Research on Ad Hoc Personal Area Network Application Development Kit

Arvo Sulakatko
Information and Communication Technology
Tallinn University of Technology
Ehitajate tee 5, Tallinn, Estonia
arvo.sulakatko@jsc-solutions.net

***Abstract*— In order to empower an individual through personal area network applications a new development perspective is to be introduced. For a long time the developers have been bound to a specific programming language paradigm dictated by the target platform. This is no longer the limitation. In this paper we will introduce and discuss a new way to think about programming a multi device application, part of which runs on a mobile or wearable device, part of which will be accessible over the network in a web browser. The paper will describe the development and user experience of future GPU accelerated web applications delivered by an ad-hoc network notification from a personal area network application.**

Keywords- future web applications, personal area network applications

I. INTRODUCTION

In the not so distant past programmers started with a programming language, created source code and then had a compiler produce the executable binary. Then the garbage collection and the virtual machine was introduced. The programmer was no longer just talking to the hardware, no longer just talking to the current multi process operating system. The programmer had to know his program was to run in a virtual machine which would provide additional runtime information like type reflection information. The programmer had the luxury of reusing the common type system available for this specific virtual machine. At each new iteration of the programming language and the underlying virtual machine, the program source code started to remind natural language. Then we had the web revolution. Part of the application was still able to run under controlled environment as the application server, yet part of the application had to travel to the user, into the web browser. Then we had the mobile device revolution. The developer can no longer expect a specific operating system version. No longer would the developer know what kind of the device will want to see and use the application. This adds up in terms of complexity and cost. Given the end user would be using multiple mobile or wearable devices, which need to share a coherent session of experience a new way to think about development has to be introduced.

While source code can be argued to be rather ambiguous, the byte code version of it is not. What is a program? What is an idea? Could every idea, every research idea be represented as program in near future? What if a word by itself would be a program, a word from which a sentence could be built, a sentence of which ideas are to be built.

Given this type of a thought process, the research question was formulated - What would it take to have a program automatically to be split and then recompiled for different environments, while preserving a single application experience and what should the user expect from future web applications?

II. DEVICES

Computers are everywhere, some will run unnoticed. Already now, a typical technology consumer has a set of smart cards, a mobile phone, a tablet and a laptop. Soon enough smart watches and smart glasses will become available. These devices form a personal area network. No longer can a programmer just write source code and have it compiled to single binary by the good old compiler and have the application span across those personal devices to create a single coherent user experience. This is where this research will fill the void.

What if we want to be able to write a program, and then have it split for all of our devices. What if we want our application to receive a shake event from the smart watch or mobile device just to pop up a notification on the smart glasses. What if we want to virtually throw our application from our phone to our laptop to be opened on the bigger screen with a bigger keyboard. What if people want to initiate and share a game session.

III. RELATED WORK

In recent years there has been a lot of interest and activity in the field of multi-device and multi-screen research domain. For example there has been some research [1] in dynamically selecting which part of the active web application should be made available on alternative devices. Another research [2] has put forth effort into translating input forms [3] between devices. Some interesting work [4] is also done in visual separation and cross display mouse pointers [5].

The most remarkable work is being done in the area of 3D research with autostereoscopic displays [6] and 3D street view [7] application. In a way everybody seem to be working on web applications or related fields. The technology described within this paper does not directly compete with cloud based remote view applications, like

Fig. 1. Personal Area Network Application. A single application codebase is shared between different types of mobile and wearable devices with different capabilities to enhance eachother and form a coherent experience..

Onlive and Gaikai. The primary difference is to enable and empower the personal area network and move away from the dependency on any cloud services.

IV. Security

In the wake of cloud computing and its constant threat of being spied upon, the applications running in the personal area network will start to replace all cloud services. The user will have full control of the devices, data and the programs. In a way it will create a personal internet experience just for the user, where third party will not have any way to interfere.

Given that your mobile device can have a constant high speed internet, it should be apparent of how it will start to serve as your own domain name service for web applications including email.

V. Personal Area Network Application

The picture above has illustrated a typical future user experience. To start a gaming session, the user shall express the intent on the tablet device by tapping the favorite game icon. Given all the devices are visible to each other in the local wireless network a multicast notification will be sent to let any other device know of the new activity. The laptop, having a bigger screen can then show a notification on the monitor. Either by actually clicking on the new notification or shaking the tablet the application can choose to continue and open itself on the laptop. Bear in mind the application originally is only installed on the tablet. The laptop does not have to have it installed as it can be streamed as a web application. To start a data entry application the steps are the same. The user shall pick an application and tap on its icon. The application can run on the tablet itself or have it opened on a nearby laptop either by clicking on the new notification or by shake. The data entry application (see fig. 3 on page 4) most likely makes use of a server side database to store all the entries. The editing session between multiple devices can be synchronized just like a multiplayer game, which in turn will provide the user the experience of one single coherent application. Similarly a sensory application can dislay the status of a battery or any other indicator of choosing. This display again can be opened on the tablet or on the laptop.

Upcoming smart glasses will also provide a small screen, which will allow one to see notifications from either the mobile device in the pocket or the smart watch on the hand. Notice that any device could have any number of sensors, from which the personal area network application would be able to benefit from to make itself useful for the user.

VI. Technology

The part of the project developed to do the heavy lifting, is called jsc, originally called JavaScript Compiler. The core library, which has the type definitions reimplemented for other platforms is called ScriptCoreLib. It has been written, within a number of years, in CSharp programming language running inside the .NET virtual machine. The main objective for jsc is to read and analyze .NET byte code, split it between client and server and then rewrite it to the source code of the target programming platform's language. In the illustration above any tablets, smart glasses, mobile phones and smart watches are expected to run Android and are to be programmed in Java programming language. The laptop is expected to have a modern web browser like Chrome which in turn supports JavaScript and Adobe Flash, which is to be programmed in ActionScript programming language.

While virtually any programming language, including CSharp, Visual Basic and FSharp, that compiles down to a .NET assembly could be rewritten by jsc to any of the target languages, it is limited in its current form and version as not all language constructs may already be supported or tested.

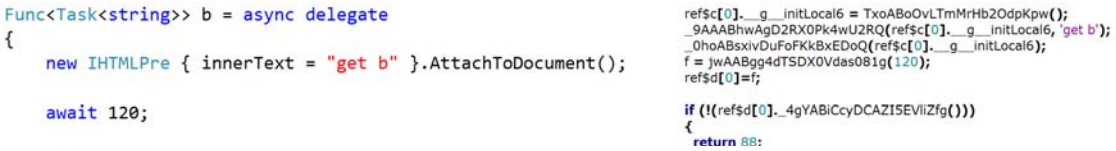

Fig. 2. A trivial async keyword example. Here we are defining an anonymous delegate, which is to be invoked asychronously. While running it will output text, wait for 120ms and then return text. On byte code level, this will be represented as a state machine. When jsc needs to convert it to JavaScript it will simplify the state machine and only then proceed to convert it. On the right side of the illustration we can see how does the generated JavaScript look like for the same delegate..

VII. What does it Take to Support 'Async' For JavaScript

Application development would not be complete if it did not use any background threads. Some tasks take longer and they may lock up the user experience by not responding. A longer calculation for example could run in the background, in isolation. The common type system allows the use of threads. In this case, the target platform is JavaScript, running in a web browser. Until recently there was no good way to translate the threading feature. As such, if a third party component wanted to do a longer calculation, and it was subject to being translated to JavaScript, it would lock up the user thread. But recently, the web browsers have been introduced to Web Workers, which allow to construct a background thread.

The async keyword was added to .NET 4.5 CSharp 5.0 programming language. Since jsc does not work on the source code level one could think such new features could be translated automatically, as byte code does not actually change. This is not the case.

Behind the scenes the async keyword will be compiled into a special state machine, which makes use of constructs which may not directly be available in the target language. The state machine also makes use of a set of types defined in the base class library, which now need a reimplementation for the target platform.

A lot of work is required to enable cross compilation. For example if the application is written in CSharp programming language and it wants to use the class System. Random then for each target platform an implementation must be provided. In this case the core library, ScriptCoreLib framework, will implement that class for Java, PHP, Action Script and JavaScript.

For the developer, a simple language feature is now available, which he can now make use of to debug the application inside .NET and also have it converted, for example, to JavaScript. Yet, to support a language features like this takes effort and is rather difficult at first, but once done, makes development much easier.

When a task needs to run on a background thread, in JavaScript, jsc will redirect the function call to a web worker and pass the function name to be called. The current limitation is, that the nested delegates cannot directly reference outer scope variables.

VIII. Components

Basically every class or type within a .NET assembly can be packaged as a separate library. Such a library can be reused in multiple projects. With jsc, such components can be used on different platforms.

The problem until recently was the discoverability. The current solution is to package the assembly into a NuGet archive and host it on a web server via RSS feed.

A component can be almost anything. It can, for example, be a 3D WebGL application, 3D Adobe Flash application or a 3D Android application. This truly allows to define a component once and have it available later for other projects.

IX. Discussion

The most simple example to start with is the client server method call. Now what if we introduce a web worker. The client has to talk to the web worker, which in turn can talk to the server. What if the application also has an Adobe Flash component. It has to serialize and talk to that, too. Now what if the Adobe Flash component or the JavaScript want to talk to the GPU? If this was not yet complicated enough, what about introducing a shared session, for example a game session.

At this time, only the most basic support for data sharing has been researched. Future research will be required to make the flow of data feel more natural. Most likely, it will look like running another thread for the application, but which is in fact running on a different device all together.

X. Conclusions and Future Work

In this paper we have introduced a framework which allows to create complex multi device web applications. We looked at the primary use scenarios and device combinations which users are expected to have. Then we looked at the architecture of a modern web application, after which we introduced and described examples for demonstration.

References

[1] G. Ghiani, F. Patern`o, and C. Santoro, “Push and pull of web user interfaces in multi-device environments,” in Proceedings of the International Working Conference on Advanced Visual Interfaces. ACM, 2012, pp. 10–17.

[2] F. Patern`o and G. Zichitella, “End-user customization of multi-device ubiquitous user interfaces,” in 5 th International Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI 2010), 2010, p. 41..

[3] F. Patern`o and C. Santoro, “A logical framework for multi-device user interfaces,” in Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems. ACM, 2012, pp. 45–50..

[4] J. Cauchard, M. L¨ochtefeld, P. Irani, J. Schoening, A. Kr¨uger, M. Fraser, and S. Subramanian, “Visual separation in mobile multi-display environments,” in Proceedings of the 24th annual ACM

symposium on User interface software and technology. ACM, 2011, pp. 451–460..

[5] M. Waldner, C. Pirchheim, E. Kruijff, and D. Schmalstieg, “Automatic configuration of spatially consistent mouse pointer navigation in multidisplay environments,” in Proceedings of the 15th international conference on Intelligent user interfaces. ACM, 2010, pp. 397–400..

[6] O. Nocent, S. Piotin, A. Benassarou, M. Jaisson, and L. Lucas, “Toward an immersion platform for the world wide web using autostereoscopic displays and tracking devices,” in Proceedings of the 17th International Conference on 3D Web Technology. ACM, 2012, pp. 69–72.

[7] A. Devaux, M. Br´edif, and N. Paparoditis, “A web-based 3d mapping application using webgl allowing interaction with images, point clouds and models,” 2012..

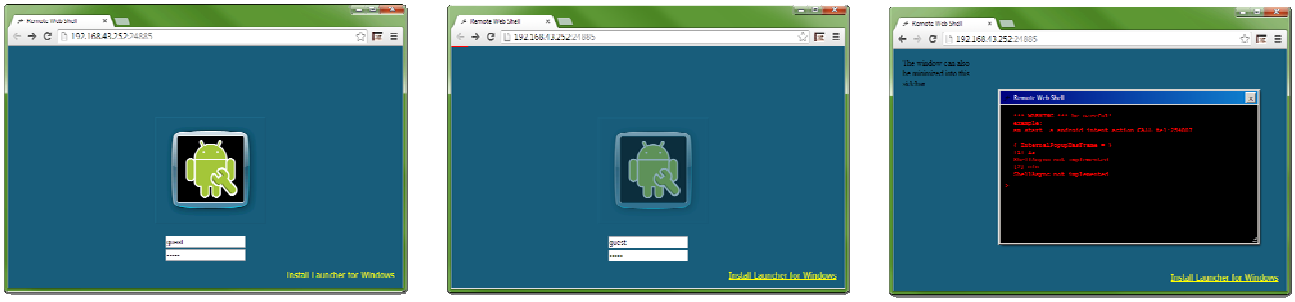

Fig. 3. Remote Web Shell. An additional example which allows to issue commands to the remote device. This concept demonstrates how an android device can send out a notification to the laptop, have user open the application, enter credentials and then request an encrypted secondary application to be sent to the browser. This also demonstrates two components, one which allows to minimize the embedded window and another which allows to pop out the embedded window.