User interfaces for intelligent household remote controls

Miroslav Macík and Václav Slováček
{macikm1|slovav1}[at]fel.cvut.cz
Department of Computer Graphics and Interaction
Faculty of Electrical Engineering
Czech Technical University in Prague

Abstract

In the last few years the number of home appliances in our environment has dramatically increased. Each device has its own user interface and a corresponding way to communicate with its users. These user interfaces usually differ for every single device. This can be confusing for users, especially aged people and disabled people. It would be optimal, if we could provide a consistent user interface for all applications. Such interface shall respect specific needs of every particular user.

We focus on developing a method for designing and developing user interfaces that would enable us to deploy single interface on any platform. For this purpose we have developed a XML-based protocol called UIProtocol that enables us to separate application logic on the server side and the user interface on the client side. This protocol also enables us to update the user interface and to notify the server about events invoked by user.

Based on the technology we developed for delivering user interfaces, we have built a server layer capable of automatic construction of user interfaces for controlling the currently available set of devices. The user interfaces are built to match the restrictions (screen, controls etc.) of target controller while keeping the effort to control the user interface at minimum.

Keywords: User Interfaces, Description Language, User Interface Generation

1 Introduction

Our work addresses the problem of developing an intelligent household with special focus on elderly and/or disabled people. This work is part of a larger project called i2home. The motivation of i2home project is to address the problem of complexity in a modern household. As mentioned, there are many devices made by many manufacturers and every such device usually has its own user interface that is usually not consistent with user interfaces of other devices in the household developed by other manufacturers. A user can be confused by such situation.

i2home aims to solve this problem by using a universal control device that would be able to remotely control most of devices in such household. Optimally the user interface of the controlling device should be designed directly for purposes of the current user and for current set of devices in the household. It is obvious that this problem could not be completely covered by human developers because of a vast amount of possible combinations. In addition not all information is available during the development process - set of devices and user preferences could vary over time.

This is the motivation for development of the UIGenerator. It should solve in runtime problems that human developers are not able to solve.

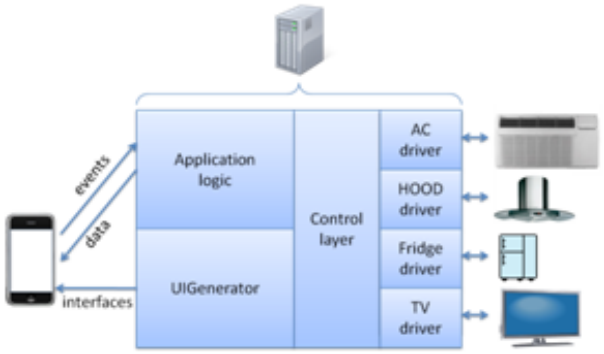

Figure 1: System architecture

2 Related work

Several efforts have been made to develop a language for describing cross-platform user interfaces [6][7]. Although most of the languages try to describe complex user interfaces, they mostly fail in separating the model and the controller from the view, which is essential for simple generating of user interfaces.

There have been some promising approaches introduced in the field of automatic user interface generation, such as the SUPPLE [1], which provides an inspiration to for our work. In this work we adapt the user interface generation into an environment of an intelligent household.

3 Our method

We have addressed the problem of providing user interfaces for home appliances by splitting it into two parts. We have solved the user interface delivery, rendering and client-server communication by developing UIProtocol. Then we have developed UIGenerator that relies on MVC design [4] of UIProtocol.

UIGenerator provides user interfaces (view) without having to have access to data (model) that are provided/updated by application logic (controller).

Whenever dynamic data are updated they are pushed directly to client without the interface having to be reconstructed on the server side. UIProtocol client handles binding of data to corresponding user interface elements automatically. Whenever the user initiates an event, the event is sent to server. UIProtocol server then locates appropriate handler for the event and executes it.

3.1 Protocol for user interfaces and communication

UIProtocol combines user interface description language and a language for client-server communication in a single type of XML files. This protocol was developed at the Czech Technical University in Prague and was originally designed for purposes of the i2home. It is based on Model-View-Controller design pattern [4] which brings many advantages. The key advantage is separation of application logic and presentation. Thanks to client-server architecture one application logic can be used for multiple clients. The original motivation was to create only one interface that can be later delivered on multiple platforms. Additionally a need to manage the client-server communication has been addressed. UIProtocol is designed to support rich clients with animations, media and styles. It also defines behavior (so-called graceful feedback) that enables simple client to render complex components that are not directly supported by rendering a hierarchical structure of basic user interface components. Key features of UIProtocol are:

- XML syntax
- MVC design – clear separation of presentation, model and application logic [4]
- internationalization support
- direct support for data binding
- application logic is programming language independent
- simple implementation of basic renderer
- implementing of the whole protocol is not necessary for simple renderer
- extensible without modifying the protocol specification
- precise visual definition of elements if necessary
- layouts support
- animation support

All these features are important for the UIGenerator and some of them have been designed with UIGenerator in mind. Firstly, the XML syntax makes the management of final user interfaces simpler because there are many tools available for processing XML documents. Secondly, the problem of generating user interfaces would be extremely difficult without the separation of presentation, model and application logic. The data binding feature inherently supported by clients is another important feature that simplifies the design of the UIGenerator. Finally, it is necessary to point out the precise visual definitions of elements, this feature is crucial for the UIGenerator because it allows computation of the estimated layout, positioning and appearance of the final interface. Such optimization would not be possible without this feature.

3.2 UIProtocol communication

The sequence diagram in Figure 2 shows an example of communication between the UIProtocol client and server. UIProtocol client is allowed to send only events to the server. In the opposite direction, the server sends the models (data) and the user interfaces. At the beginning, the client notifies the server about the connection and sends its description (screen resolution, supported widgets etc.) (1.). The server answers with a model containing its own description (http server port etc.) (2.). In the next step the client asks the server for the public.application model, which contains, name of the master interface (interface displayed on root of the application) (3.). The server answers with the public.application model (4.). As soon as the client knows the name of the master interface, it asks the server for its description (5.). The server sends the Master interface back to the client (6.). The Master interface contains elements that are bound to a model called Master data. The client automatically asks the server for this model (7.). The Server replies with the Master data model (8.). After some time the user invokes an action in the user interface rendered on the client, for example he presses "+" button of the heating control. An event with id temperature.up.pressed is being automatically sent to the server (9.) The application logic on the server handles all the work that, in this case, causes the change of temperature element of Master data model. Update of this model is automatically sent back to the client (10.) and by using the data biding propagated to the user interface.

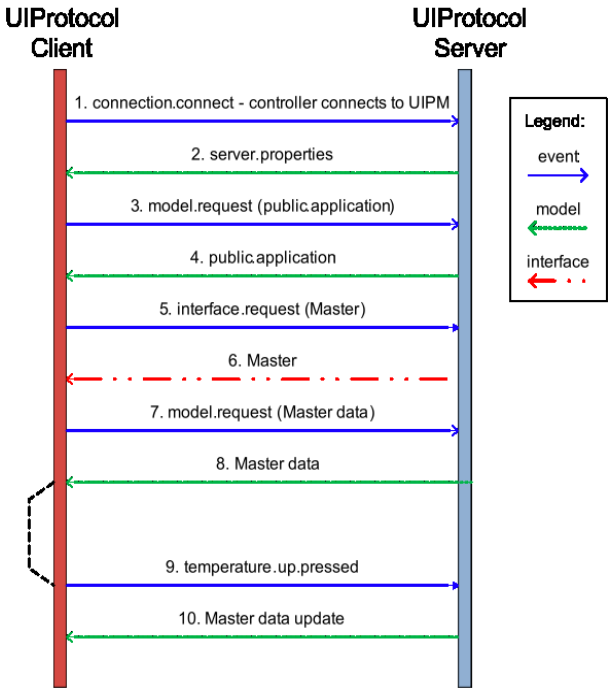

Figure 2: Sequence diagram of example UIProtocol communication

3.3 UIProtocol user interfaces

In Figure 3 is a sample user interface in UIProtocol. The description in UIProtocol (see next page) is not very space-saving but on the other hand UIProtocol provides extensibility without modification of schema file and implementation of a XML parser is easy.

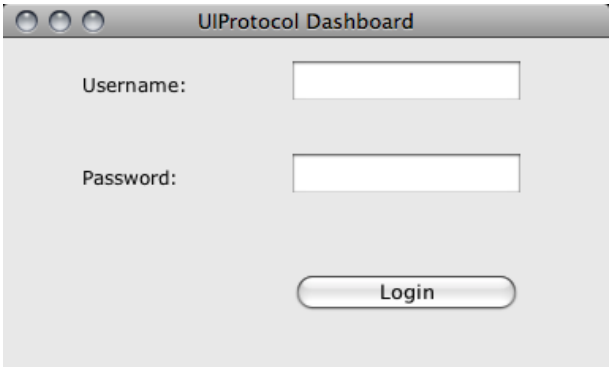

Figure 3: Example of UIProtocol UI

3.4 Data Binding

Very important feature of UIProtocol is support of binding. Binding enables to connect any property (position, style, content of text component) of any element in user interface to specified data in model.
Binding separates dynamic data from the user interface structure. This enables the application to alter the user interface without knowing its structure and it is important for making the application logic independent on the user interface generation process.

Changes of values in the model are automatically propagated to all associated user interface elements. Application itself does not have to (and cannot) handle this process on the client side.
Data binding also provides a simple way to animate user interface elements by specifying an interpolation that is then used to change an original value to an updated value in a specified amount of time.

3.5 Event based communication

Event based communication is a second aspect of separation of the user interface structure and the application logic. The client sends to the server a UIProtocol document describing an event whenever the user performs an action that was declared to trigger an event.
By design UIProtocol enables application logic only on the server side. This is essential for delivering same interfaces with the same behavior on different platforms.
Although this brings some disadvantages such as latency when responding to user feedback, it is essential for proper functionality across platforms. UIProtocol provides features to eliminate these disadvantages, however description of these features is out of the scope of this paper.
Both event based communication and model binding enable developing application logic without knowing anything about user interface. Only the list of events and important (bound) values have to be passed to user interface generator to connect the interface and application logic.

```
<?xml version="1.0" encoding="UTF-8"?>
<UIProtocol xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                        xsi:noNamespaceSchemaLocation="http://solari.cz/resources/xsd/uiprotocol/1.0"
                        version="1.0">
  <interfaces>
    <interface class="MyInterface">
      <container>
        <element class="public.text">
          <position>
            <properties names="x,y" values="50,35"/>
          </position>
          <property name="text" value="Username:"/>
        </element>
        <element class="public.input">
          <position>
            <properties names="x,y,width,height" values="180,15,150,25"/>
          </position>
        </element>
        <element class="public.text">
          <position>
            <properties names="x,y" values="50,95"/>
          </position>
          <property name="text" value="Password:"/>
        </element>
        <element class="public.input">
          <position>
            <properties names="x,y,width,height" values="180,75,150,25"/>
          </position>
        </element>
        <element class="public.button">
          <position>
            <properties names="x,y,width,height" values="180,150,150,30"/>
          </position>
          <property name="text" value="Login"/>
        </element>
      </container>
    </interface>
  </interfaces>
</UIProtocol>
```

Code 1: Example of UIProtocol UI description

3.6 User groups with special needs

Some parameters of human capabilities are involved in the usability of particular user interface. The user interface enables a user to interact with an application but a badly designed user interface needs much more effort to deal with. The parameters of the evaluation function should also correspond to the estimated effort needed for the successful interaction with the user interface. The better the user interface is the lower should be the value of this function. In the following text we call this function "estimated user effort function".

This chapter summarizes requirements on function formulating the estimated user effort. The following parameters are crucial for a successful user interface and the function should them take into account.

- **Load of working (short-term) memory:** The user interface should not contain complex structures. Such structures require memorizing of the current position or other parameters. The information required for a particular task should be accessible at the same time and place.
- **Usage of low capacity channel between sensory and working memory:** The user interface should be well organized and corresponding elements should be visually grouped.
- **Cognitive complexity:** User interfaces should not be complex. The user interface should contain only elements needed for satisfaction of required goals. New user interfaces should be maximally consistent with the existing ones.
- **Visual appearance:** Size of elements should satisfy the needs of users with low vision capabilities and respect a device where the user interface is rendered on.

The aim is to create a function, which expresses the estimated user effort needed to manage a user interface. Let us have such a function and a possibility to make changes in the user interface. Then we can use well-known optimization techniques to find an optimal user interface with the minimal value of this function.

3.7 Automatically generating user interfaces

Thought the UIProtocol is very capable itself it is not able to address all problems in the intelligent household. For example the context awareness could not be involved into the static user interfaces. In addition, there are different needs and capabilities of particular users. It is very hard to design a user interface tuned directly to cover needs of particular user in a general way because each user is different. Last but not least because of an intelligent household is a complex system, developers of particular components usually do not know the whole context of usage.

Addressing these problems was the original motivation for implementing the UIGenerator. We analyzed state of the art solutions in this area and we have found SUPPLE [1] being very promising approach. We adopted the basic idea to minimize the estimated user effort necessary to deal with the user interface. We called our solution UIGenerator and integrated it into the current i2home system and the UIProtocol become the output language.

3.7.1 Rendering pipeline

The generating process of user interfaces is divided into multiple separate steps. This separation makes the design clear and extensible. Figure 4 illustrates the layout of the rendering pipeline.

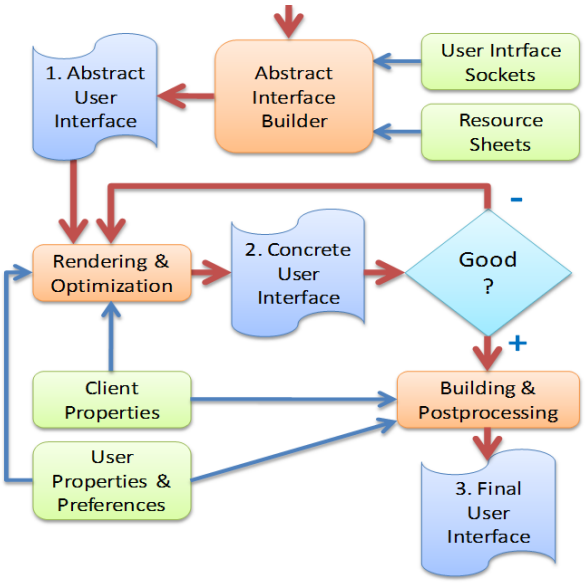

Figure 4: Rendering process

1. The first step is the construction of an abstract user interface by the Abstract interface builder. An abstract user interface is a hierarchical structure of so called abstract user interface elements. The abstract user interface is an input for the following step in the rendering process. Currently the abstract user interfaces are based on the description of connected appliances to be controlled.
2. The second step is the rendering of concrete user interface and its optimization. The input to this process is an abstract user interface, properties of the client (screen resolution, supported user interface elements etc.) and properties of the user (quality of vision, cognitive capabilities etc.) and his preferences (desired language, usage patterns etc.). The purpose of these parameters is the computation of a function representing the estimated user effort to deal with the final user interface. The estimated user effort is also the parameter which is minimized during the optimization process.

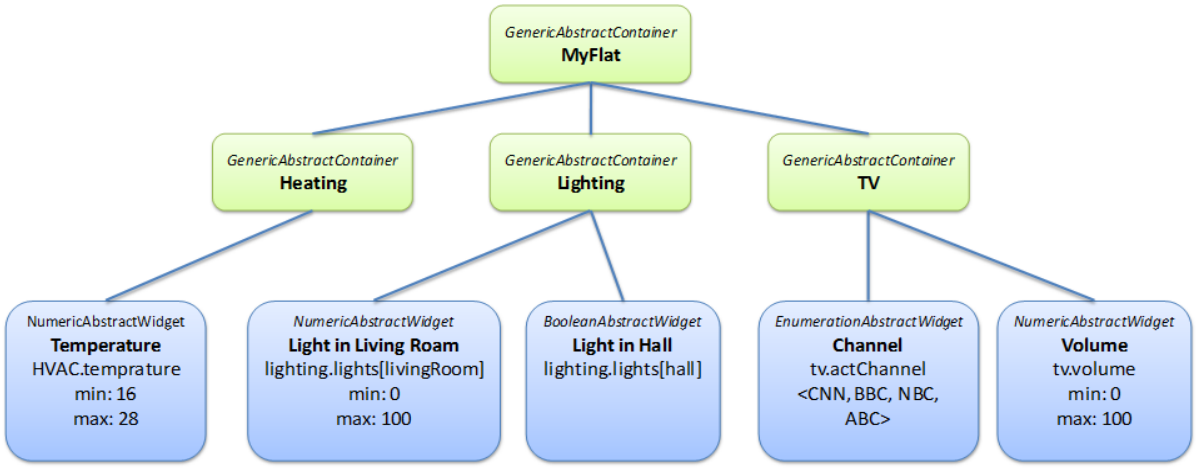

Figure 5: An example of an abstract user interface

3. The third stage is building of a user interface in UIProtocol and post-processing. The building is a transformation from the object representation to the XML representation of UIProtocol. During the post-processing further adjustment is performed, but the structure of the user interface is not changed. For example aligning of elements and containers or applying of styles takes place here.

At this point the architecture of the UIGenerator and its rendering process has been introduced. In following text individual components of UIGenerator and their functionality are described in detail.

3.7.2 Abstract interface builder

There are two types of these abstract elements:

- **Abstract Containers:** Elements that can contain other abstract elements as children.
- **Abstract Widgets:** Elements that refer to particular property of a connected appliance to be controlled. Currently there are these abstract widgets: Boolean, DateTime, Enumeration, Numeric, Media, String, and Trigger.

In Figure 5is an example of an abstract user interface based on a hypothetical intelligent household with three appliances: Heating system, Lighting and a TV. The heating provides the adjustment of temperature. The lighting subsystem makes it possible to control light in the living room which has the dimming functionality and a simple light in the hall. Finally for the TV we can switch between channels and adjust the volume. We use this example to show how the UIGenerator works.

3.7.3 Concrete interface builder and optimizer

The building process begins when the root Concrete Interface is initialized with corresponding Abstract Interface. The hierarchical structure is then built recursively.

We defined classes that are object representation of the UIProtocol user interface elements. Each such class can compute its own value of estimated user effort function. This value depends on the user and particular controlling device. The value of particular elements is summarized for the whole user interface and corresponds to the overall estimated user effort. The concrete interface builder also works with Concrete widgets and containers. This is similar to abstract interface builder, but the semantic of these elements is different. There are also these concrete widgets and containers which correspond to the UIProtocol elements:

- **Concrete containers:** Frame, Tabs and Generic container.
- **Concrete widgets:** Button, Checkbox, Choice, Textfield, Audio clip, Image, Video clip, Slider, Label, Textarea.

At the beginning the Concrete interface builder provides a user interface with the minimal value of user effort function. This process is trivial – the widget and container providers provide widgets with minimal value of user effort function. The resulting user interface has also minimal value of user effort function but the parameters of the controlling device are not taken into account. For example this user interface usually does not fit into the screen resolution of the controlling device. This interface is passed forward to the optimizer.

Optimization of the user interface begins with the interface provided by the concrete user interface builder. The first step is removal of widgets that are not supported by the controlling device. This affects selection of widgets that can be used in the user interface. For the minimization of the overall value of the estimated user effort we adopt the simulated evolution because of this well known algorithm is very flexible and produces good solutions for many complex problems. In some publications is this algorithm designated as evolutionary algorithm (EA) [5].

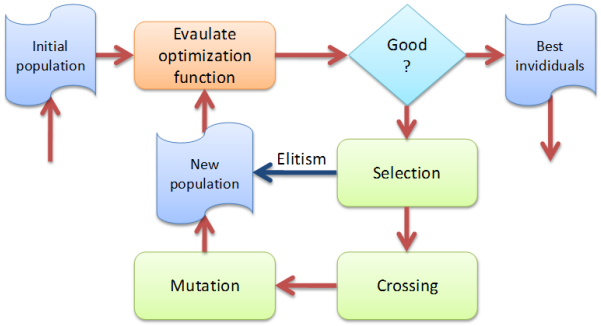

Figure 6: Simulated evolution

In Figure 6 is the diagram of a simulated evolution algorithm. In our case the initial population is the concrete user interface with minimal value of estimated user effort produced by the concrete interface builder.
Each cycle of simulated evolution consists of three basic steps: Selection, Crossing, and Mutation. In the selection step some individuals are randomly taken from the previous generation. In this process is used so-called artificial roulette which provides the probability to be selected corresponding to the fitness (value of optimization function) of the particular individual.

4 Results

Figure 7 shows examples of generated user interfaces. The interfaces enable direct control of several home appliances. In the first example, in which more home appliances are available, the optimizer has to divide user interfaces for particular devices into tabs to be able to fit it onto the screen. The second example is the case where the complete interface fits the screen. Both user interfaces have minimal estimated user effort in our metric.

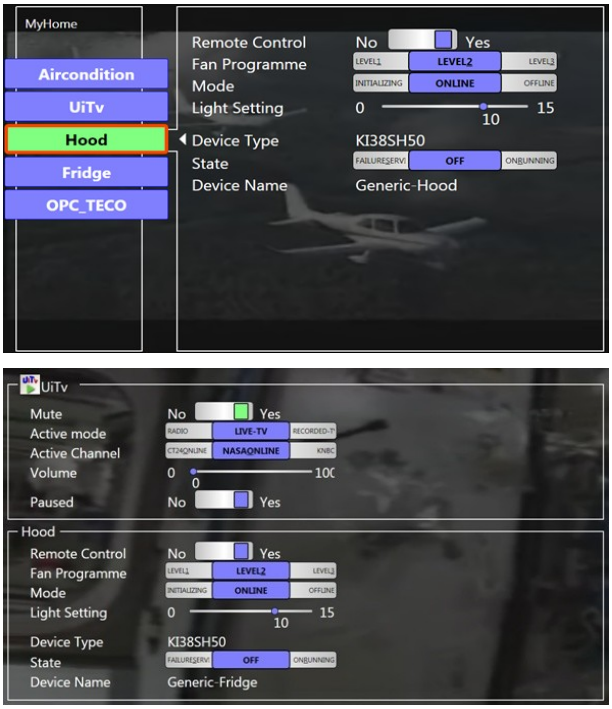

Figure 7: Example of generated user interfaces

Currently the optimization metric of estimated user effort is set for particular elements manually. It is a subject of future work to develop better approach to address this issue.

5 Conclusion and Future work

By developing the UIProtocol and the UIGenerator we have successfully addressed many problems in an intelligent household. Now we can provide one user interface on multiple controlling device platforms. In addition we are able to generate a user interface with minimal estimated user effort. During the work we uncovered new ways how we can extend our approach and make the intelligent household more user friendly.
First of all it is necessary to evaluate our results with target user audience. Results will feed back into our development to enable the final product to fit the user needs.
We would like address the problem of adapting a user interface to a particular user more generally, in the way of involving a module that will test the user capabilities. This module should be in a form of a wizard. Currently the estimated user effort of particular elements is set manually. Result of this test should be a modification of the properties of estimated user effort function to fit the need of the particular user. The Result should also provide information about a preferred color scheme and contrast thresholds for particular user.
A Significant contribution of the UIGenerator can be seen if the context awareness will be involved to the generation process of the user interfaces. This connection brings advanced features like the intelligent selection of the interaction device, automatic invocation of tasks and extended possibilities on how to adapt the user interface directly for the current situation.
The usage of the UIGenerator should not be restricted only to a generation of new user interfaces. The user generator could eventually analyze even an existing user interface. Using the optimization process, it may provide a feedback to the designer of a user interface. This feedback will contain information about what is wrong and the guidelines on how to improve the interface for a particular user or a group of users.

6 Acknowledgements

We would like to thank to all partners in i2home project (www.i2home.org). This work was funded by EU 6th framework program under grant FP6-033502(i2home).

References

[1] Gajos K., Weld D. *SUPPLE: Automatically Generating User Interfaces*, 9th international conference on Intelligent user interfaces, University of Washington, Seattle, 2004

[2] Macík M. – *User interface generator,* Master's thesis, Czech Technical University in Prague, Department of Computer Graphics and Interaction, 2009

[3] Slováček V. – *UIProtocol specification draft 5.* Draft of specification, Czech Technical University, Prague, 2008

[4] Krasner G., Pope S. – *A Description of the Model-View-Controller User, Interface Paradigm in the Smalltalk-80 System*, ParcPlace Systems, Mountain View, CA, 1988

[5] Bäck T. - *Evolutionary Algorithms in Theory and Practice,* Oxford University Press US, 1996, ISBN 0195099710

[6] Souchon N., Vanderdonckt J. – *A Review of XML-compliant User Interface Description Languages*, Lecture notes in computer science, Université catholique de Louvain, Belgium, 2003

[7] Stöttner H. – *A Platform-Independent User Interface Description Language*, technical report 16, Johannes Kepler University Linz, Austria, 2001