The MICOLE Architecture: Multimodal Support for Inclusion of Visually Impaired Children

Thomas Pietrzak, Benoît Martin, Isabelle Pecci, Rami Saarinen, Roope Raisamo, Janne Järvi

The MICOLE Architecture: Multimodal Support for Inclusion of Visually Impaired Children

Thomas Pietrzak, Benoît Martin, Isabelle Pecci
LITA
Université de Metz, UFR MIM
F-57045 Metz Cedex 01, France
+33 3 87 31 56 27
{pietrzak, benoit.martin, pecci}@univ-metz.fr

Rami Saarinen, Roope Raisamo, Janne Järvi
Tampere Unit for Computer-Human Interaction
Department of Computer Sciences
FIN-33014 University of Tampere, Finland
+358-3-3551-4035
{rs, rr, jj}@uta.fi

ABSTRACT
Modern information technology allows us to seek out new ways to support the computer use and communication of disabled people. With the aid of new interaction technologies and techniques visually impaired and sighted users can collaborate, for example, in the classroom situations. The main goal of the MICOLE project was to create a software architecture that makes it easier for the developers to create multimodal multi-user applications. The framework is based on interconnected software agents. The hardware used in this study includes VTPlayer Mouse which has two built-in Braille displays, and several haptic devices such as PHANToM Omni, PHANToM Desktop and PHANToM Premium. We also used the SpaceMouse and various audio setups in the applications. In this paper we present a software architecture, a set of software agents, and an example of using the architecture. The example application shown is an electric circuit application that follows the single-user with many devices scenario. The application uses a PHANToM and a VTPlayer Mouse together with visual and audio feedback to make the electric circuits understandable through touch.

Categories and Subject Descriptors
H5.2 **[Information interfaces and presentation]**: User Interfaces. - *input devices and strategies*, *auditory (non-speech) feedback*, *haptic I/O*, *user-centered design.*

General Terms
Design, Human Factors

Keywords
Universal access interfaces, Multimodal input and output interfaces, Haptic interfaces, Distributed/collaborative multimodal interfaces.

1. INTRODUCTION
Technology and methodology related to software engineering have been greatly improved during the recent decade. The input and output peripherals have developed to support more modalities than are normally used in the typical graphical user interfaces thus allowing more versatile and natural communication between the human and the computer. Following the conception of object-oriented paradigm and the introduction of the Internet, distributed computing has become one of today's key technologies. These new advances allow us to seek out new ways to support the computer use and communication of disabled people. With the aid of the new interaction technologies and techniques visually impaired and sighted users can collaborate.

The goal of the MICOLE project (Multimodal Collaboration Environment for Inclusion of Visually Impaired Children) was to create a software architecture that makes it easier to create multimodal multi-user applications for visually impaired children. We chose visually impaired children as the focus group as they have a lack of information technology to support them and they get a great benefit from multimodal interfaces that allow them to actually use computers. The modalities used are visual, audio and haptic feedback. The framework is based on agents that communicate with each other using messages. All agents, local or remote, communicate through a shared communication bus.

The hardware used in this project includes a VTPlayer Mouse that has two built-in Braille displays, and several haptic devices such as PHANToM Omni, PHANToM Desktop and PHANToM Premium. We also used a SpaceMouse and various audio configurations, in addition to visual feedback. We studied various scenarios that are supported by the architecture: single user with single device, single user with many devices, two users with shared devices and many users with many devices.

In this paper we present the software architecture and an example of its use. The example given is an electric circuit application that follows the single-user with many devices scenario although it can also be used in the many users with many devices scenario. The application uses a PHANToM and a VTPlayer Mouse to make the electric circuits understandable by touch.

Although the multimodal agent architecture brings added complexity and execution overhead we feel that its positive effects exceed the negative ones. By using agents we can incrementally and easily add new features and functionality in the framework. Agent systems are also scalable and can be set up in various

structures. The agents and their interconnections can change rapidly: new agents are started and some agents are shut down depending of the state of the application. Since multimodal applications are usually multi-threaded the agent-based approach fits well in their implementation.

2. RELATED WORK

There is earlier work on the use of haptic and other feedbacks to support missing or partially impaired senses. For example, Jansson and Billberger [13] studied the properties of the PHANToM device and came into the conclusion that visually impaired persons can identify 3D objects faster with their hands than with the device. With practice the performance improved somewhat. Magnusson *et al.* [16] found out that blind users can navigate in virtual environments provided that the environment is realistic. Blind users can also recognize quite complex virtual objects. Sjöström [28, 29] studied non-visual haptic interaction using the PHANToM device. He came up with a list of design guidelines for one-point interaction haptics which include guidelines for navigation, finding objects and understanding objects.

The GRAB project [6] developed a three-dimensional multimodal environment for visually impaired and blind people. The modalities used were touch and audio (for input and feedback). The application was used with a force-feedback haptic device that was used with two fingers. The GRAB system uses two-handed interaction which makes identifying objects and orientation easier for the blind. The GRAB device has been used in several other projects such as the HAPTEX project [11] that uses the device to enable manipulation of virtual textures and PURE-FORM [24] virtual museum that makes it possible for the user to touch the virtual representation of the sculptures.

Agents allow autonomous execution of multiple tasks making monitoring user actions easier. Most of the agent platforms follow a structure where a central server handles bootstrapping and connecting the various computers and agents together. Agents usually reside in specific software containers that provide all the necessary services such as communication interfaces and agent lifecycle management. There is also an agent standardization program, Foundation of Intelligent Physical Agents (FIPA) [9], which specifies agent components and services.

Open Agent Architecture [19] (OAA) provides a distributed agent architecture especially targeted for multimodal user interfaces. The main emphasis is on speech recognition, gestures and pen input. The system follows the traditional structure with a central facilitator that handles and forwards the tasks that the agents publish to be carried out. OAA is a mature project and several applications have been built using it.

JADE is a Java-based agent architecture [2] that follows the FIPA 2000 specification [10]. The agent system can be divided between computers that run a specific agent container. The main container, Front End, provides the dictionary services and the communication facilities between other agent platforms. The agent is run in its own execution thread and it has its own message queue for incoming events. An agent is composed of different behaviors that will describe the agent's functionality. Only one behavior can be executed at any given time. Behavior can be one-time or repeating. It can also be simple or complex behavior.

IBM ABLE [3] is another Java-based FIPA 97 compliant agent system. Just like in any other agent system, the agents are run inside a specific container: AbleBeanContainer. However, what separates ABLE from other systems is that an agent is also an AbleBeanContainer. This means that an agent can have other agents inside it. There are three ways for agents to communicate with each other. Firstly, an agent can subscribe to observe another agent's state changes. Secondly, the agents can be chained together so that the output of one agent is directed to input of another one. This forms a structure much like the Pipes and Filters design pattern [4]. Finally, the agents can share properties that will be automatically updated for each agent.

Nigay and Coutaz presented a PAC-Amodeus architecture [20] that combines Arch model [1] and PAC design pattern [4]. PAC is a hierarchical agent model that divides the application in smaller functional parts. Each part is divided in three areas: presentation (UI), abstraction (functional core) and control (communication). Arch on the other hand is a conceptual model for interactive applications that divides the application in functional responsibilities starting from user interaction component and ending to application specific component. In the middle of these two extremes is dialogue component that joins the user interface to the application logic. In PAC-Amodeus the Arch model is built using PAC agents so that each component has its own set of PAC agents that communicate with the agents on the same component and with agents on the next functional level and vice versa. There are many extensions and additions to PAC-Amodeus.

The agent system built in University of Tampere [26, 27] focused on building an agent framework suitable for shared simulation systems that use the 3D haptic environment provided by PHANToM device and ReachinAPI. We used that work as a starting point in building the architecture.

There are quite a few message bus implementations available for use. The Channel library provides a lightweight template framework for asynchronous distributed message passing and event dispatching [5]. Similar in concept is the Ivy software bus [12]. However, Ivy is better suited for more traditional Internet software rather than distributing high-speed data with heavy load, such as real-time GUI events and device coordinate management.

3. SOFTWARE ARCHITECTURE

The software architecture we describe in this paper is meant to be used to develop multimodal and collaborative software for the visually impaired. We are especially interested in situations where a visually impaired and a sighted child use the software collaboratively when sharing the computer together. For example, the visually impaired child could use the PHANToM device and the sighted user could use the mouse and the keyboard.

The haptic devices currently supported by the architecture are the ones supported by the Reachin API [25] and our own libraries. The first class of devices contains the SensAble PHANToM devices: 3D force feedback devices with 6 degrees of freedom input. We used PHANToM models with 3 degrees of freedom force feedback, such as the PHANToM Omni (see Figure 1), PHANToM Desktop and PHANToM Premium 1.5. In addition, Reachin API also supports other similar devices, such as ForceDimension Delta, ForceDimension Omega, and Novint Falcon.

Figure 1. PHANToM Omni

The VTPlayer mouse is the other supported class of haptic devices that provide tactile feedback. It is a mouse with four buttons and two 4x4 pin matrices (see Figure 2). It can be used to display patterns, felt by the tip of the fingers.

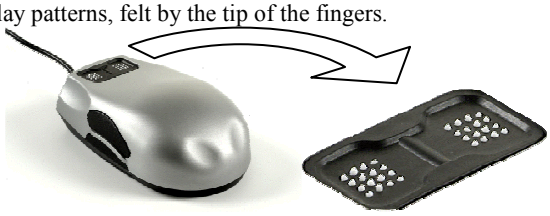

Figure 2 . VT Player Mouse

The architecture supports speech synthesis and multi-channel audio. Visual feedback is also supported as many visually impaired users still have residual sight. The software must also take care of the visual display as it is meant to be used both by visually impaired and sighted users.

3.1 Communication bus and base agents

The architecture is based on a multi-agent system. An agent is a piece of software responsible for a task; for example, an agent can be responsible for getting the coordinates of the device.

An agent interacts with the other agents through a bus that is used as a communication channel. This is done by using a method provided by the abstract class Agent. The idea is that every piece of information potentially useful in several parts of the application is sent through the bus as ASCII strings that we call messages. A typical agent interaction is illustrated in Figure 3, where the agent A is sending a message about mouse coordinates: "IN MSE : pos=(5, 20);". The syntax of the message denotes the type of the message. Then, messages are routed to all agents. To receive a message, an agent must use bindings which consist of a list of regular expressions linked to callbacks. In Figure 3, agent B defines the regular expression "IN MSE : pos=((.*), (.*));" used in a callback to catch mouse coordinates that flow on the bus. The agent A sends the message at time t_1, and the agent B catches it at time t_2, because the message matches one of its bound regular expressions. The callback of an agent is called whenever a message matching the associated regular expression is sent on the bus. The regular expression filters out all extra data leaving the meaningful data to be handled it the callback function.

The bus can be viewed as a piece of software that manages a list of bindings, and calls the agent's callbacks whose regular expression matches with the messages sent on the bus. The messages are managed immediately after being sent on the bus. If several bindings match the message, they are called in sequence in the order of the declaration of the bindings. On the other hand, if no binding matches the message, nothing happens.

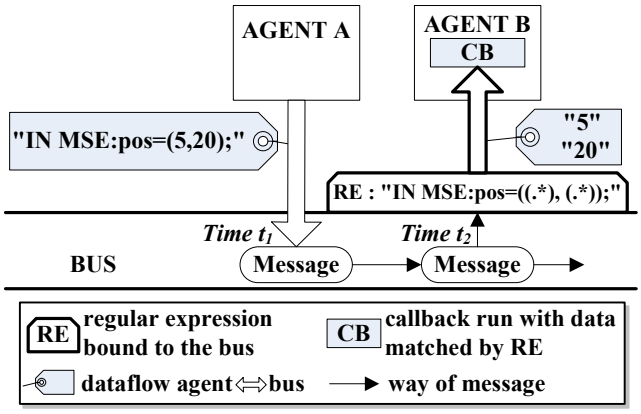

Figure 3. Links between the bus and agents.

There is only one bus per application instance. It can be used alone, or connected to the other ones. In the latter case, messages that have to be shared by all the busses on the network must be prefixed by "NET:". So these messages are routed to all the other computers linked in the system. The application must know the server address to connect to the network. If no server is found at this location a local server is created, and other applications can connect to it.

There are two categories of agents in the architecture: non-threaded and threaded (see Figure 4). Non-threaded agents only react to messages thanks to their bindings. They can send result messages to the bus. Threaded agents have their own execution loop that works in parallel with the rest of the program. They are useful for agents that poll input devices, for example. Of course, they can also receive messages from the bus based on the bindings.

3.2 Multimodal integration

In the context of multimodal interfaces, we need to manage several devices for input and output. In our architecture, the interaction with the devices uses three kinds of agents: two are in the device side and one in the application side. Their purpose is to free the application developer from using a low-level API for each device (see Figure 4). Depending of the capacities of the device, the developer only has to know which agents to use and the syntax of the messages that will be used from and to these agents.

To add an input/output device X in the architecture, the developer must implement an XSenderAgent and an XOutputAgent agent on the device side. The XSenderAgent is usually a threaded agent (see Figure 4) as its role is usually to poll the device to get some device information (mouse coordinates, mouse button state, keystrokes, etc.) through a driver or an API, and to send it on the bus so that the information can be used by all the other agents. The XSenderAgent determines the syntax of the messages that come from the device X. In some cases, it can be implemented with a non-threaded agent depending of the polling policy of the device.

The XOutputAgent describes the way to interact with a feedback device. Usually it gives functionalities that make use of a device driver or an API to display information on device X (force rendered on the PHANToM, a sound played on a speaker, a pattern on a pin matrix, etc.). These functionalities are

implemented through a callback bound to the bus. The XOutputAgent determines the syntax of the messages that come to the device X. An application developer can use these functionalities by sending specific messages on the bus. Thanks to the bindings of the XOutputAgent, the right functionality will be run through the callback.

In the application side, the messages that come from the XSenderAgent must be handled. There are two ways to do this. In the first one, the application developer can implement a separate agent and bind the messages needed. By this way, the developer has a total control. In the second one, the architecture offers an abstract class XInputAgent. It has callbacks matching all messages sent by the XSenderAgent, and stores different data. As it is an abstract class, it needs to be inherited to actually use the stored data. When the data changes, some abstract methods are called. By this way, the application developer has less code to write but also smaller liberty. When a developer creates a sender agent for a device, he or she should create an abstract input agent so that anybody who needs the information from this device will only have to inherit this agent.

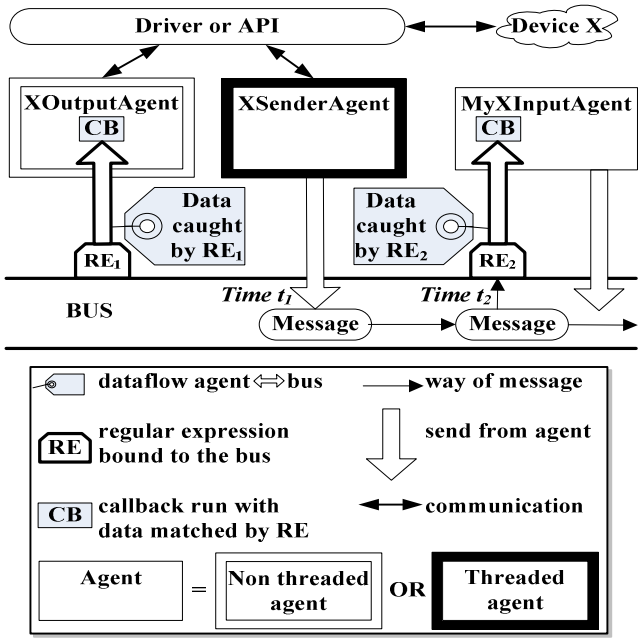

Figure 4. Types of agents to manage a device

4. AGENTS

The architecture provides agents for several input and output devices: a keyboard, a regular mouse, a PHANToM and a VTPlayer. For the PHANToM devices and some other devices we based our library on the Reachin API [25].

4.1 PHANToM agents

The PHANToM is handled through the Reachin API. This API has a scene graph, that joins the devices (like the PHANToM) and the objects the devices interact with. Each object or device is a node of the scene graph, and can be linked to each other directly or through scripts. These scripts, including the PHANToM coordinate updates, are evaluated in a loop, called the scene graph loop, running as fast as possible (usually at least 60 Hz). The forces applied on the PHANToM must be computed at 1000 Hz to have a smooth feeling, so the Reachin API provides a second loop called the realtime loop, running at 1000 Hz. The programmer can access this loop using force models. A force model consists of a function which is aware of the PHANToM position and computes the forces to be applied. A linear force towards a point is a simple example of a force model. The API allows creating several force models. In that case the developer can define weights, and the API will compute the force to be applied to the PHANToM by adding all the forces given by the force models, multiplied by the weights.

The PhantomSenderAgent is a threaded agent that polls the Reachin API at a fixed frequency (120 Hz by default), and sends the position on the bus with a motion message. When a button is pressed or released, the coordinates and the button state is sent on the bus with a button message. Thus, the PhantomInputAgent has onMove, onButtonUp and onButtonDown virtual methods to make the interactions easier with the PHANToM. The measurements used in the Reachin API are always in meters. We used the same measurements in the architecture.

Even though the forces must be computed at 1000 Hz, we can change the force models less often and still have a smooth feeling. The role of the PhantomOutputAgent is to provide several kinds of generic force models. The application developer just has to send a message on the bus with the right parameters and the PhantomOutputAgent will apply the corresponding force model to the PHANToM. The new force model will be evaluated at 1000 Hz until it is changed.

If the developer wants to create a new force model, he or she has to inherit the PhantomOutputAgent, implement the force model (a callback function) and do a binding to be able to activate it by a message. The architecture includes basic force models: lines, points, discs, spheres, etc. as well as two higher level force models we will describe next: PICOB and guiding.

4.1.1 PICOB

The PICOB (Post-It haptiques par COdes-Barres) force model [21] consists of two kinds of haptic guiding cues using bumps (Figure 5). The idea is to give information with force feedback.

The first one is a fully guided mode: the user is dragged back and forth by some millimeters with the PHANToM to provide some directional information. There are three parameters for these bumps: direction, amplitude and duration. It has previously been shown that users can recognize 6 directions (up, down, right, left, forward and backward) and 2 amplitudes (4 mm and 16 mm) [21], with a duration fixed at 250 ms. In the architecture it is implemented as a point force model whose position is computed in the realtime loop according to the time. The programmer using the architecture can create a sequence of bumps, and choose the values for the three parameters by sending a message. Below is an example of a message to create a guided sequence of bumps at location (0.2, 0.1, 0), with two 0.1 seconds long bumps on the x axis, which lasts for 0.6 seconds with a pause of 0.2 seconds between the bumps:

```
OUT PHTM : picobg=(0.2, 0.1, 0);
bumps=((0.1, 0, 0),(0.1, 0, 0));
bumplength=0.6; pauselength=0.2;
```

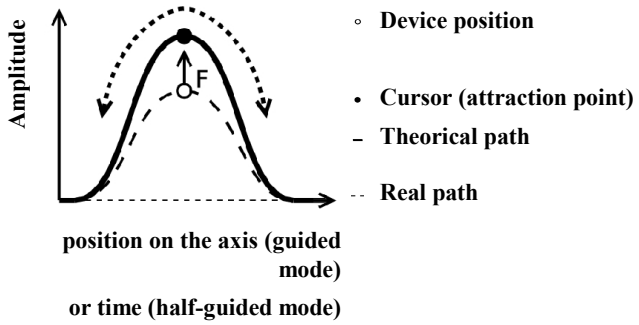

Figure 5. Bumps used in PICOB interaction

The second mode is half-guided where the user feels some bumps while moving along a line. The parameters are quite the same as those of the previous mode: direction, amplitude and width. The directions are more limited than in the other mode because we can't use the directions of the line. According to earlier studies users can recognize 4 directions (up, down, forward and backward, the line being from left to right) and 2 amplitudes (4 mm and 16 mm) [22], with a width fixed at 5 mm. They have been implemented as a point force model whose position moves according to time, and a line force model with a deviation in the bump position. The application developer can use sequences of bumps with a message on the bus, and choose the orientation of the line, the orientation of the bumps, their length and their width. Below is an example of a message used to create a line force model with two bumps at the center between the positions (-0.5, 0, 0) and (0.5, 0, 0), with the bump width of 5 mm and a pause of 2 mm between them:

```
OUT FF3D : picobhg=((-0.5, 0, 0),(0.5, 0, 0)); bumps=((0, 0.004, 0),(0, 0, 0.016)); bumplength=0.005; pauselength=0.002;
```

4.1.2 Guiding

The goal of the guiding force model is to bring the user to a fixed position while he is exploring a shape. It can be used for example in a multi-user situation where a teacher wants to show something to a student: the teacher uses the mouse to guide the student who uses a PHANToM that utilizes the guiding model force.

The guiding force model is an additional force model. It is managed by the PhantomGuidanceAgent that creates a force model in addition to the force model generated by the PhantomOutputAgent. The user can feel the shape defined by the main force model and is being dragged to a point at the same time. For example, the main force model constrains on a horizontal line while a guiding point is moving along this line. Usually the attractive point must belong to the shape of the main force model. The guiding follows a linear track and fits well with lines. It is usable with other force models but as the guiding is straight, it can disturb the exploration of an ellipse for example: the ellipse will likely be felt as a polygon. One solution is to guide only on short distances. This is an example of guidance message, where we guide to the point (0.1, 0.2, 0.3):

```
OUT PHTM GUIDE : point=(0.1, 0.2, 0.3);
```

The force model uses the speed and acceleration of the PHANToM to produce a continuous force when moving the PHANToM as in [7]. The attraction is linear, with a fixed stiffness and if the distance to the attraction point is nearer than a fixed threshold the force decreases, as proposed in [17]. The stiffness and threshold can be changed by changing variables with a simple message sent to the PhantomGuidanceAgent.

4.2 VTPlayer agents

The VTPlayer Mouse is controlled with the libusb [15] through an API developed at the University Paul Verlaine - Metz [18]. When using this API instead of the official driver, the mouse doesn't use the system's mouse cursor, so we can easily use the VTPlayer and a regular mouse at the same time.

The API allows getting the mouse position and the state of the buttons. It allows displaying any 4x4 pattern on each of the two cells. It is also possible to display animations, which are pattern sequences displayed at a speed defined by the developer. The animations are called dynamic icons; with respect to simple patterns we call static icons.

From the VTPlayer side, the VTPlayerSenderAgent is a non-threaded agent which sends a state message on the bus whenever the VTPlayer moves or a button's state changes. The state is received with a listener of our VTPlayer API. The VTPlayerOutputAgent allows displaying any icon (static or dynamic) on the right or the left pin matrix of the device.

The abstract VTPlayerInputAgent is inherited from the AbstractMouseInputAgent, so it has the virtual methods: onMouseMove, onPressed and onReleased. The callbacks are implemented in the AbstractMouseInputAgent so that it can be used with the VTPlayer or the regular mouse messages. The VTPlayerInputAgent just do the binding from its messages to the callbacks.

4.3 Keyboard and Mouse agents

A regular mouse and a keyboard are only input devices. There are sender agents (KeyboardSenderAgent and MouseSenderAgent) and abstract input agents (KeyboardInputAgent and MouseInputAgent). As the ReachinAPI can also manage the regular mouse and the keyboard, it is used to access these devices. The KeyboardSenderAgent is a non-threaded agent that sends a message on the bus whenever a key is pressed or released. The MouseSenderAgent is a threaded agent that polls the Reachin API for the mouse position at a fixed frequency (50 Hz by default). A motion message containing the coordinates is sent on the bus. Similarly, a button message is sent if the user presses or releases a button on the mouse.

The input agents provide virtual methods that need to be inherited to perform action on the device events. The KeyboardInputAgent has onKeyPressed and onKeyReleased virtual methods, and the MouseInputAgent has onMouseMove, onButtonDown and onButtonUp virtual methods.

4.4 Miscellaneous agents

There are additional agents built to support application development. The most important ones are described in this section.

SpeechOutputAgent is an agent that can speak a given text string of contents of a file using the Microsoft Speech API (MS SAPI). Any MS SAPI compatible speech synthesis can be used.

The SpaceMouse has three associated agents following the input device agent structure. First, there is the MagellanStateSenderAgent that polls the actual device and sends the state changes to the message bus. Second, there is an abstract MagellanInputAgent that listens to the state changes of buttons, updates the internal button representation and finally calls an empty virtual onButtons method. Finally, there is the actual input handler agent, the MagellanModeSelectorAgent that extends the MagellanInputAgent and provides an application specific onButtons method implementation.

A separate lightweight and linear guiding agent was also created to allow a mouse user to guide another user with the PHANToM device to a certain point in virtual space.

5. AN EXAMPLE APPLICATION: ELECTRIC CIRCUITS EXPLORATION

This architecture was first used to write an Electric circuits schematics exploration software (see Figure 6). The purpose of this software is to use multimodal interaction to allow visually impaired users to have an access to electrical schematics. For that purpose, the PHANToM and the VTPlayer are used to display information in order to make the circuits understandable by touch. An electric circuit is composed of wires or components linked by nodes. Here we only consider 2D schematics on a horizontal plane.

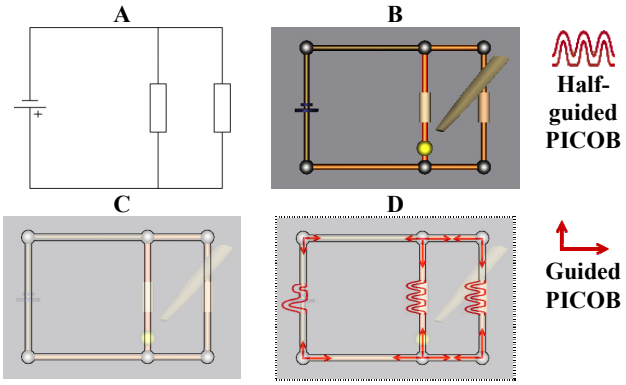

Figure 6. An example of circuit. A: visual schematic, B: schematic in the software, C: haptic constraints, D: schematic with PICOB information

5.1 Interaction techniques

The user is constrained into the circuit so he or she can feel its shape (C on Figure 6). In order to help him or her navigate we included some directional cues. We placed some guided PICOB at the intersections: the user is dragged by several millimeters to show him where he can go (arrows on D on the Figure 6). These cues are only played when the user asks for help with the PHANToM button, because according to discussions we had with users, they prefer exploring by themselves and only ask for help when they want it. We use two bump amplitudes so that we encode already explored directions with little bumps, and new directions with big bumps. The goal is to let the user know if he has already explored the whole circuit. The same information is displayed on the VTPlayer mouse, with directional icons (see Figure 7) [23, 8]. We use multimodality to give the user several ways to understand the information. Some user may prefer the force-feedback interaction, while others may prefer this tactile interaction.

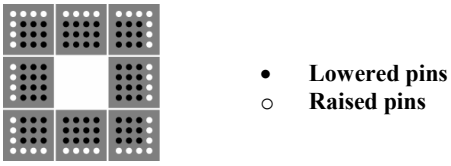

Figure 7. Directional icons on the VTPlayer

The electrical components are felt in another way: while the user is exploring a wire on the circuit, he or she can feel some bumps with the PHANToM that indicate there is a component. This is the half-guided PICOB interaction. The components are coded with bump sequences: one for a lamp, one small and one big for a battery, two for a capacitor, and three for a resistor (curves on D on the Figure 6). We also display the components on the VTPlayer: we designed an icon for each component. Once again we use multimodality (see Figure 8).

Like any software based on the architecture, this software is a collection of agents that work together. First we needed to choose the agents needed from the architecture. We wanted to use the PHANToM and VTPlayer output so we created an instance of the PhantomOutputAgent and one of the VTPlayerOutputAgent. We also used the guidance features so we created an instance of PhantomGuidanceAgent. Then we needed information from the PHANToM and the mouse so we instantiated the sender agents PhantomSenderAgent and MouseSenderAgent. The last agent provided by the architecture was the ReachinMicoleApplication which gives us the access on the Reachin scenegraph and runs the main loop.

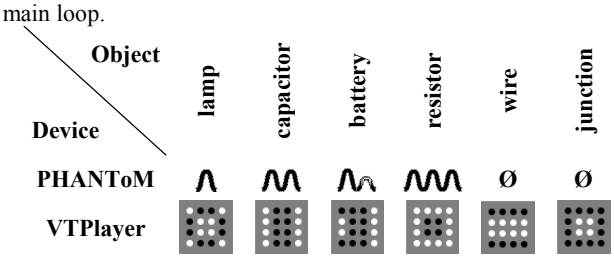

Figure 8. PHANToM and VTPlayer representation of electric circuit parts

We have inherited the MyPhantomInputAgent and MyMouseInputAgent to use the PHANToM and mouse coordinates.

5.2 Input Agents

The PHANToM is the main exploration device in our software, and the mouse is a secondary device used mainly for guidance. In order to constantly know the current position of the two devices in the schematic, we have to compute projections. So both the MyPhantomInputAgent and the MyMouseInputAgent will compute a projection of their device on the circuit, and take care that the cursor only moves from an object to an adjacent object (see Figure 9). These cursor positions are regularly sent on the bus.

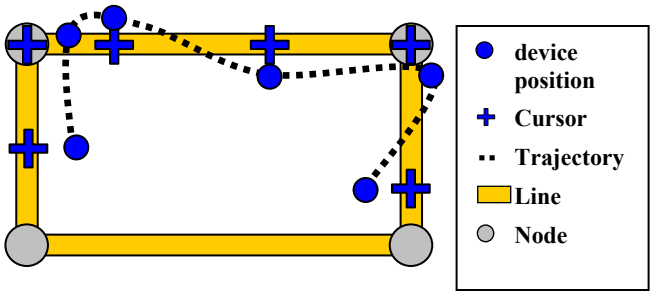

Figure 9. Exploration trajectory and projection

The MyPhantomInputAgent sends messages about the objects currently being explored so that the information can be displayed multimodally in various ways. For each output device or output modality used to display the information, we developed a rendering agent whose duty was to bind all the messages concerning the information it displayed, and to send messages to the correct output device.

5.3 Rendering agents

The visual representation of the circuit was managed by the Reachin API. The circuit didn't change during the execution. However, the bus was used to display cursors according to PHANToM and the mouse projection on the circuit.

Component characteristics sent by the MyPhantomInput-Agent were used by all the rendering agents, in particular to constraint the user on the circuit. The PHANToM rendering agent used the PhantomOutputAgent provided by the architecture to activate force models by sending messages to use the suitable force model. The line force model was used for wires, the disc force model for junctions and the half guided PICOB force model for components, using the codes defined in Figure 8.

The VTPlayer rendering agent used the component characteristics sent by the MyPhantomInputAgent to display an icon representing the component type, the junction or the wire. It sends messages to the VTPlayerOutputAgent to communicate with the device. The patterns used are shown in Figure 8.

5.4 Guidance agents

The guidance systems we have developed use the linear point guidance force model described in section 4.1.2. The principle of the guidance systems we used is to manage a list of waypoints. When the user was close enough to the next waypoint, the user was guided there. The trigger distance is a constant. In this software we wanted to constrain the movement based on the circuit, so we don't use directly the device coordinates as waypoints, but projections on the circuit. The force model is a linear force toward the next waypoint [17]. Two guidance modes have been implemented for this application:

- *The replay mode*: the idea is to record the path of a pointing device (mouse, PHANToM, etc.) in a file, and then replay it by moving a guiding point along this path. So there are two phases: the recording phase and the replay phase. In the recording phase we store the waypoints. They are recorded only if they are farther than a fixed threshold to prevent from recording too many points. The waypoints used are the cursor coordinates sent by the MyPhantomInputAgent. In the replay phase, we read the list of waypoints in a file.
- *The dragging mode*: dragging of the PHANToM with the mouse when the two phases happen at the same time. The points come from one device: the mouse (with the MyMouseInputAgent) and the guidance is given with the PHANToM. So the waypoints list is dynamic.

Even though these modes were developed for one application they are usable in several different kinds of new applications.

5.5 Evaluation and preliminary results

The Electric Circuits application was tested with 13 visually impaired children. Some of them were blind and others had residual vision. Typical circuits with components were presented to users and the goal was to evaluate the circuit's shape recognition and the component list recognition.

Users with partial vision did not have problems to recognize the circuit's shape: they often used their partial vision. Some blind users had problems to recognize circuit shapes for many reasons:

- Lack of spatial references: a circuit was explored completely, but a user had difficulties to link the different parts that he or she explored.
- Unable to explore the entire circuit: guided PICOB at the junction did not help much; few users used them or understood them.
- Difficulty to understand the distances: a user confused a square with a rectangle.

Half of the users used the VTPlayer icons to recognize components, and the other half used the half-guided PICOB with the PHANToM. The bump sizes of the PICOB appeared not to be suitable for all users and certainly need personal user settings. Unfortunately, no user asked to be guided, so this functionality has to be tested in further studies.

The implementation of the electric circuits application on top of the MICOLE architecture was a test to validate it. During the development and evaluation of the application we found some areas of improvement in the architecture. For example, the callback execution is currently blocking while a more suitable solution would be a threaded and parallel execution of the callbacks. This would also increase fault tolerance of the system.

The other improvement that needs to be done is to move the guidance agent of the application in the MICOLE architecture. It must be flexible enough to allow the programmer to have a maximal control on the guidance. Then it will have to deal not only with individual waypoints but also the guidance on the path between the waypoints.

6. CONCLUSION

We have presented a software architecture that aims at making multimodal multi-user applications easier to implement. The architecture supports visual, auditory, and haptic feedbacks, as well as several input devices like a mouse, a keyboard or a SpaceMouse. The architecture is scalable and easily extendable. For example, adding support for other devices, such as the Dot View tactile displays [14], is a matter of writing few additional agents. We presented an example of a multimodal application that uses this architecture to enable visually impaired children to learn about electric circuits.

7. ACKNOWLEDGMENTS

We would like to thank Jérôme Wax and Jouni Salo for their work on the architecture. This study has been funded by the European project MICOLE (IST-2003-511592 STP).

8. REFERENCES

[1] Bass L., Faneuf R., Little R., Mayer N., Reed S., Seacord R., & Szczur M. R. A metamodel for the runtime architecture of an interactive system. Technical report, 1992.

[2] Bellifemine,F., Poggi, A., and Rimaza, G.. JADE - a FIPA-compliant agent framework. *In Proceedings of the Practical Applications of Intelligent Agents*, 1999.

[3] Bigus, J. P., Schlosnagle, D. A., pilgrim, J. R.,. Mills III, W. N, and Diao, Y. Able: A toolkit for building multiagent autonomic systems. IBM Systems Journal, 41(3), 2002.

[4] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M.. A System of Patterns: Pattern-Oriented Software Architecture. John Wiley & Sons, 1996.

[5] Channel, a asynchronous, distributed message passing and event dispatching library. http://channel.sourceforge.net/

[6] Computer GRaphics Access for Blind people through a haptic virtual environment. http://www.grab-eu.com

[7] Crossan, A., Williamson, J. and Brewster, S. A General Purpose Control-Based Playback for Force Feedback Systems. In *proceedings of the Eurohaptics International conference EuroHaptics 2006*, Paris, France, July 3-6, 2006

[8] Crossan, A. and Brewster, S. A. Two-handed navigation in a haptic virtual environment. In *CHI Extended Abstracts 2006*, pages 676-681. Montréal, Canada, 2006

[9] Foundation for Intelligent Physical Agents. http://www.fipa.org/

[10] The foundation of intelligent physical agents (fipa) abstract architecture specifications, 2006. http://www.fipa.org/specs/fipa00001/SC00001L.html

[11] HAPTEX -- HAPtic sensing of virtual TEXtiles. http://haptex.miralab.unige.ch/

[12] Ivy Software bus. http://www.tls.cena.fr/products/ivy/

[13] Jansson, G. and Billberger,K., The PHANToM Used without Visual Guidance. In *The First Phantom Users Research Symposium (PURS 99)*. http://mbi.dkfz-heidelberg.de/purs99/

[14] KGS Corporation, Dot View tactile displays, http://www.kgs-jpn.co.jp/.

[15] LibUSB. http://libusb.sourceforge.net/

[16] Magnusson, C., Rassmus-Gröhn, K., Sjöström, C. and Danielsson, H., Navigation and recognition in complex haptic virtual environments - reports from an extensive study with blind users. In *Proceedings of Eurohaptics 2002*, University of Edinburgh, 2002. http://www.eurohaptics.vision.ee.ethz.ch/2002.shtml

[17] Magnusson, C., Danielsson, H. and Rassmus-Gröhn, K., Non Visual Haptic Audio Tools for Virtual Environments. Presented at the *Workshop on Haptic and Audio Interaction Design*, University of Glasgow, 31st August - 1st September 2006

[18] Martin, B., Pecci, I. and Pietrzak, T. Angle Recognition Cues using a new API dedicated to the VTPlayer Mouse. In *International Conference on Human-Machine iNteraction, HuMaN'07*, Timimoun, Algeria, 2007

[19] Moran, D. B., Cheyer, A. J., Julia, L. E., Martin, D. L. and Park, S., Multimodal User Interfaces in the Open Agent Architecture. *In Proceedings of the 2nd international conference on intelligent user interfaces*. ACM Press, 1997, 61-68.

[20] Nigay L. and Coutaz J. A generic platform for adressing the multimodal challenge. In In Proceedings of the SIGCHI conference on Human factors in computing systems, pages 98 -- 105. ACM Press/Addison-Wesley Publishing Co, 1995.

[21] Pietrzak, T., Martin, B. and Pecci, I.. Affichage d'informations par des impulsions haptiques. In *IHM 2005: Proceedings of the 17th French-speaking conference of human-computer interaction*, Toulouse, France, 2005

[22] Pietrzak, T., Martin, B. and Pecci, I. Information display by dragged haptic bumps. In *Enactive 2005: proceedings of the 2nd International Conference on Enactive Interfaces*, Genoa, Italy, 2005

[23] Pietrzak, T., Pecci, I. and Martin, B., Static and dynamic tactile directional cues experiments with VTPlayer mouse. In *proceedings of the Eurohaptics International conference EuroHaptics 2006*, pages 63-68. Paris, France, July 3-6, 2006

[24] PURE-FORM -- Museum of Pure Forms. http://www.pureform.org

[25] Reachin API 4.1 http://www.reachin.se

[26] Saarinen, R., Järvi, J., Raisamo, R. and Salo, J. Agent-based architecture for implementing multimodal learning environments for visually impaired children. *In Proceedings of the 7th International Conference on Multimodal Interfaces* (ICMI'05), pages 309-316. ACM Press, 2005.

[27] Saarinen, R., Järvi, J., Raisamo,R., Tuominen, E., Kangassalo, M., Peltola, K. and Salo, J. Supporting visually impaired children with software agents in a multimodal learning environment. *Virtual Reality*, 9(2-3), Springer_Verlag, London Ltd. 108_117, 2006.

[28] Sjöström, C., Designing haptic computer interfaces for blind people. In *Proceedings of Sixth International Symposium on Signal Processing and its Applications (ISSPA 2001)*. IEEE, 2001, 68-71.

[29] Sjöström, C., *Non-visual haptic interaction design: Guidelines and applications*. Doctoral dissertation, Certec, Lund Institute of Technology, 2002.