

MIEL: The Multi-user Interactive Environment Language

White paper. Fredrik Bridell, Peter Lundén, Olof Bendt
Interactive Institute: version 2.0.6 / 2007-06-14



Abstract

The Multi-user Interactive Environment Language (MIEL) is an interaction control system, primarily used to power interactive physical installations.

MIEL includes a special document format used to describe interaction, a graphical editor to make such documents, an interaction engine that runs the scripts, and an extensible device virtualization architecture that makes it easy to adapt the system to new sorts of devices.

This document is a primer that introduces some of the main ideas behind the system. It also shows you some examples of how it has been used and how it might be used in the future.

Contents

INTRODUCTION	3
HOOKING IT UP: THE DEVICE ARCHITECTURE	4
CASE: AVESTA STEELWORKS	6
OVERVIEW	6
THE EXPERIENCE	6
THE TECHNOLOGY	7
CASE: CHINA BEFORE CHINA	8
OVERVIEW	8
THE EXPERIENCE	8
THE TECHNOLOGY	9
MORE ABOUT MIEL	10
THE XML DOCUMENT FORMAT	10
THE EXTENSIBLE DEVICE ARCHITECTURE	10
THE MIEL GRAPHICAL EDITOR	12
DESIGNING AN INTERACTIVE SITE WITH MIEL	13
PLANNING THE SYSTEM	13
SETTING UP THE INFRASTRUCTURE	13
EXTENDING THE SYSTEM	13
MAKING THE SCRIPTS	13
WHAT MIEL HAS DONE	14
WHAT MIEL COULD DO	14
FUTURE USAGE SCENARIOS	14
HOW TO GET MIEL	15

Introduction

The Multi-user Interactive Environment Language (MIEL) is an interaction control system that facilitates the construction of interactive installations and environments, such as museum exhibitions and art installations. It is a generic system that can be expanded to accommodate any kind of inputs and outputs, such as different kinds of sensors and control panels, audio and video outputs, dimmable lights, movable curtains, animatronics, other software systems, etc.

MIEL consists of several parts:

1. The author/designer/programmer uses the **MIEL editor**, a graphical editor, to create the documents that describe the interaction. To ensure maximum flexibility the editor is not integrated into the interaction engine, but rather a separate application. You may use any other XML or text editor if you prefer.
2. The **MIEL interaction script document format** is the XML format used to write the actual interaction scripts. (“When the user enters here, this sound should start playing on these speakers and these lights should dim up...”)
3. The **MIEL interaction engine** is the system that actually “runs” the interactive environment by following the instructions in the XML documents. It connects to the physical devices and lets users interact with them.
4. The **extensible device architecture** is what allows the system to use all types of devices (lights, speakers, screens, different types of sensors, etc).
5. Last but not least, the system is built to model **the people using the system**. There are different ways of doing this, ranging from straight pre-programmed sequences to quite complex interaction scenarios.

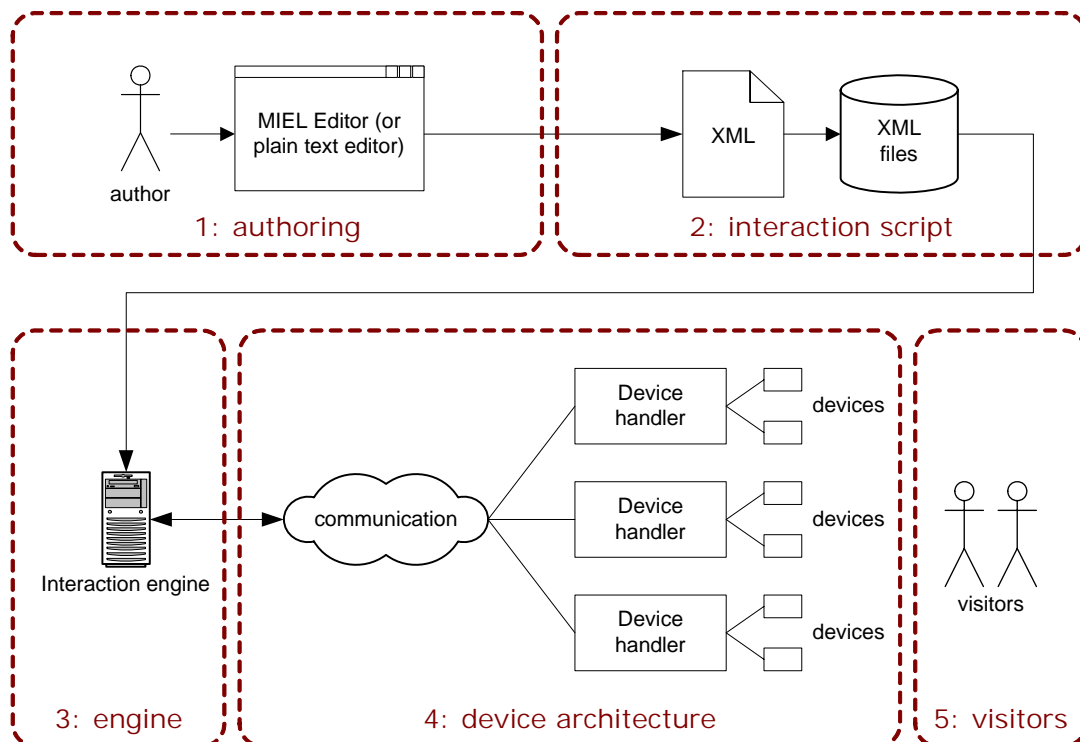


Figure 1. MIEL System overview (simplified)

Hooking it up: the device architecture

The image below (fig 2) shows a slightly more detailed overview of the MIEL system.

Here you can see some actual examples of the **device architecture** works. This has two main strengths: it is **extensible** and it uses a sort of **virtualization**.

To be **extensible**, in this context, simply means that it is simple to extend the system to allow new types of devices. A **device**, as we see it, is any item that the visitor will actually interact with in some way. This includes lights, speakers, screens and video projectors, different kinds of sensors (buttons, touch sensors,

infrared “burglar detectors”, remote-control type of inputs...) as well as any other kind of device you can imagine.

In an actual installation, these devices connect to the system in a variety of ways (speakers to amplifiers and sound players, screens to DVD players or computers, lights to switches or dimmers, sensors to whatever kind of hardware they need, and so on.) These things in turn communicate with the interaction engine (for example using a computer TCP/IP network or a serial cable). Each such subsystem that somehow allows the interaction engine to communicate with the devices we call a **device handler**.

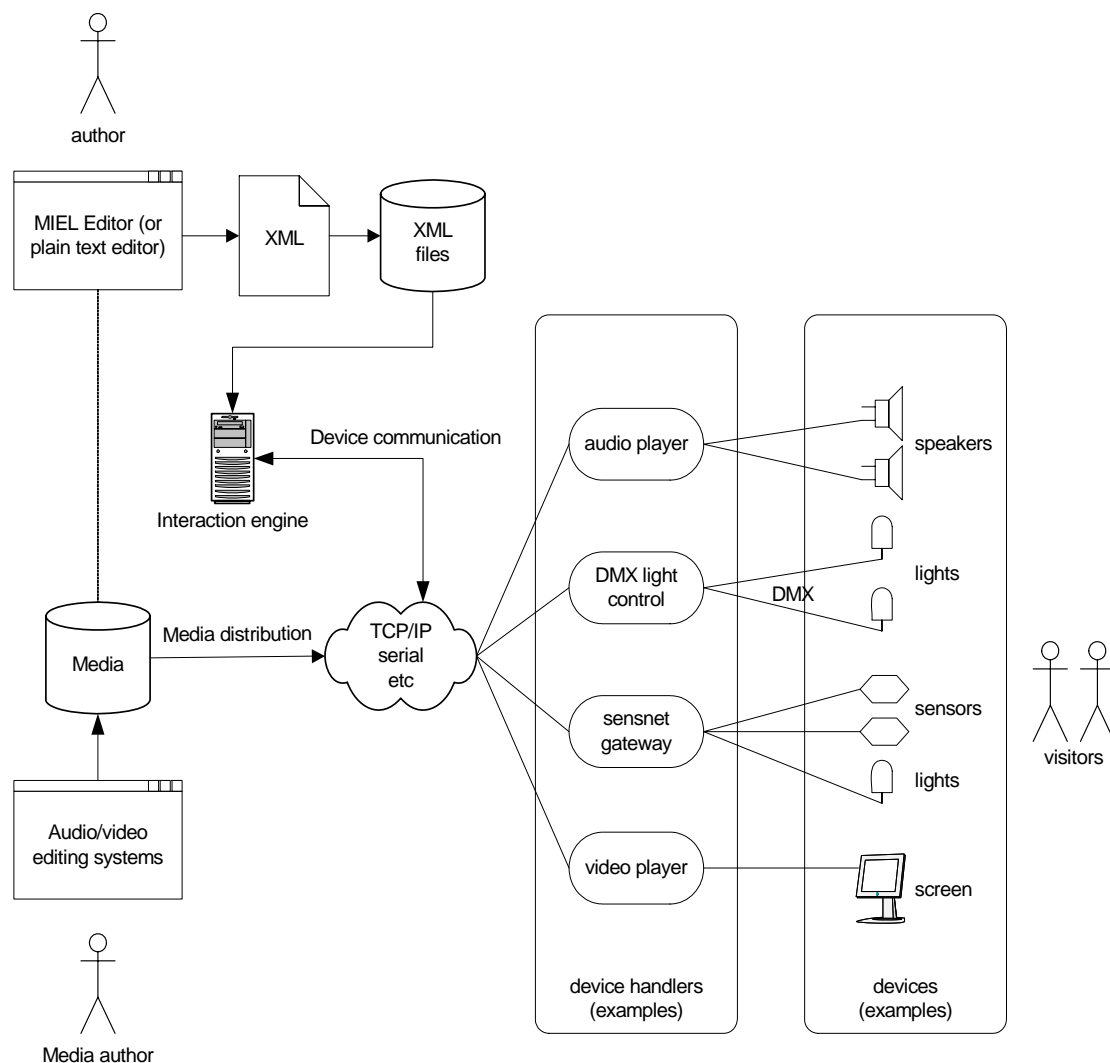


Figure 2. MIEL System overview (detailed)

The concept that devices connect to device handlers is a simple¹ but powerful idea that allows us to hide a lot of the actual wiring and network protocols. The purpose of this cloaking (or **virtualization** in IT business jargon) is that it simplifies things for the person authoring the interaction.

An example from the figure 2 (top right corner) is the pair of "speakers" connected to an "audio player". The audio player is not just any device capable of producing sounds. A further requirement is *that it can communicate with a computer* in some well-defined way. This is necessary, because we ultimately want to be able to say "play this sound on that speaker". This means that it must be possible to program a **device handler driver** (a piece of Java code that plugs into the interaction engine) that can control that actual piece of hardware.

The interaction author only needs to know that (for example) "there are a number of speakers here, each capable of playing any mono sound file". Using the MIEL editor (or hand-coding the XML document), the interaction author simply selects a speaker and a sound to play on it. *The interaction author thinks only in terms of devices* – how they connect to the system is not interesting at that point.

The next example in figure 2 is a DMX device handler and a number of DMX lights. DMX (or formally "DMX512/1990") is an industry standard in the stage and lighting business, used to control dimming and moving lights, moving pieces on theatre stages etc. This is an example of how the MIEL architecture has been expanded to embrace existing standards. Since DMX is the de facto standard used

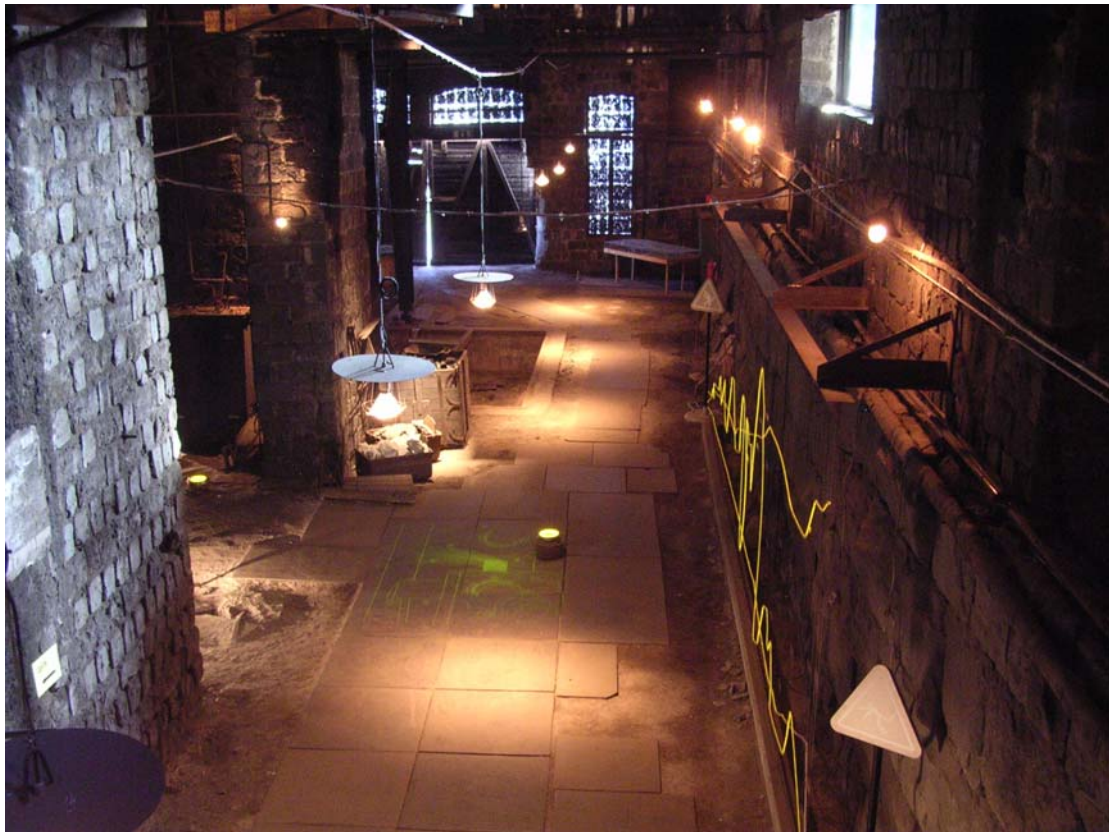
by lighting professionals, it is of course very handy to allow them to work with familiar and standardized equipment – not to mention that fact that you can buy it off the shelf.

Continuing down in fig. 2, we see the sensnet gateway, which handles a couple of sensors and a light. This is an example of a very specific piece of non-standard, custom-built electronics equipment. Here we created custom drivers and types to allow MIEL to handle these special devices.

There is one further difference between figures 1 and 2, and that is the introduction of a new character: the **media author**. MIEL was designed primarily to handle interactive shows where the interaction was very much about physical objects coming to life – not so much about screen based stuff – but even there we had movies, a lot of sound files and some flash animations. MIEL could very well be used in other settings where the focus is much more on new and old moving media. Since it is easy to expand the device architecture, MIEL could also be used as a hub for distributing media content to screens and speakers, and MIEL could be made to connect to existing media players. In such setting the role of the media author(s) would be perhaps be more pronounced, while the interaction might be less complex. Obviously the interaction and media author could be the same actual person.

¹ Simple, when presented like this. In fact, it took us a lot of experimenting to arrive at it, and it really is rather complex. A screen connected to a PC, a push button and a DMX-controlled spotlight where you can control intensity, pan and tilt are three very different kinds of devices, but the system can handle that.

Case: Avesta Steelworks



The Avesta Steelworks is an industrial heritage site. It was used for producing iron and steel until the late 1930s, when the production was moved to another facility. Now it has been turned into an interactive environment open to the public.

Overview

The Avesta Steelworks visitor environment consists largely of two parts, one around the blast furnace hall (where iron was produced) and the Martin oven hall, where iron was turned to steel. Here we focus on the blast furnace hall, which was realized using MIEL technology. As a visitor, you are equipped with a customized flashlight. You use it as a light source in discovering the deliberately dimly lit area, but it also functions as your interaction tool. Apart from sending out light, it also transmits a special infrared signal, somewhat like a remote control. The signals are picked up by **hyperspots**, special sensors that are clearly visible as glowing yellow rings

distributed around the hall. Each flashlight transmits a unique signal, which allows the system to distinguish how each visitor moves. The flashlight works like a mouse – the hyperspot lights up when you “mouse over” and you can trigger it with a button press.

The experience

In the hall, there are a number of interactive devices that can be activated using the flashlights, for example:

The **sound pebbles** are metallic objects on the floor. Each sound pebble contains a speaker and a sensor. When you activate it, it plays back a short sequence from interviews with people who worked in the environment.

The **slag tapping** shows you what it looked like when slag was tapped from the blast furnace. The white-hot slag was poured into cold water, which caused quite

a lot of hissing and steaming. The installation uses 14000 watts of infrared heaters, a smoke machine and a set of speakers in an attempt to recreate the event.

The **inferno experience** takes place *inside* one of the blast furnaces. A clever use of UV lights and fluorescent paint gives you an illusion of glowing coal, and together with heated air, smoke, and the sound of crackling fire this gives you the impression of being inside the “inferno”. If you look up inside the blast furnace, you will find a hyperspot above. Pushing your flashlight button releases a mechanical contraption that looks like a load of burning coal – but is in fact Styrofoam – falling towards you.

The **interactive map** is a flash animation projected on the floor. In one mode, it shows a map of the blast furnace hall, where all the hyperspots are indicated. The ID from your flashlight is stored every time you interact with a hyperspot, and at the map this information is used to replay the path you have taken through the environment, as well as indicating which parts you have not visited yet.

The technology

The Avesta steel works is a large environment. It uses more than 90 channels of light and 24 sound channels, and several kilometers of cables.

SensNet

A custom technology handles all inputs and some outputs. The SensNet system consists of nodes and gateways. The nodes each have an embedded microprocessor, but are mainly used to connect to sensors and actuators. The hyperspots, which contain a sensor and a ring of glowing LEDs, are such nodes. They are connected in a bus, tree or star

topology to a gateway, which collects the information and communicates with the interaction engine using TCP/IP.

DMX

Most of the ambient and effect lighting (as well as smoke machines and heaters) was built using standard DMX equipment. This allowed the light designer to work with his customary light control table, noting the appropriate levels and feeding them into the MIEL scripts. We have successfully tried several kinds of PC-to-DMX interfaces (including DMX-Dongle on a parallel port and two kinds of ArtNet-compatible DMX ethernet gateways) and written device handler drivers for them.

Audio player

There are 20 or so speakers, used for the ambient soundscape and effect sounds, and for the interviews played from the sound pebbles. They all connect to a sound player; a Linux-based computer that plays sound files from the hard drive. The computer listens to commands from a network socket. This allows us to play any sound on any speaker.

Flash interface

The flash map is an ordinary Flash MX animation that opens up a TCP network socket and talks to the MIEL engine. (This technique also works with Macromedia Director). A special device handler driver (in Java) extracts user information and sends the data needed for the map to display the user's path.

Power handler

The video projectors (one for the flash animation and another that is just a looping movie) can be controlled using a serial cable. We wrote a tiny application that opens a network socket and listens for two kinds of signals to turn the projectors on or off.

Case: China Before China



Photo: Karl Zetterström

China Before China is a permanent exhibition in the Stockholm museum of Far Eastern Antiquities (Östasiatiska Museet). The museum holds a truly unique collection of ancient (5000-3000 BC) Chinese pottery. The exhibition, which covers one room in the museum, deals with these pots.

Overview

The visitors walk freely in the room, there are no special interfaces to be carried, worn or attached. Motion sensors respond to movements in the room, while other sounds and lights are triggered by explicitly touching indicated points on display cases or opening drawers.

The experience

The **hill of pots** is a large display case that covers one short end of the room. Inside the case, a number of the ancient pots are on display. The case is divided in three sections, each of which contains a number of pots, a sensor and a speaker. The sensor is mounted on the inside of the glass, and is indicated by the shape of a hand where you are invited to place your own hands. If you do, a light will turn on pointing out one of the pots, and a

recorded clip of two children discussing the pots will play. Each section contains several sequences of animated lights and sounds.

The **night** is a room inside the room that you can enter. Inside there is a 3D sound piece and a video projection. The ceiling contains a fiber optic display that looks like stars, and the room is intended to feel more like “at night in ancient china” rather than a room in contemporary Stockholm. Motion detectors sense when there is somebody in the room and change the ambience.

The **Chinese river** is a large video projection that covers one wall in the room. It uses eight video projectors that project onto a print of a photograph of a river in China. It is not interactive, but the interaction engine controls the projectors, so that they show one continuous picture across the entire wall.

The **sound drawers**: a display case lining one wall is filled with various objects. Also there are a number of drawers underneath which you are encouraged to open. If you do, a speaker behind the drawer will play a sound, giving you the impression that the

sound “is in the drawer”. If you close the drawer the sound will stop, and if you open it again another (but thematically similar) sound will play this time.

The technology

PLC

A programmable logic circuit (PLC) reads all the sensors – the ones in the drawers, in the pot hill, and the motion detectors – and sends information to the interaction engine. The PLC is an off-the-shelf solution that we incorporated in the system.

Audio player

The audio system used in *China Before China* is the same as in the Avesta Steelworks.

WatchOut

The Chinese River uses a piece of software called WatchOut, which allows us to treat the wall and the array of projectors as one large (extremely wide) screen. The person editing the video made it as one long video with labeled key frames. The WatchOut programs running on four computers opens network sockets and listens for commands that tell it to jump to a certain key frame. This allows the interaction engine to control the video wall and synchronize what happens on the wall with the rest of the exhibition. One example of this is how there is a “rain and thunderstorm” sequence. It plays on the large wall, inside the night and on the ambient soundscape, to create a coherent ambience all through the exhibition.

DMX

Like the Avesta case (but to an even higher degree), *China Before China* relies on DMX controlled lighting.

More about MIEL

The XML document format

MIEL is an XML standard, defined in an XML schema. There are also other types of XML documents used to declare device types and device handler types. There are also XML schemas defining the document format used to write these types of documents.

MIEL is a distant relative of SMIL (pronounced “smile”), a standard used to define interaction in screen-based multimedia productions. (See <http://www.w3.org/AudioVideo/>), but MIEL is geared towards handling multiple users and interactive environments rather than single users and screen-based multimedia – MIEL is the Multiple User Interactive Environment Language.

MIEL is not really a programming language, even though it is, in a sense, used to program the environments. MIEL is more of a markup language that basically treats interaction as a type of data. This does not, as far as we can tell, imply any sort of restriction on the type of interactivity you can describe using MIEL.

The MIEL scripts rely on describing the interaction using three kinds of time containers: sequences (SEQ), parallel (PAR) and exclusive (EXCL) nodes. This is the most important relationship to SMIL, which also relies on these three time containers. Inside the containers, which can be nested arbitrarily deep, you place other kinds of elements, such as AUDIO or VIDEO elements. So, to play two sounds after each other, the code looks something

```
<seq>
  <audio device="speaker1" src="foo.wav" />
  <audio device="speaker1" src="bar.wav" />
</seq>
```

Figure 3. A piece of simple MIEL script to play two sounds in sequence.

like the XML code in figure 3.

There are quite a few other features that are necessary to allow you to write more complex behaviors. The most important is the BEGIN element (which goes inside any other element) and waits for an event and/or checks a Boolean condition before it allows its parent element to start.

Another handy element is the SCRIPT element that allows you to embed JavaScript code in the MIEL scripts. It is even possible to talk directly to the Java classes, for example the MIEL logger or even directly to the device handler drivers.

The extensible device architecture

The main traits of the MIEL device architecture were covered in the introduction. To recap, the device architecture builds on the notion of **devices** attached to **device handlers**. The interaction author should not need to know a lot about the specific protocols that are used between the interaction engine and the device handlers and/or devices – to the author, it appears as though there are a number of devices that magically talk to the interaction engine.

In the interaction engine, devices and device handlers are instantiated using a combination of a **driver** (a piece of java code) and a **type** (which is defined in an XML document). When it is possible to use an existing driver, you can customize your device using only the XML device type document.

For example, you could define a specific

kind of DMX spotlight that uses two channels of DMX, where the first is dimming and the second is rotation, measured in degrees, from 0 to 180. To do this, you would not need to write any Java code, just copy an existing, similar device, and change a few lines in an XML document.

Writing custom device handler drivers, in Java, is also a relatively simple feat. The drivers we have written so far range from about a hundred to a few hundred lines of (well-documented) Java code.

The example in fig. 4 shows a video player setup. This kind of video player would be a piece of software running on a separate PC with a computer screen. The video player software is controlled using a custom network protocol, which sends “start” and “stop” commands along with the name or URL of the file to play.

Here the device handler for the video player has its own type (“ImagoPlayer”) and driver (“ImagoPlayerHandler”), but the screen is a standard component. It uses a Screen type, which really is only used to give it a special icon in the editor, and the default “Device” driver, which is a dummy driver that does not really do anything.

What figure 4 tries to convey is that this setup consists of four parts. There is a device handler and a device, and they each both have a type and a driver.

The types are defined in XML documents, and the drivers in Java code.

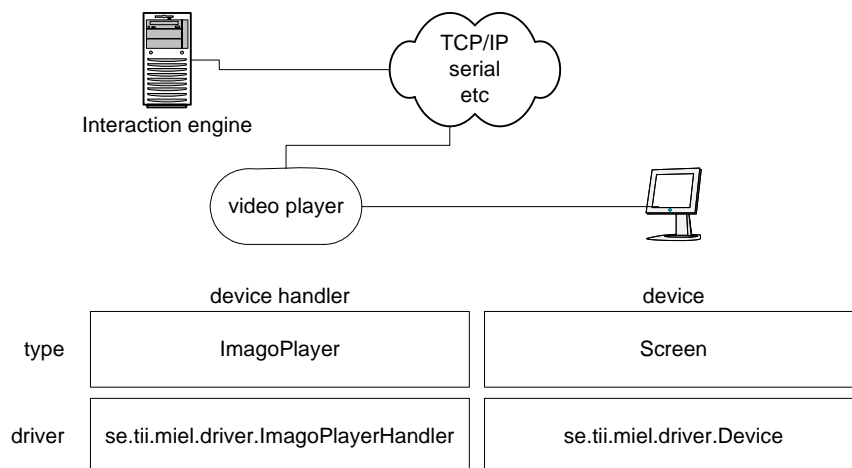


Figure 4. Devices and handlers are created by combining drivers and types.

The MIEL Graphical Editor

The MIEL editor is a standalone Java application. It generates the XML document that contains the script that you run in the interaction engine.

Defining the site

In the editor you typically start by placing a map of the physical space you are working in as a backdrop in the editor. Then you just drag-and-drop the devices you have onto the map, and you can set the different settings you need to enter to actually connect these devices to the correct device handlers etc.

Editing the interaction

Once you have a site defined, you can start editing the actual interaction. In the editor there are different views for things you can do. The tree view (the “Explorer” view in Fig. 5) is a direct mapping to the XML document and reflects the way the interaction is described in MIEL. The Timeline view opens to represent selected portions of that tree as a timeline, perhaps more familiar to users accustomed to working in video editing tools, Flash, etc.

There are also views to do things similar to key frame animations of property values, e.g. to dim a light, where you have time on a horizontal axis and values on the vertical axis.

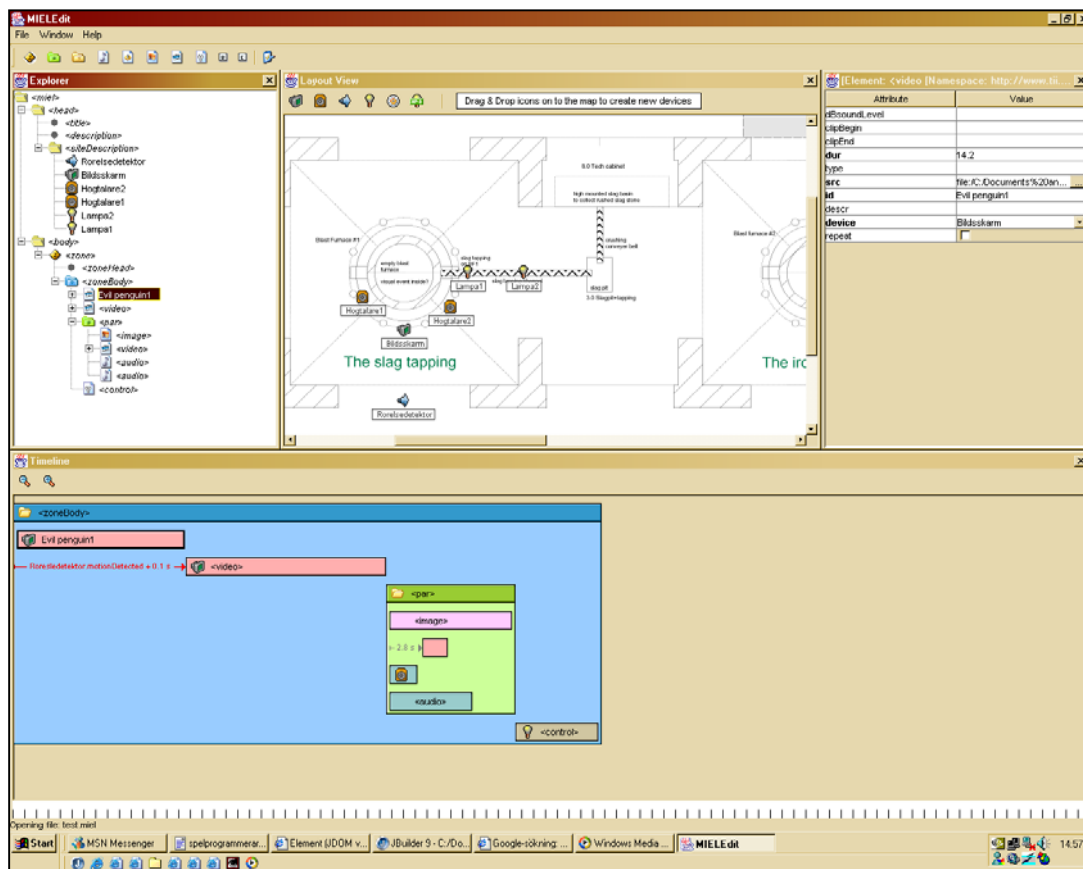


Figure 5. The MIEL Editor (screenshot)

Designing an interactive site with MIEL

Planning the system

Planning a site is basically doing what interaction designers do. You need to decide what the user(s) should experience, what they should be able to do and how the system should respond. In parallel with the "pure" interaction, you also need to decide on what hardware and software to use. Depending on your project, this may include anything from hooking up a PC to laying kilometers of cables as well as designing and constructing elaborate electromechanical gadgets.

Setting up the infrastructure

Some things you need to do are

- Installing the MIEL engine on a Java-enabled computer
- Setting up hardware, connecting it the computer using TCP/IP, serial, parallel, USB or possibly other kinds of communications
- Installing the appropriate device handlers, device handler types and device types

Installing the MIEL engine as such is no major feat. It requires a computer capable of running Java. We have used both Windows and Linux computers, and Macintosh should be possible as well. MIEL depends on a number of open-source Java libraries, and our own codebase. This can all be distributed in the form of a number of Java Jar files.

Setting up that rest of the hardware can be anything from really simple to extremely difficult. It should also be possible to retrofit MIEL to be the "brain" in an existing system. It is of course easier to use MIEL with hardware that has some sort of open

protocol, but this is true of any system trying to do what MIEL does.

Extending the system

Things you can do include

- Customizing a device by writing custom **device type** documents in XML
- Customizing device handlers by writing custom **device handler type** documents in XML
- Writing custom **device handler drivers** in Java
- Writing custom **device drivers** in Java

Our hope is to allow MIEL users to build up a library of reusable drivers and types. As more and more types of devices and device handlers are being added to MIEL, it will become easier to simply copy and paste existing solutions. We already have all the Java and XML definitions needed to run a number of applications, such as DMX lighting, a sound player capable of handling a large number of audio channels, a media player capable of playing most types of digital media from a computer, Flash animations, etc.

Making the scripts

The best way to make the script is to use the editor². However, it is perfectly possible to edit them (or make them from scratch) using any kind of XML or text editor.

As was noted in the "scripting in MIEL" section above, MIEL is not really a programming language – or, alternatively, it uses a programming paradigm that is not procedural, functional or object-oriented, at least not in the conventional sense.

² It is, in theory. Note however that the graphical editor needs some more work, as we have prioritized working on the interaction engine rather than the editor, as the editor is less critical for the operation of MIEL. You can always make the scripts in another XML or text editor.

We would like to point out that MIEL is not a miracle solution. Designing the interaction, at least for more advanced scenarios, *is* inherently a programming task. A proper tool can make it easier, and hide the details of the coding – allowing you to do it graphically. In a sense, this is still a kind of programming; it is just a lot easier to do. Point being: yes, MIEL can make it feasible for non-programmers to design interaction, but we expect the most creative uses of MIEL to come from interaction authors with a modicum of programming experience.

What MIEL has done

So far, MIEL has been used for two large museum installations. These prove that MIEL is a dependable solution that is capable of handling that type of interaction. Some of the things we do in these exhibitions include:

Responding to simple **sensor readings** of touch sensors or infrared presence detectors

Picking up “tagged” sensor readings that carry a **user identification** into the system, and handling that user information to process information for each user, while allowing a large number of simultaneous users. In the Avesta case this information is used to store the path the visitor has taken through the exhibition, and outputting this on a map along with an indication of what parts are left to explore.

Playing **audio and video files**, in any combination of sequences and parallel modes. The device architecture makes it simple to play a given media file on a given device (provided that you, like us, have the type of audio/video player that supports this, of course).

Controlling lights, including handling a large number of simultaneous animated (dimmed) lights, following pre-

programmed sequences (“light cues”), random animations or levels computed on the fly.

Controlling **smoke machines, heaters, motorized equipment** etc – anything that can be powered on or off or dimmed using e.g. DMX

Controlling **custom-built solutions**, such as the drop mechanism in the Avesta steelworks case. Here we asked the company doing the electric winch installation what was easy for them, and we agreed on using a simple relay that closes or opens on an impulse from MIEL.

Talking to other software systems using any kind of interface (tcp/ip, serial, etc).

As has been noted repeatedly it is easy to extend this list with new types of devices.

What MIEL could do

Neither of the two cases is particularly good at actually exploring the full capabilities of MIEL. In particular the **multi-user handling** contains some features that are not used at all, such as “locking” parts of the exhibition to a single user to handle conflicts with other visitors trying to use the same pieces.

The type of user identification used in the Avesta case, where you know who the user is (or, more correctly, you know the identity of the flashlight the user is using), could be used to do much more complex interaction. Some obvious ideas are allowing the user to set the **preferred language**, or using different interaction for different **categories of users** (such as children, tour guides, or the average adult visitor).

Future usage scenarios

The two cases described in this document are fairly similar: interactive visitor environments where the interaction is

used to encourage users to explore some sort of heritage (a museum exhibit containing ancient Chinese artefacts, an actual industrial site opened to visitors). Of course, MIEL could be used in other, similar settings, but we could also consider some other potential usage scenarios:

MIEL could be used to drive **interactive art** pieces.

MIEL could be used as a control system, for example for **theatre or dance performances**. One could make a custom interface where somebody off stage controls sound and lights, either directly or by triggering pre-recorded sequences (the latter being the model that most naturally fits with the way stage productions normally work). It would also be possible to have things triggered by people on stage, using some sort of sensing equipment, or to have some parts triggered on stage (e.g. a sound effect when an actor punches a doll, or lights that follow an actor) and some parts (ambient lighting, say) controlled off stage.

MIEL could be used to create **interactive storytelling** applications, where the interaction is much more complex than in the use cases described in this document. There seems to be a current trend towards **pervasive gaming** – interactive games that take place “in the real world” rather than only on-screen. It would also be possible to use MIEL as a control system that handles sensors, lights, media etc, which connects to a backend that handles interaction that is even more complex.

MIEL could be used to build other kinds of rides and spectacles of the type associated with **theme parks and fun fairs**.

MIEL could easily be equipped with a web or other **online component**, e.g. to let

users online control physical installations, perhaps while watching it with a webcam.

MIEL could be used to a control **commercial or information displays** in e.g. a mall or fair. As a simplest case, MIEL could simply be used to “push media” to screens and speakers, at the other extreme this could be done fully interactively, e.g. using RFID technology to access information tied to particular objects in a store.

How to get MIEL

Currently (June 2007), MIEL is an in-house application owned and controlled by the Interactive Institute.

MIEL was developed by Peter Lundén, Fredrik Bridell, Olof Bendt and William Sporrang at the Interactive Institute.

It is not released as open source, but it is built using only our own code and LGPL-licensed open source libraries.

If you are interested in using MIEL, please contact the authors of this document directly.

fredrik.bridell@tii.se
peter.lunden@tii.se