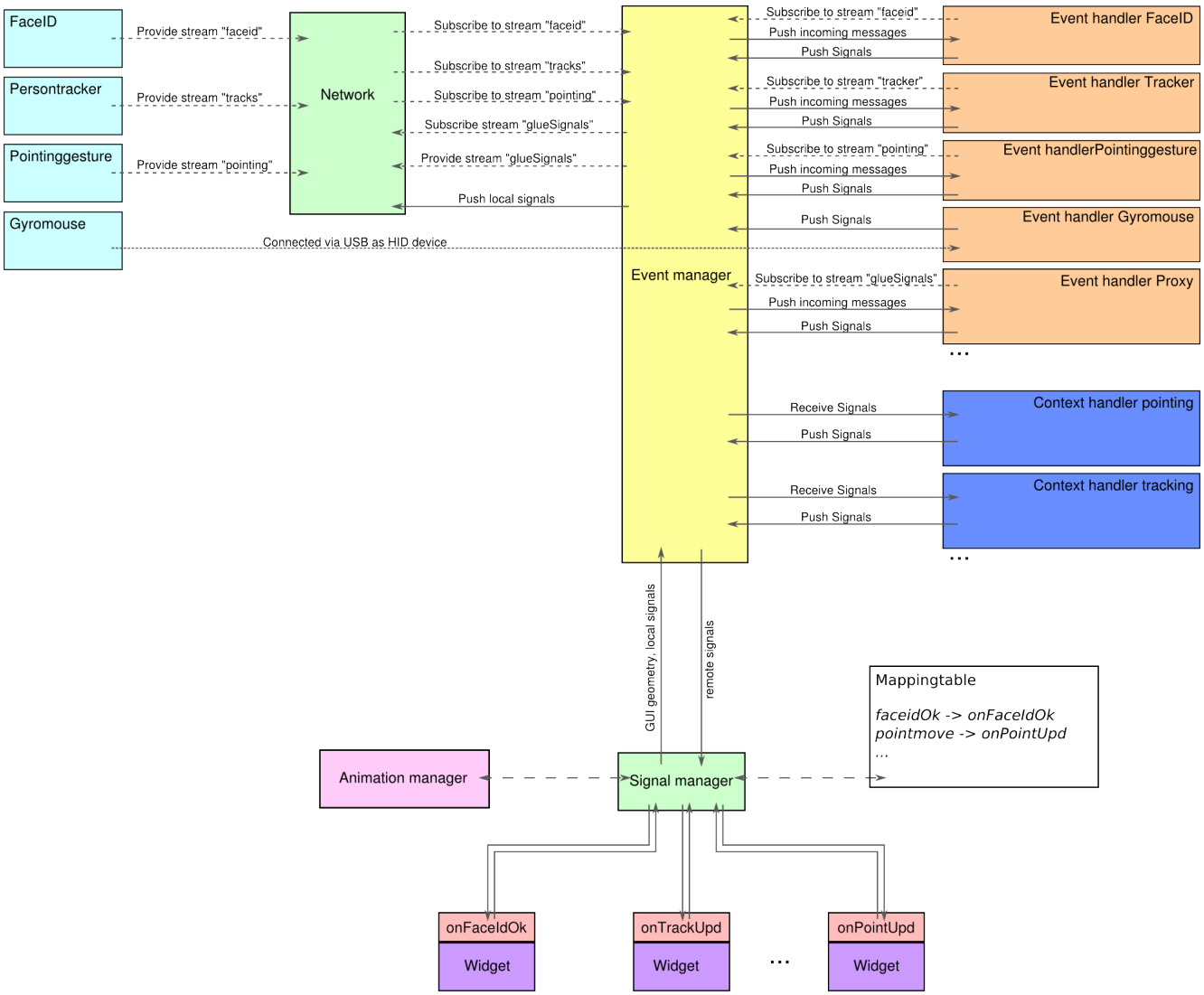

Figure 2. glueTK architecture overview with exemplary input modalities and corresponding event handlers

independent of the actual formatting, protocols or required pre-processing the modalities rely on.

Event handlers

Input modalities come with a wide variety of interfaces. This ranges from devices directly connected via USB, requiring special drivers or emulating a mouse, to systems running on different machines, communicating via a network. Because this variety of sources complicates the handling of multiple modalities, glueTK abstracts from them at the earliest possible point using event handlers. While many established input devices can simply be wrapped by an event handler, so their input data can be accessed in a generic fashion, there are many new modalities that can be greatly improved in the event handler. Two examples for improvement are data filtering and added functionality. Unlike the input device itself, the event handler has access to the current state and layout of the user interface. This allows for taking this context information into account when filtering data to, for example, make use of assisting technologies like force fields [3]. An example for added functionality is the click event we added to our pointing gesture recognition system. The pointing gesture recognition system only provides the pointing direction. To actually interact with an interface it lacks a way to trigger a click. An easy way to implement a click functionality is a dwell timer - keeping the pointing arm at the same position for a certain amount of time triggers a click. Instead of implementing this feature inside the pointing gesture recognition system it was implemented within the corresponding event handler. This way, the event handler will only start the dwell timer if the gesture points towards a click able object. Also, the event handler can easily pass information about the dwell progress to the application, which provides the user with immediate visual feedback as illustrated in Fig. 3. As the pointing gesture recognition is not as robust and accurate as a mouse, the amount of filtering applied to the pointing data can be increased when close to a click able object to assist the user in keeping the cursor still.

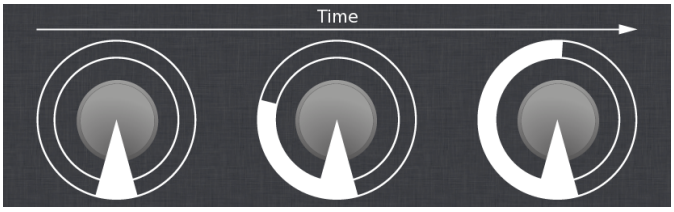

Figure 3. Visual feedback when clicking a button

Context handlers

While it is common to make raw input data like coordinates from a person tracker available, in many cases application developers do not want to deal with the details of this information as this would require additional logic in the interface. To abstract from this low level information, context handlers subscribe to the signals from one or more event handlers and provide higher level information, again as signals, to the application. In the case of a person tracker, the kind of high level information could be relations of persons to displays (eg. "person_x_left_display_y"'). In case of pointing modalities that are detached from a specific display and simply provide pointing directions in the room coordinate system, the data has to be mapped to the displays in the room to provide display coordinates that can actually be used to interact with an interface on a screen. Context handlers are, however, not limited to abstracting from data from a single event handler. Fusion of both similar modalities and heterogeneous modalities can be done within a context handler. For example, using both person tracker and pointing system data, the detected pointing directions can be associated with the corresponding persons. This allows to differentiate between a multipoint gesture by a single person and two single point gestures by two different persons.

COMMUNICATION

GlueTK provides a communication mechanism based on the signal and slot concept [27]. It is used to provide the input data from the event handlers to all applications within the glue application. This makes input modalities usable in a whole room independent of the actual machine they are connected to. In addition, the same mechanism is used to allow widgets to communicate with each other, again independent of what machine they reside on, and to transfer widgets between displays and therefore machines.

Signals and slots

The idea of a signal and slot system is, that objects can send signals containing arbitrary information which - if connected - are received by other objects and will execute special functions, so called slots. This concept allows for very flexible, asynchronous communication. Unlike previous implementations, a distinctive feature of the signal and slot implementation in glueTK is that local signals are automatically distributed over the network and are available to all parts of the glue application. Therefore the way signals and slots are connected in most implementations:

```
connect(Object1, Signal1, Object2, Slot2)
```

Is extended to support the connection of signals and slots across several applications:

```
connect(App1, Object1, Signal1,
        App2, Object2, Slot2)
```

This allows to connect one specific signal to one specific slot. The identifiers in the connect command are just strings, so there is no need to have access to the objects in code, and signals and slots can be connected at any point, in any part of the glue application. In many cases it can be convenient to address all applications or all slots or all objects. To achieve this, any identifier in the connect command can be substituted by the wildcard character "*".

```
connect(App1, Object1, Signal1,
        *, *, hide())
```

Every interface element has default slots for common functionality, one of them is the "hide" slot, which hides the element when called. The above command connects a single signal to all "hide" slots of any object in any application and would hide all interface elements without addressing each individually. Independent of what machine an input modality is connected to, independent of where widgets reside, the communication is always based on this signal and slot system and that is all the developer has to deal with.

Widget transfer

As glue applications can spread across multiple screens, users should be able to move widgets across screens without worrying about the underlying technical details. This allows for example, to attach a personal menu widget to a person's position in the room and have the menu show up on the display closest to the person. Or to simply drag a widget from one display to another. Making this possible however, is not trivial, as screens can be connected to different machines. So moving a widget from one screen to another might actually require the widget to be transferred to a completely different machine. While the network transparent signal and slot system abstracts from the fact that a glue application runs across multiple machines to drive multiple displays with regard to the communication, signals are designed for low latency communication and are therefore not suitable for transferring large amounts of data. The transfer of a widget can be triggered explicitly or by setting its position relative to a different screen. It is then serialized as JSON data, attaching binary data as encoded strings. The data is transferred to the target machine via a network connection, where a new widget is then created by deserializing this data. To avoid delays in case of large attached binary data (large textures for example), widgets have a flag which, when set, causes them to be synchronized. This means that every machine keeps a continuously updated copy of the widget. This way, if the widget is transfered across displays, the data is already available on the target display. This allows to move widgets across displays, no matter if these displays are connected to the same machine, running a single application or if they are connected to different machines, running different applications.

Signal manager

The signal manager is the link between the input and output layer. All signals from input modalities or other applications, that come in via the event manager, are passed to the signal manager. Using a mapping table, it looks up which slots are connected to incoming signals and calls the corresponding functions. The signal manager also offers the bidirectional interface for developers, to integrate the input modality abstraction of glueTK into existing applications.

For a glue application, the signal manager is extended to also drive connected displays and manage the rendering of widgets. It passes any signals emitted by the widgets to the event manager, so they are available within the whole glue application. As the signal manager is highly multi threaded, there is no limit to the number of simultaneous signals that can be passed on and handled, which is important when dealing with not only multi touch, but multi person, multi touch on large display walls.

GLUE OUTPUT

GlueOutput is responsible for creating and rendering user interfaces on displays. Despite the fact that these displays vary greatly in size, resolution and aspect ratio, the goal is to offer the application developer an easy and flexible way to create user interfaces for an arbitrary range of displays. For this reason, glueTK offers elementary building blocks, that allow for easy assembly of custom, interactive widgets that can be connected to input modalities as well as communicate with each other. While this requires additional work, it allows to build widgets specifically tailored towards the properties of the available input modalities as well as screen sizes and resolutions.

Widgets

In existing GUI toolkits, widgets intend to solve one specific input task, usually obtaining one well defined type of value from the user. There are common sets of widgets that provide input elements for the most common types of values. This causes users to get used to certain ways information is asked from them and makes them feel comfortable even with unknown applications as many parts of that application are in fact, well known. This kind of reuse is possible if the input modalities as well as the range of display sizes and resolutions stays the same. Of course it is possible to simply scale such widgets to any screen size, but that would most definitely not result in an optimal interface. Also, given the wide range of modalities glueTK supports, not all input modalities would work with widgets tailored towards mouse use.

In most cases, this means that custom widgets, to allow user input, have to be created. This way, the best possible interface can be created given the target displays and input modalities. To assist users in the creation of such custom widgets, glueTK contains building blocks for images, videos, text, maps and web content that can be easily combined. Every building block has a set of default properties such as position, size, rotation, etc. so that placement and manipulation is the same for all types of building blocks. Each widget created from one or more of these building blocks, has the ability to offer slots to allow access to its functionality. These slots can be connected to signals from input modalities to, for example, set the widget's position to that of a person in front of the display to make the widget follow the person.

Animation manager

The animation manager allows to apply a predefined set of animations to any widget to provide visual feedback to the user. In addition, every property of a widget can be manipulated with custom animations to create more appealing interfaces. Animations are applied by calling a slot in the animation manager, which allows their use at any point in the glue application, even across displays and machines. When dealing with input modalities that are not universally known, providing this kind of feedback is important as users tend to need more guidance.

EVALUATION AND EXAMPLES

To demonstrate the functionality of glueTK, an application for a next generation control room has been implemented. In addition to this overview of functionality that glueTK enables, we evaluated the network transparent signal and slot system with regard to its performance, as it is an important aspect of the high flexibility described above. It is therefore important that its performance does not constitute a bottleneck for the framework. We did not evaluate the rendering performance since we rely on the external Clutter library [1]. As Clutter is OpenGL based, the performance is highly dependent on the utilized hardware and display resolutions. We do however drive displays up to 2160p with a single NVIDIA GTX480 and up to eight XGA displays using two low end NVIDIA GF9600GT without having any performance issues.

Performance evaluation

There are four ways signals are used within glueTK: from event handlers to widgets of the same application, from widgets to other widgets within the same application, from event handlers to remote applications and from widgets from one application to a remote application. There is, however, no difference internally between sending a signal from an event handler to a widget or from another widget. For evaluation we therefore differ between internal signals within the same application and remote signals, between separate applications that run on different machines. Most signals are lists of parameters which result in rather small amounts of data which have to be transferred. For evaluation, we chose a typical signal size of 342 characters ($\sim 0.33kbyte$) and for comparison a 76480 characters signal ($\sim 74kbyte$ or about twice the text of this paper). For remote signals we used two machines in a switched gigabit ethernet network. We transfered 100000 signals 10 times. The time from sending a signal to triggering the target slot was measured. The following table shows the averaged results as the number of signals per second.

type/size	342 characters	76480 characters
local	115500.11sps	111844.31sps
remote	6690.75sps	407.84sps

Considering that most input modalities, which usually produce higher traffic than inter widget communication, update

with 50Hz or less, the performance of the network transparent signal and slot system leaves enough room to not cause any significant latency.

Control Room Application

While glueTK is a generic framework which has been used to build different applications, we present a control room application as a specific example. The task of a control room is the planning and dispatching of relief squads, to deal with disaster situations. Each user is typically assigned a role according to the continental staff system which assigns a certain set of tasks and responsibilities. Cohen at al. [6] have shown early on that technology can assist users in command and control environments. So unlike current, rather low tech rooms, the goal of the "smart" control room is to allow multi modal interaction with a multitude of displays and devices to make the use of technology more intuitive and efficient. In the following, an example scenario will be described that shows some aspects of the application and how glueTK manages a wide variety of input modalities ranging from person tracking to head pose estimation and display devices from a $9cm$ smartphone to a $427cm$ video wall. We will first describe the involved hardware and software components.[1].

The main display is a $4m \times 1.5m$ back projection video wall with a resolution of $4096px \times 1536px$ which serves as an overview of the whole geographic region of the area of authority. A $0.9m \times 1.2m$ digital back projection table with a resolution of $1400px \times 1050px$ serves as an interactive workstation for specific assignments. Tablet computers can be used on the table as high resolution digital magnifiers and annotation boards as described in [10]. On a total of 4 machines (Intel QuadCore 2.4Ghz) with 10 cameras (9 Axis 211a, 1 Logitech Quickcam Pro), we have six computer vision components running:

A commercial face recognition system [2] using a webcam attached to the right side of the video wall. Four cameras around the video wall are used to build a 3D reconstruction of the area in front of the video wall and derive pointing gestures from that information as described in [22]. A person tracking component utilizing a camera with a fish eye lens in the ceiling can track multiple persons throughout the whole room. At the digital table, a stereo camera set up allows the use of not only pointing gestures, but also the detection of more detailed hand gestures as described in [21]. A marker tracker can locate and identify tablet computers and smart phones on the digital table using a MCMXT-Marker [10] based tracker. Finally, a head pose estimation system [25] uses a camera that observes the person standing in front of the digital table and analyses the persons' focus of attention.

The task of the glueTK application is to make use of all the information these components provide and create user interfaces on the displays to assist users in assessing the situation and plan missions for the relief units, based on the information about incidents coming in. One example usage of the application will be described in the following, highlighting what happens behind the scene to make the interactions possible.

[1]A video of the smart control room application is available at: http://www.youtube.com/watch?v=cTDqDbBrysk

Figure 4. Visual feedback indicating the face identification in progress.

The video wall displays an overview map of a city while the digital situation table displays a small subsection of the map in greater detail. A person, S1, is standing at the digital situation table as another person, S2, comes in. S2 starts his shift, and needs to get an overview at the videowall of what is going on in the city. To be able to use the system, he has to identify himself by looking into the webcam attached to the side of the video wall. The face identification component sends out status information like *face_detected*, *identification_in_progress* and *successfulid_idX* or *unknown_person*. A face identification event handler receives these status messages via the network and makes them available as signals in the glueTK application. A face identification widget gives the user visual feedback about the ongoing identification progress by providing slots to display banners (see Figure 4) that are connected to these signals. A personal menu widget has a slot that is connected to the *successfulid_idX* signal, which causes the personal menu belonging to S2 to appear right in front of him after identification.

To make the personal menu appear right in front of S2 it needs information about the person's location as well. To provide this information as signals, so it is readily available to the widgets, there is a tracking event handler that receives updates about the locations of all persons in the room. At the video wall, since it is such a large display, the location information is not only used to make the menu appear at the person's location initially but it also follows the person as he walks along the video wall.

The map used on the video wall is rather minimal to avoid clutter on the overview. However, S2 can get more detailed information by selecting a map overlay tool from his personal menu. To do this, he simply points to the according icon in his personal menu (Figure 5).

A few things happen in the background at this point. First of all, there is an context handler for the pointing gesture component that detects dwell based clicks, while giving visual feedback as described above. As mentioned before, the personal menu follows the person along the video wall. This can be a problem if the person tries to select an item from the menu. To make this easier the pointing signal is not only con-

Figure 5. Pointing gesture to select items in the locked menu.

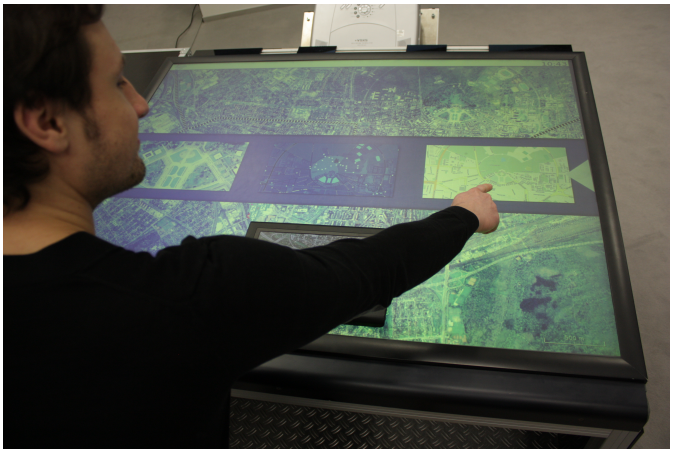

Figure 6. Map layer selection with pointing gesture on the digital table.

nected to the cursor but also to the personal menu, that will lock in place if it is pointed at. The map overlay that appears, displays a more detailed map in a rectangular region around the pointing direction and moves with the pointing gesture across the whole wall. With another click the overlay can be dismissed.

Now that S2 got an overview, he joins his colleague at the table and walks from the video wall to digital table. The personal workspace automatically follows S2 from the video wall to the table. A context handler uses the information about the person's position and the displays' positions, to generate signals like *at_wall* or *at_table* which are connected to corresponding slots of the personal menu widget and trigger a display transfer. Here, glueTK abstracts away from the fact that these are different displays driven by different applications running on different machines. The user has a coherent experience of interacting with one room-embracing glue application.

While S1 and S2 are at the table an alarm message about an explosion is displayed at the video wall. Both go up to the video wall and their respective personal workspaces appear in front of them. They can now collaborate at the wall by using additional tools from their workspaces to analyse the alarm with respect to the overall situation.

To actually plan the deployment of relief squads in detail, they move back to the table. To signal anyone looking at the overview wall that this alarm is taken care of, one of the operators loads the alarm message into his personal workspace using a pointing gesture. By doing so, the event is now assigned to him and as soon as he arrives at the digital table, it moves and zooms the map to the location of the explosion.

At the digital table, special hand gestures allow different map manipulations. With all five fingers stretched out the map can be moved, with two fingers stretched out the map can be zoomed with both hands, and a single stretched out finger triggers a click signal (see Figure 6).

Since glueTK gets information about the head pose orientation of the person in front of the digital table, it can keep track of what display the operator currently looks at. While both are focused on the table, planning the deployment of relief squads, another alarm comes in which is only displayed on the table. If the operator in front of the table looks up to the video wall, the alarm is displayed there as well with additional information to help him understand the global context. The operator can then point from the table to the alarm on the wall to acknowledge the message. Even so, this pointing gesture is detected by the cameras above the table, which are connected to the machine that drives the table display, glueTK makes this information available globally, so it can be used to interact with the video wall as well.

Another operator S3 comes in with a smart phone to transfer some images he took to the video wall. He can control a special cursor on the wall using the phone's gyroscope and transfer the pictures by clicking on them. Of course the transfer works both ways, so to get the deployment blueprint S1 and S2 created on the table, the smart phone can be placed on it, where it is located and identified using a marker on its back. The transfer is initiated by placing one finger on the table and another on the smart phone.

All of the described components and functionality is implemented and works in real time. The full glue application has been presented at CeBIT 2011 and proven to work within the challenging conditions of an exhibition booth.

CONCLUSION

In this paper, we presented a framework for creating multi modal applications that can span across multiple displays and machines. A generic approach is used to handle any input modalities, not limited to touch like modalities, they are made available through a single coherent interface. It allows for bidirectional communication to enable the use of context information. As many modalities targeted do not provide touch like data, this context information is important and input data can not simply be used to emulate a mouse as it is common practise in many frameworks with a focus on point and touch input. This abstraction layer can be used independently in existing applications without drawbacks, to integrate new input modalities. The presented framework assists in the creation of interfaces for a wide range of display sizes, aspect ratios, and resolutions. The distribution of user interfaces across displays

and machines as well as all communication, is transparent to the application developer as well as the end user. In an example application for crisis response rooms, we demonstrated the integration and interaction of many input modalities like gestures, person tracking, face identification, headpose orientation, a gyroscope as well as the utilization of multiple displays in a room. While glueTK has already been used by other people than the original developers, in the future we would like to make glueTK available to a greater audience. As for extending glueTK, we will explore ways of providing more complex widgets, which are still usable independent of screen sizes and connected modalities, and further improve the use of context information as this becomes more important in such complex environments.

Acknowledgements

This work is supported by the Fraunhofer-Gesellschaft Internal Programs under Grant 692 026.

REFERENCES

1. Clutter toolkit. *http://www.clutter-project.org* (accessed October, 2012).
2. Videmo face sdk. *http://videmo.de/products* (accessed October, 2012).
3. Ahlström, D., Hitz, M., and Leitner, G. An evaluation of sticky and force enhanced targets in multi target situations. In *4th Nordic conference on Human-*, no. October (2006), 14–18.
4. Bragdon, A., Deline, R., Hinckley, K., and Morris, M. R. Code Space : Touch + Air Gesture Hybrid Interactions for Supporting Developer Meetings. In *ITS* (Kobe, Japan, 2010).
5. Brooks, R. A. The intelligent room project. In *Proceedings of the 2nd International Conference on Cognitive Technology*, CT '97 (Washington, DC, USA, 1997), 271–.
6. Cohen, P. R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L., and Clow, J. Quickset: multimodal interaction for simulation set-up and control. In *Proceedings of the fifth conference on Applied natural language processing*, ANLC '97, Association for Computational Linguistics (Stroudsburg, PA, USA, 1997), 20–24.
7. Echtler, F., and Klinker, G. A multitouch software architecture. In *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*, NordiCHI '08, ACM (New York, NY, USA, 2008), 463–466.
8. Fernandes, V., Guerreiro, T., Araújo, B., Jorge, J., and Pereira, J. a. Extensible middleware framework for multimodal interfaces in distributed environments. In *Proceedings of the 9th international conference on Multimodal interfaces*, ICMI '07, ACM (New York, NY, USA, 2007), 216–219.
9. Flippo, F. A Framework for Rapid Development of Multimodal Interfaces. In *Design* (2003).
10. Geisler, J., Eck, R., Rehfeld, N., Peinsipp-Byma, E., Schutz, C., and Geggus, S. Fovea-tablett : A new paradigm for the interaction with large screens. In *Human Interface and the Management of Information.*, vol. 4557 of *Lecture Notes in Computer Science*. 2007, 278–287.
11. Gokcezade, A., Leitner, J., and Haller, M. Lighttracker: An open-source multitouch toolkit. *Comput. Entertain. 8*, 3 (Dec. 2010), 19:1–19:16.
12. Hansen, T., Hourcade, J., Virbel, M., Patali, S., and Serra, T. PyMT: a post-WIMP multi-touch user interface toolkit. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ACM (2009), 17–24.
13. Hoste, L., Dumas, B., and Signer, B. Mudra: A Unified Multimodal Interaction Framework. In *Proceedings of ICMI 2011 13th International Conference on Multimodal Interaction* (2011), 97–104.
14. Johanson, B., Fox, A., and Winograd, T. The Interactive Workspaces project: experiences with ubiquitous computing rooms. *IEEE Pervasive Computing 1*, 2 (Apr. 2002), 67–74.
15. Johnson, J., Roberts, T. L., Verplank, W., Smith, D. C., Irby, C. H., Beard, M., and Mackey, K. The xerox star: A retrospective. *Computer 22*, 9 (Sept. 1989), 11–26, 28–29.
16. Kaltenbrunner, M., and Bencina, R. reactivision: a computer-vision framework for table-based tangible interaction. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, TEI '07, ACM (New York, NY, USA, 2007), 69–74.
17. Krumm, J., Harris, S., Meyers, B., Brumitt, B., Hale, M., and Shafer, S. Multi-camera multi-person tracking for EasyLiving. *Proceedings of the third International Workshop on Visual Surveillance* (2000), 3–10.
18. Lalanne, D., Nigay, L., Palanque, P., Robinson, P., and Vanderdonckt, J. Fusion Engines for Multimodal Input : A Survey. *Interfaces* (2009), 153–160.
19. Melchior, J., Grolaux, D., Vanderdonckt, J., and Van Roy, P. A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications. In *Proceedings of the ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '09 (New York, NY, USA, 2009), 69–78.
20. Oviatt, S., and Cohen, P. Perceptual user interfaces: multimodal interfaces that process what comes naturally. *Communications of ACM 43*, 3 (Mar. 2000), 45–53.
21. Peinsipp-Byma, E., Geisler, J., and Bader, T. Digital map and situation surface: a team-oriented multidisplay workspace for network enabled situation analysis. J. T. Thomas and D. D. Desjardins, Eds., vol. 7327, SPIE (2009), 732703.
22. Schick, A., Camp, F. v. d., Ijsselmuiden, J., and Stiefelhagen, R. Extending touch: towards interaction with large-scale surfaces. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ACM (2009), 117–124.
23. Serrano, M., Nigay, L., Lawson, J.-y. L., Ramsay, A., Murray-smith, R., Denef, S., and Augustin, S. The OpenInterface Framework : A tool for multimodal interaction. In *Design* (2008), 3501–3506.
24. Shen, C., Vernier, F. D., Forlines, C., and Ringel, M. Diamondspin: an extensible toolkit for around-the-table interaction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04, ACM (New York, NY, USA, 2004), 167–174.
25. Voit, M., Nickel, K., and Stiefelhagen, R. Neural network-based head pose estimation and multi-view fusion. vol. 4122, Springer (2007), 291–298.
26. Wang, X., Zhou, Q., and Xin, Y. The construction and application of multitouch interactive platform based on touchlib. In *Proceedings of the 2011 4th International Conference on Intelligent Networks and Intelligent Systems*, ICINIS '11, IEEE Computer Society (Washington, DC, USA, 2011), 153–156.
27. Weis, T., and Geihs, K. Components on the desktop. In *Proceedings of the Technology of Object-Oriented Languages and Systems*, TOOLS '00 (Washington, DC, USA, 2000).
28. Wu, A., Mendenhall, S., Jog, J., Hoag, L. S., and Mazalek, A. A nested api structure to simplify cross-device communication. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*, TEI '12, ACM (New York, NY, USA, 2012), 225–232.