

Development Framework for Pervasive Computing Applications

Václav Slováček, Miroslav Macík, Martin Klíma

Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Computer Graphics and Interaction

slovavac@fel.cvut.cz, macikmir@fel.cvut.cz, xklima@fel.cvut.cz

Introduction

Pervasive computing [1] brings the technology closer to the users by enabling the users to use daily-life devices (mobile phones, TVs, touch screen walls, etc.) for controlling their environment and accessing information virtually anywhere. Interacting with such devices does not remind users of classical computers and enables them to more naturally interact with the controlled system, if user interface is designed properly. These devices usually operate in networked environments with every controlling and controlled device connected to a central hub. Bringing easy-to-use applications to such environments faces the challenge of highly heterogeneous, dynamically changing environment and necessity to deploy applications to controlling devices with very different features (display size, input methods, operating systems, etc.).

The *i2home* project is focused on developing smart home applications that would enable various non-technical target groups including elderly people, people with Alzheimer disease, blind people, etc., to take advantage of modern technologies. These technologies would enable the users to stay in touch with their family and be connected to other people while being able to be more independent in daily life. *i2home* tries to achieve this by better integration of home appliances and by providing better easier-to-use controls for the target users.

Recent development in the area of mobile and home entertainment devices led to increased usage of different devices (mobile phones, PDAs, TVs, etc.) for performing various tasks and managing daily life. Such devices may also be used in a smart home for controlling home appliances from any location at home or outdoors. Developing the applications for such environment gets increasingly difficult when different properties of the controlling devices, user preferences/abilities, and different set of controlled appliances are considered. Such constraints force the application developers to either focus on a small subset of available controlling devices and/or to develop multiple interfaces for different platforms that may lead to poor user experience due to inconsistencies across multiple similar products.

Because of the reasons mentioned above, a new development framework was designed for the *i2home* project. This framework is based on the UIProtocol that enables to define interfaces using XML and also defines the standard client-server communication protocol that is very important for pervasive applications in which a server has to handle various client entities that are based on different technologies. In the same time the framework abstracts from details of asynchronous communication between the client and the server. That enables both application logic developers and user interface designers to focus on their primary goals of developing usable and accessible user interfaces and on developing and fine-tuning application logic.

Problem statement

In current environments, the development of the client-side and server-side part of an application is tightly connected with each other and the application logic developer has to communicate often with the user interface designer and vice versa. Often the application logic developers are forced to co-develop user interfaces and user interface designers are forced to implement simple client-side application logic (e.g. in JavaScript). Such development leads to less maintainable code and to a lot of portability issues when changing the client platform. Having server-side application logic simplifies the development and enables to dispatch events, that are invoked by a user's interaction, in a single point on the server side. That enables an easier monitoring processes, debugging the application, and in the future also adding application-wide aspect oriented features [2] that would enable to perform corner-cutting application logic.

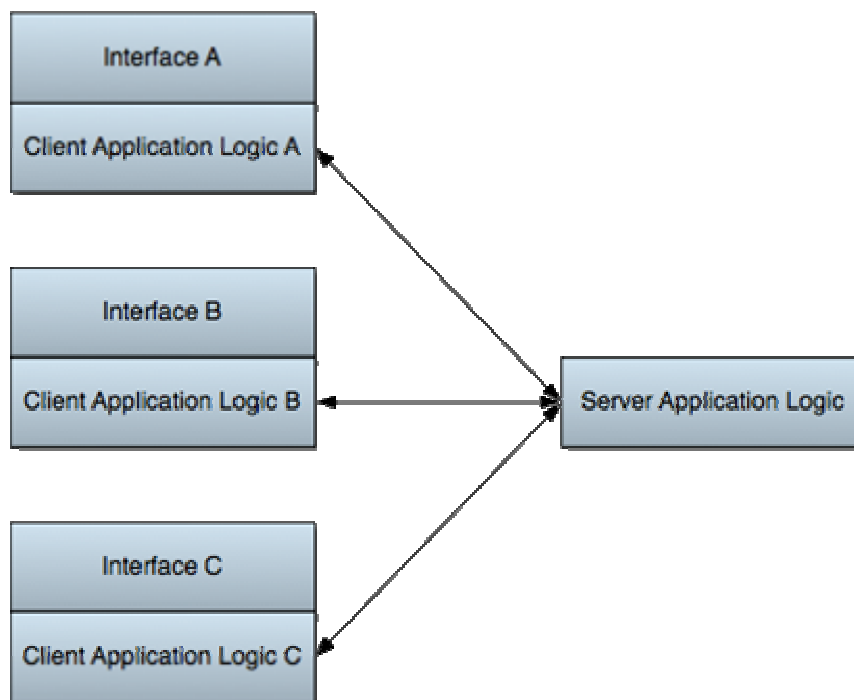


Figure 1: Traditional Client-Server Architecture

The Figure 1 shows how a traditional architecture in pervasive computing environment looks like with each client having different user interface and different client-side application logic. Application logic on client handles communication with the server side and eventually performs some client-side operations as it is closer to user interface. Having a scripting language on the client side is very helpful for creating rich applications with instant feedback to user's actions as there is no latency caused by the network communication. While this is a much-desired feature, it brings a lot of issues related to the efficient development of maintainable applications.

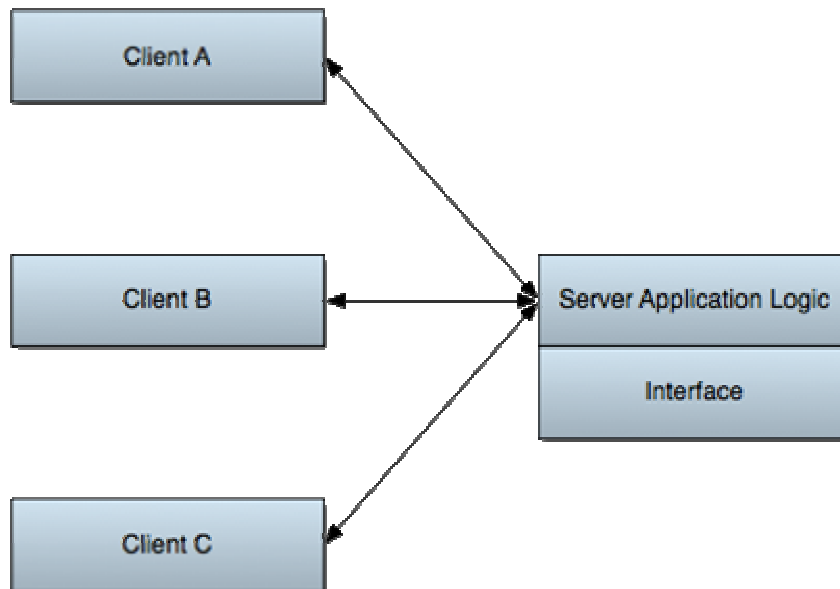


Figure 2: UIProtocol Client-Server Architecture

The Figure 2 shows the way how we intended to minimize the impact of switching platforms on the user interface design and the application logic development: No application logic is implemented on the client side. Even the user interface should be pushed from server in order to simplify deployment similar to most existing web applications. All clients in such framework are thin clients from the developer's point of view. Because this approach requires implementing complex clients (similar to web browsers) it is necessary to ensure that any chosen solution will minimize the complexity of the clients.

In the *i2home* project we followed a User Centered Design. In the iterative design we needed to create a number of prototypes that were immediately verified by usability testing. It is important to rapidly evaluate prototypes on different platforms without a need to rewrite any application logic or a user interface. This enables to evaluate the user interface on multiple devices and choose the best device for users after usability testing is performed. It is also important to be able to create high-fidelity prototypes that may later become fully functional applications without having to re-implement already implemented behavior.

Existing solutions

To date, several various platforms for developing rich internet applications (RIA) are available including but not limited to Adobe Flash, HTML+JavaScript+XML(AJAX), JavaFX and Microsoft Silverlight. The decision to develop UIProtocol framework was based on a study of several existing solutions already available. Adobe Flash was used for developing first prototypes in the *i2home* project. While designing user interface was very fast, creating high fidelity prototypes that would become final applications and would communicate with server side had proven to be difficult with growing complexity of the application.

Apart from the high-profile technologies mentioned above, several other languages has been examined [3, 4] to evaluate whether they are suitable for application development in *i2home* environment.

Adobe Flash

Adobe Flash is the most widely used platform for developing rich internet applications. Adobe Flash is exceptionally powerful tool for creating web vector graphics, animations and delivering media. Adobe Flash provides very good tools for user interface designers, but provides a very poor support for efficient development of the complex application logic. The application logic is often spread thorough the whole user interface and developers have to cooperate very closely in order to complete an application. Debugging the Adobe Flash code is very difficult due to the fact that the application logic is usually spread throughout the debugged application.

Adobe provides Flash development environment and Eclipse-based Flex Builder for developing applications that are more complex. Problematic development of complex applications led several third parties to introduce alternative tools for creating Flash-based applications. These tools however lack some features such as a visual editor.

JavaFX

JavaFX is relatively new solution for rapid prototyping and application development. Stable version of JavaFX 1.2 is available for Mac OS X, Windows and Windows Mobile. Beta version is available for Linux and OpenSolaris. JavaFX enables to built RIA that can utilize very rich Java API. JavaFX behaves similarly to a Java applet or an Adobe Flash movie and may be additionally dragged out of the browser and behave as a standalone application.

JavaFX provides robust support for data binding that proves very useful when dealing with the user interfaces. The major disadvantage is the fact that the application logic is still on the client side [5] and implementation of JavaFX client is relatively complex as whole Java Virtual Machine must be available and support for non-XML JavaFX syntax has to be implemented.

Several development tools are available for JavaFX enabling to import Adobe Photoshop and Adobe Illustrator files and add behavior to the static content designed in these applications.

HTML and JavaScript

HTML is an open standard that can be rendered on almost any platform including mobile devices. The look and feel of any HTML content may be customized by CSS, while dynamic behavior may be added by JavaScript. Communication with server is done by AJAX [6] that is used to exchange the XML (or other files) in the background dynamically updating only portions of a website bringing HTML close to other RIA technologies. For the user interface updates to appear instant, a JavaScript must be invoked in a defined intervals to ask the server for updates.

While the HTML with CSS and JavaScript in theory offer features that match other RIA technologies, the applications get much more complex and less maintainable, due to a lot of application logic written in a scripting language on the client side. For developing RIA applications, the HTML is often mixed with Adobe Flash because the development in pure HTML gets more difficult for highly interactive applications.

There are many development tools for HTML and related technologies that enable both visual editing and direct code editing. The key problem remains debugging.

Google Web Toolkit

Google Web Toolkit (GWT) is based on an idea similar to UIProtocol in terms of moving all application logic to server side. It uses Java to JavaScript translator that enables to write application logic in Java programming language having all the advantages of statically typed language while developing client-side application logic. For the client side logic there is only a subset of Java API that can be translated to JavaScript available. For the server side application logic it is possible to use full set of Java API. User interfaces in GWT are created programmatically using Java language. Parts of user interfaces generated by GWT may be embedded into static HTML websites and may also use CSS styles for modifying look and feel of the user interface. All user interface components in GWT provide methods that enable directly setting HTML-related properties such as CSS style.

The main disadvantages are that the user interface is created programmatically. While creating user interfaces programmatically is very flexible it is not natural for user interface designers. Application logic running on client and server may also be mixed with each other and also with creating a user interface.

During development GWT applications are run in a special environment (hosted mode) that enable to debug the application directly in a Java IDE overcoming some shortcomings of JavaScript debugging and accelerate development. After debugging in the hosted mode the application is evaluated directly in web browsers to solve browser-specific problems.

GWT is supported in all the major Java IDEs including Eclipse, IntelliJ IDEA and Netbeans.

Microsoft Silverlight

Microsoft Silverlight [7, 8] technology is Microsoft's competitor to currently dominating Adobe Flash. It is based on Microsoft .NET Compact Framework and aims to provide similar features to Adobe Flash while making the platform more developer friendly. Microsoft Silverlight enables to write application code in any .NET compatible language, but API is limited to .NET compact framework due to security concerns and dependency of .NET framework on native Microsoft Windows-only libraries.

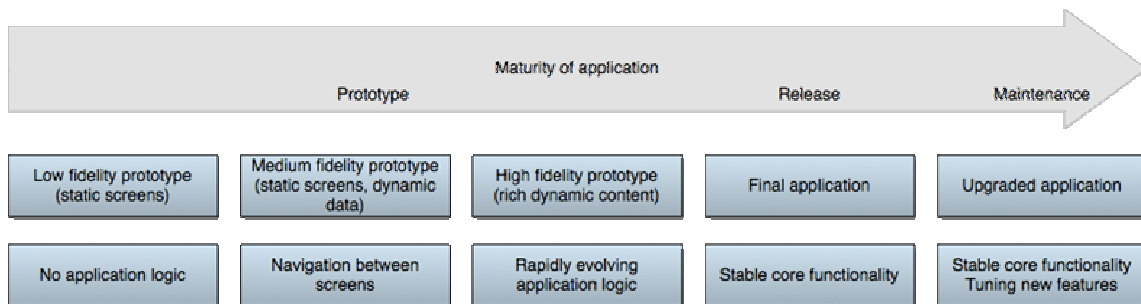


Figure 3: Application Development

Development Process

The major motivation behind UIProtocol was to create a solid foundation for rapid prototyping of user interfaces and in the same time enabling the prototypes to be converted to fully functional applications without additional development. In the Figure 3 is depicted the typical development process of an application. First a set of prototypes with incrementally increased degree of fidelity is created. A test results from usability testing of a prototypes are employed in a next prototype. Increasing the fidelity of the prototypes leads to implementing a lot of application logic that gets debugged in the process. Not using the already implemented application logic leads to wasting a lot of resources and opens a possibility to reintroduce already solved problems.

UIProtocol enables to simply cover all phases of the development and maximize usage of already developed solutions. Ability to develop application logic in both scripting (e.g. ECMAScript) and static languages (e.g. Java) enable to rapidly evaluate and alter application logic written in scripting language and then eventually implementing it in a static language as it becomes stable.

Another focus is to enable simple adding of the dynamic behavior to a static content of a low fidelity prototype. This is achieved by binding support that enables the user interface designer to simply identify dynamic content in a prototype and also enables application logic developer to update the dynamic data without knowing how they are exactly represented in the user interface. The binding simply connects the dynamic application data with their representation in a user interface. Whenever the data are updated, all user interface components that are influenced by this change are updated automatically.

Architecture

UIProtocol was designed for the client-server architecture that is typical for pervasive applications by clearly separating the client and the server part. Still, both client and server may run on a single machine, behaving like a standalone application. This enables application developers to choose how the application will be deployed very late in the application development lifecycle.

The client in the UIProtocol architecture is a terminal device that is used by the user to interact with the system and is defined by the following properties:

1. Insecure environment (a device running client may be stolen or abused by user to threaten server or be used to send malformed data to server or the communication link between client and server may be attacked)
2. Directly receives a user's input
3. Renders interface for a user
4. Not allowed to perform any critical application logic

The server in UIProtocol architecture is defined as the following:

1. Running on a trusted machine (thus the application can directly access data, check user input, etc.)
2. Performing critical application logic and manipulating persistent data
3. Allowing multiple clients to be connected in the same time and ensuring proper information exchange between clients (synchronization and transactions)

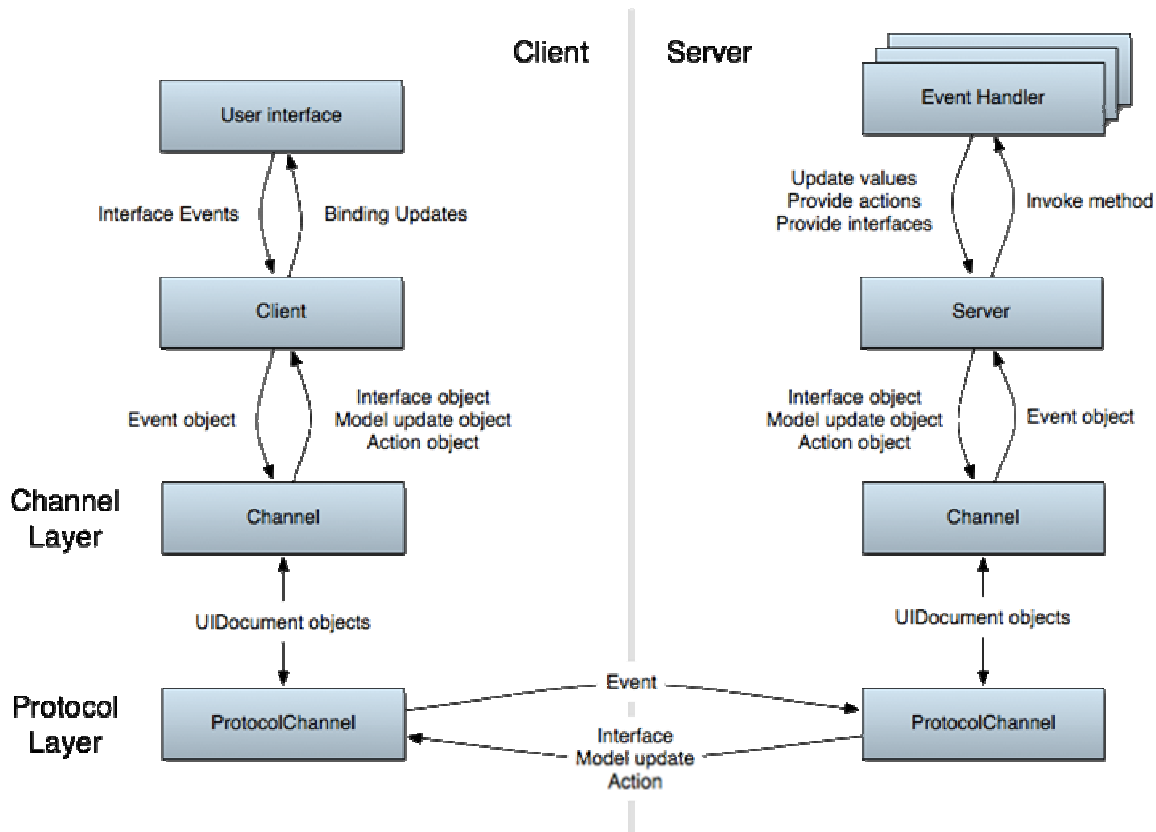


Figure 4: UIProtocol Client-Server Architecture

A UIProtocol server provides the client the interfaces that should be rendered and dynamic data (application-wide models) that should be displayed in the user interface. The client is allowed to send only events to server without actually knowing what the events will cause.

The communication and type of information that are exchanged between client and server is shown in the Figure 4 that describes current client and server implementation in *i2home* project.

Separation of UI and Application Logic

One of the most important features of the UIProtocol is that it does not enable any application logic to be used on the client side. Although an implementation of a server may offload some application logic to the client, this process should be totally transparent from the developer's point of view. No imperative (scripting-like) language features are enabled by design in the UIProtocol.

The main purpose of separation of the application logic and the user interface was to minimize the amount of information exchanged between the user interface designer and the application logic developer.

The information that is exchanged between the user interface designers and the application logic developers is limited to identifiers of events that are send from the client (and their properties) and to meaning of values in application models containing the

dynamic data. This information is usually provided by the user interface developer, as he/she usually identifies what data in a user interface should be dynamic and what application logic should be executed when the user interacts with the user interface.

Binding

Binding is essential for separating data from the user interface in the UIProtocol client. Thanks to the data binding, the server-side application logic does not have to directly access the user interface structure to modify the displayed content (similarly to DOM [9] in HTML+JavaScript). All user interface components displaying the dynamic data are automatically updated when the data are changed. Additionally all dynamic data can be interpolated to a newly assigned value. This enables to use the animations very easily virtually anywhere in the user interface, thus improving user experience.

Pushed Updates

The UIProtocol clients also provide support for pushed updates as there is permanently opened socket between client and server. This enables an immediate reaction of the user interface to any activity on the server side. While keeping socket open requires some resources to remain allocated, the communication is then faster as it is not necessary to re-establish connection for every request. Furthermore, the server does not have to maintain the information it receives from an external source until all clients ask for updates, but may automatically push that information to all affected clients.

Protocol Abstraction Layer

Although currently UIProtocol is used for exchanging user interface-related information the architecture itself is not strictly bound to using UIProtocol language. Any language that has the same expressing power may be used. Each UIProtocol document is a human readable XML file that makes it easy to edit by any XML editor. However that makes UIProtocol very memory inefficient. In current environment we also use UIProtocol-B which is a binary representation of UIProtocol that is faster to process and requires less bandwidth. These features may be especially important in pervasive computing environments as mobile devices have less processing power and data transfers over wireless are usually much slower than wired connections.

As a visual editor for UIProtocol is under development we expect that the human-readable UIProtocol won't be necessary in the future and more efficient language may be used instead.

Simple Client Structure

UIProtocol was designed to be easy to process and easy to implement. No necessity to implement a scripting engine, and concept of binding used thorough the client enables simply wrap available native user interface components. There is also no necessity to implement all features of UIProtocol as some of the features such as animations and complex components may not be supported by client in order to keep user interface functional.

Language Agnostic

UIProtocol was developed to enable all application logic to be written in statically typed languages, because they enable to write more error-free code due to advanced tools available in current IDEs such as static code analysis. By design UIProtocol does not require

the application logic to be written in any particular language and the event handlers in a single application may even be written in different languages. Current Java server implementation enables event handlers to be written in Java and any scripting language supported by Java Scripting API including ECMAScript, Groovy, Python, Ruby, etc. Support for additional format of event handlers may be added in the future without having any impact on server architecture.

Using several different languages for handling events may seem to defeat the purpose of simplifying the development and make it more error-free, especially when using dynamic scripting languages for handling events. Using the scripting languages enables to alter the behavior of the server-side quickly during debugging and testing without having to recompile and redeploy the application. Additionally, using languages such as ECMAScript is simpler for less experienced developers. Writing portions of rapidly evolving application logic in a scripting language and migrate to native handlers after e.g. usability testing may significantly accelerate development as these logic may have to be modified several times before reaching mature status.

Implementation

UIProtocol Client

Currently, several implementations of UIProtocol are available with different level of compatibility with the current specification, due to the fact that UIprotocol is still under development. Currently two most advanced implementations of clients are available for Java and Microsoft .NET platforms. There are two additional client implementations for Adobe Flash and Microsoft Silverlight.

Figure 5 shows example user interfaces designed for a living room TV using .NET UIProtocol client. Figure 6 shows user interfaces designed for touch screen device that is located near doors and enables user to check status of household and control a home security system.

UIProtocol Proxy

UIProtocol proxies enable to connect clients that do not support UIProtocol to UIProtocol server. They provide abstraction level for UIProtocol server because they behave as an ordinary client while translating all events, interfaces, model updates and actions to a format that is understandable by a proxied client. Figure 7 shows a proxy that translates UIProtocol to HTTP so a web browser may be used to connect to a UIProtocol server. An implementation of such proxy in ASP.NET is available providing support for basic UIProtocol features. In future versions of servers this functionality may be directly integrated.

UIProtocol Server

We have developed 2 UIProtocol servers in *i2home* project. The server currently used in *i2home* environment is based on Microsoft .NET platform and fully supports UIProtocol except streaming, that might be used in the future for transferring VOIP data, upload files, etc.

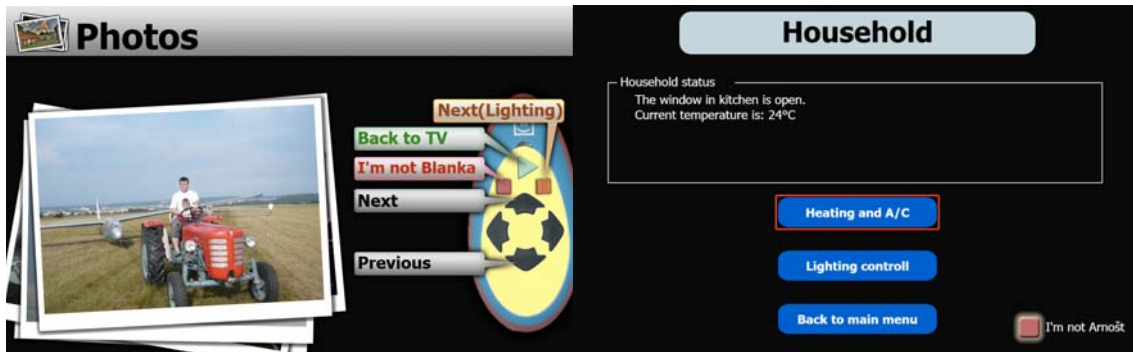


Figure 5: Interfaces rendered by .NET client

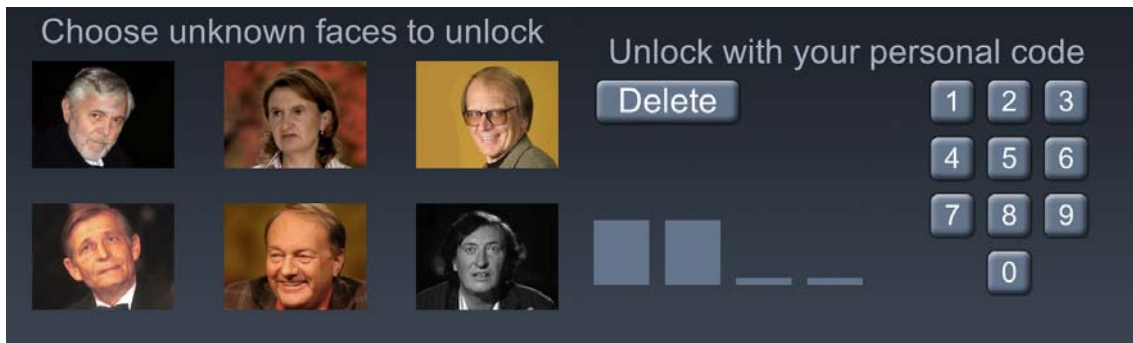


Figure 6: Interfaces rendered by Java client

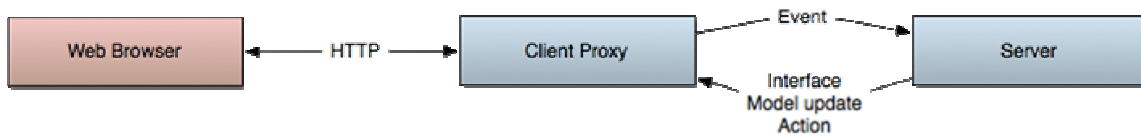


Figure 7: HTTP-UIProtocol proxy

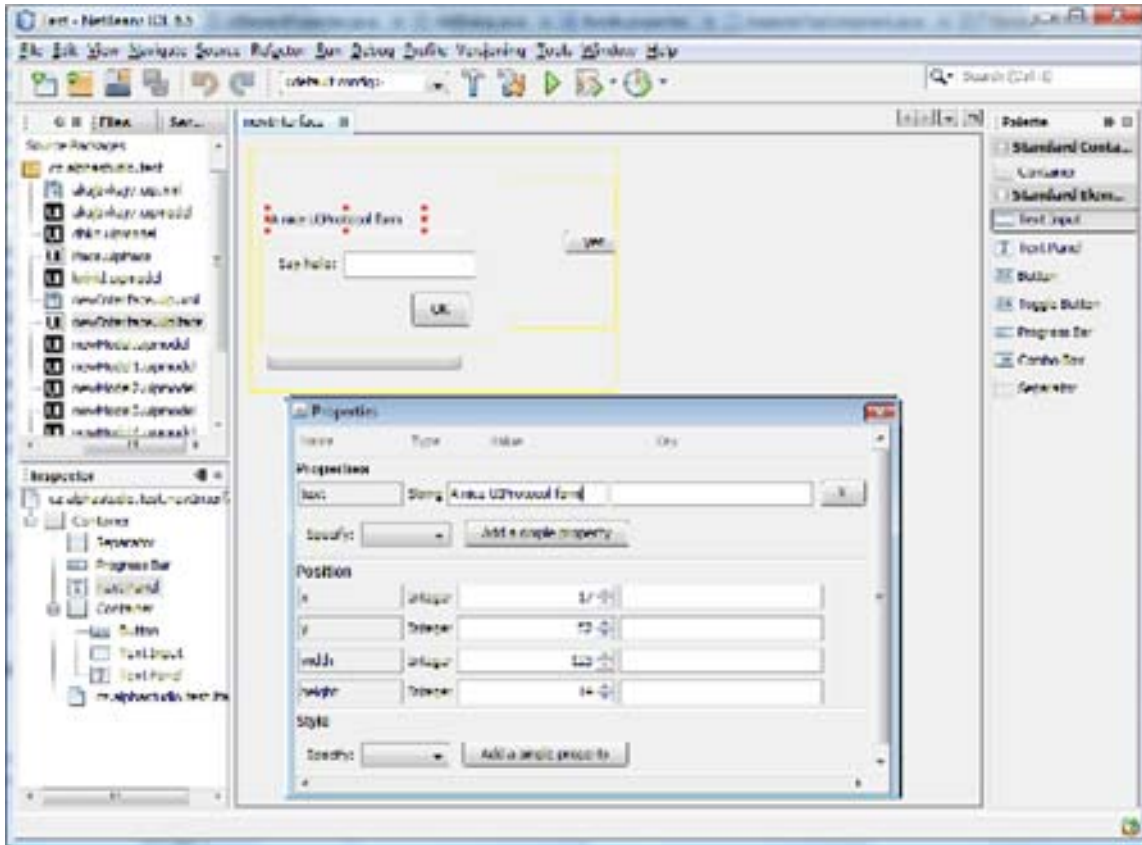


Figure 8: UIProtocol Editor

UIProtocol Editor

A visual editor (figure 8) for UIProtocol is currently under development. The editor is based on the open source Netbeans platform and will be available as a plugin for the Netbeans IDE. Currently, the editor supports drag and drop of user interface components, modifying properties of the components, positioning them, simply binding user interface components to dynamic data and attaching behavior to the components.

Conclusion

The UIProtocol draft specification is available and defines concepts and architecture of the development framework discussed in this paper. The current specification also includes support for basic user interface components, their properties, styling, positioning and layout. The final list of supported elements is not finalized, but serves well for creating user interfaces in *i2home* project. It is necessary that real-world scenarios are evaluated in order to finalize list of components and their properties.

Future Research

Future research will be focused on finalizing UIProtocol specification by strictly defining behavior for each user interface component.

As UIProtocol architecture enables implementation of application logic in any language and even use any combination of languages in a single application. Using workflows and task models may be considered for modeling high-level application logic and complex processes inside an application.

Acknowledgments

This research has been done within project *i2home* funded by the sixth Framework Program of European Union under grant FP6-033502 (*i2home*). This research has been partially supported by the Ministry of Education, Youth, and Sports of the Czech Republic under the research program MSM 6840770014. This research has been also partially supported by the Ministry of Education, Youth, and Sports of the Czech Republic under the research program LC-06008 (Center for Computer Graphics).

Bibliography

1. Weiser, M., *Ubiquitous computing*. IEEE Computer, 1993. **26**(10): p. 71-72.
2. Kiczales, G., et al., *Aspect-oriented programming*. 2002, Google Patents.
3. Souchon, N. and J. Vanderdonckt, *A review of xml-compliant user interface description languages*. Lecture notes in computer science, 2003: p. 377-391.
4. Theses, M. and G. Abroad, *A Platform-Independent User Interface Description Language*.
5. Weaver, J., *JavaFX Script: Dynamic Java Scripting for Rich Internet/Client-Side Applications*. 2007: Apress.
6. Eichorn, J., *Understanding AJAX: Using JavaScript to Create Rich Internet Applications*. 2006: Prentice Hall PTR.
7. MacVittie, L.A., *XAML in a Nutshell*. 2006: O'Reilly Media, Inc.
8. Swift, J., et al., *Professional Silverlight 2 for ASP.NET Developers*. 2009: Wrox Press Ltd.
9. Wood, L., et al., *Document object model (dom) level 1 specification*. W3C recommendation, REC-DOM-Level-1-19981001.

About the Authors



Václav Slováček is a PhD student at Czech Technical University (CTU) in Prague. He received his master degree in software engineering at CTU in 2009. He is currently involved in *i2home* and VITAL projects that are funded by 6th Framework Program of European Union. His research is focused on development of accessible user interfaces by using task models early in application development lifecycle to optimize workflows in application for different user groups.



Miroslav Macík is a PhD student and junior researcher at the Czech Technical University in Prague, Department of Computer Graphics and Interaction. He graduated the same university in 2009 receiving his master's degree. His Master's thesis "User Interface Generator" is followed by his research where he focuses on automatic user interface generation. He is further interested in usability engineering, accessibility and web technologies. Currently he is involved in two EU founded projects *i2home* and VITAL (6th Framework program).



Martin Klíma is a researcher and a lecturer at the CTU in Prague, Department of Computer Graphics and Interaction. He received his PhD in 2009 at the same university. His research interests include mobile user interaction, data adaptation for mobile environment, computer supported collaborative work in mobile environment, and user interfaces for users with special needs like elderly or handicapped. Between 2001 and 2002 he was working on adaptation of multimedia documents on PDA at ZGDV Darmstadt, Germany (analysis, design, prototype implementation). After his return to Prague he became involved in number of EU funded projects including Mummy, ELU, *i2home*, Vital, Vital Mind, Aegis and others. He is an author or a co-author of more than twenty publications on various international events in HCI.