MobDSL: A Domain Specific Language for multiple mobile platform deployment

Dean Kramer
School of Computing and Technology
Thames Valley University
London, UK W5 5RF
Email: dean.kramer@tvu.ac.uk

Tony Clark
School of Engineering and Information Sciences
Middlesex University
London, UK NW4 4BT
Email: t.n.clark@mdx.ac.uk

Samia Oussena
School of Computing and Technology
Thames Valley University
London, UK W5 5RF
Email: samia.oussena@tvu.ac.uk

***Abstract*—There is increasing interest in establishing a presence in the mobile application market, with platforms including Apple iPhone, Google Android and Microsoft Windows Mobile. Because of the differences in platform languages, frameworks, and device hardware, development of an application for more than one platform can be a difficult task. In this paper we address this problem by the creation of a mobile Domain Specific Language (DSL). Domain analysis was carried out using two case studies, inferring basic requirements of the language. The paper further introduces the language calculus definition and provides discussion how it fits the domain analysis, and any issues found in our approach.**

***Index Terms*—Domain Specific Languages, Mobile Computing, Platform-Independence**

I. Introduction

Today, the penetration of modern smart phones is vastly increasing with over 172 million smart phones shipped worldwide in 2009 [1], and with the emergence and successes of sources for consumers to install third party applications opens a new market for developers to reach consumers. However, developing an application for multiple mobile platforms can incur different obstacles including differences in development tools available, different language and platform constraints and availability of software libraries. Difficulties in producing software for more than a single platform has been evident for many years outside of the mobile realm. For decades, software portability used to be large concern during development, mainly due to very large spectrum of different CPU Instruction Set Architectures (ISA), which also wasn't helped by the large variety of Operating Systems in use. Nowadays though this has become much less of an issue, largely due to many factors including the decrease in CPU ISAs, the dominance of a limited number of operating systems and commonly used languages including Java. Largely though, because the mobile market, with respect to third party applications is fairly new, there are large differences in implementation languages and development environments. Software porting and cross platform development remains the most common method for multi-platform development. For large software companies this is not a problem, but for smaller mobile business this presents a problem. Firstly, as most of the different platform use different languages, there can be learning curve issue with the developer needing to know each language and development environment. Secondly, it will require the business to invest into more testing equipment for the different platforms. By being able to write once and deploy to many, this can help application be delivered faster and more economically.

Within multi-platform development for mobile devices, we explored different approaches that were seen as more viable, and suitable solution to our problem.

A. Implementation Approaches

Frameworks: The use of frameworks can be seen as a method of software abstraction using common code, which can be overridden and extended by a user. Within mobile development, frameworks have been developed to help with specific tasks including media playback, access to sensors and graphic and UI manipulation. Further on we discuss a software framework[2], which can be used to help make code bindings between the different platform specific frameworks. This method concentrates on solving all computational problem, which can increase complexity in application development, further becoming a hindrance to the developer. An alternative method includes mobile web applications.

Web Applications: A mobile web application essentially is a regular Internet application designed to fit the average screen sizes of most mobile devices, bringing various benefits to the developer. Some applications that require high amounts of processing can greatly benefit from allowing the processing to be handled in the cloud while the device merely has to process the UI. Other benefits of this method would be that for some the use of HTML, CSS, and images may be easier to develop, especially for particularly simple applications.

One large problem to this approach is the reliance on network connectivity for the application, which can be in some situations either not be available or not desired. Web applications in general can have shortcomings in the amount of rich UI widgets, with animation for certain widget interaction being increasingly difficult to implement in a mobile web application. Other problems and some what linked to the previously mentioned disadvantage from doing web based applications is the limitations of the web-browser on the mobile devices, possibly leading to inconsistencies in application functionality

between different platforms because of lack of APi for using different device components (e.g. accelerometers, vibration motors, GPS etc). One way that this can be overcome on some mobile platforms is by creating web applications that are run on the device, but since because of the fragmentation in webkit implementation for application development is very high including differences in api calls for the specific devices, this is rather unfeasible for our goal. Because of this, the creation of a Domain Specific Language was chosen as our solution.

Domain Specific Languages: A Domain Specific Language (DSL) is primarily designed to be used in a certain area/-domain, abstracting away from the software implementation making implementation easier. Though this abstraction is designed to aid the developer, the language should merely be domain complete and not be capable of solving any computational problem making it 'Turing Complete'. DSLs have existed for many years. Languages that were created for particular domains include FORTRAN[3] used to allow direct mathematical formula, Structured Query Language (SQL)[4] for database access and manipulation, and Algol[5] for algorithm specification. In recent times, the use of DSLs have been proposed and used in different domains including the production of rich web applications [6], mashups of web apis and services [7], and system integration [8]. Because of the complexities in mobile development, we believe there is room for abstraction in the development for mobile devices.

The outline of this paper is as follows: Section 2 explores related work, with Section 3 giving the domain analysis introducing our two case studies. Section 4 introduces the languages with the basic Calculus, how it meets our requirements and the architecture. Section 5 we provide some discussion into the issues involved in our approach and finally Section 6 describing our furture work.

II. Related Work

The idea of being able to deploy one application to many platforms is far from a new one, with the work in this section outlining a few of the approaches. When writing cross-platform and multi-platform applications, abstraction is needed, and this can be done via DSLs/Modelling, or alternately using a software library/framework.

A. Frameworks

The DIMAG Framework[9] was developed for automatic multiple mobile platform application generation. This was accomplished by creating a declarative definition language which is comprised of 3 distinct parts; firstly a language DIMAG-root, provides references to the definitions for workflow and user interface in the application; secondly the language State Chart eXtensible Markup Language (SCXML) defines the workflow by the definition of states, state transitions, and condition based actions; and finally DIMAG-ui language based on MyMobileWeb's IDEAL language using CSS to control the user interface.

The shortcomings with this method is that it relies on server-side code generation and download, which can preset problems because of the inability to install application from sources other than the manufacturers app store. The approach described in this paper also differs in that the native-platform executable code is not compiled directly from the DSL but more interpreted using a virtual machine.

Other frameworks include XMLVM[10], [11] developed at San Francisco State University, was created to support byte-code cross-compilation and avoid source-code translation through the use of a toolchain. This toolchain currently translates Java Class files and .Net executables to XML documents, which then can be output to Java byte code/.NET CIL or to Javascript and Objective-C. This toolchain was firstly used to cross compile Java applications to AJAX applications [12], because of the lack of IDE support and difficulty in creating an AJAX application. Further work to include Android to iPhone application cross-compilation[2] was completed. API mapping between the two platform was carried by the creation of a compatability library.

Though this method would be a feasible method of doing apps for android and then creating native iPhone apps, it does not address higher level programming, and with the API mapping, further issues can occur making development more confusing for an amateur developer. These methods concentrate too highly on supporting all problems unlike Domain Specific languages.

B. Domain Specific Languages and Modelling

As said earlier, DSLs can be used in abstracting the concept to a higher level, leaving low-level boiler plate code, thus making the software easier to write and maintain. Notable work of [13], [14], [15], [16] have quantitative results showing benefits of using DSLs.

Research into the abstraction of the development of mobile applications by the use of a Model Driven Development (MDD) has been much researched. The first study [17] is to combat the difficulties for people when developing and deploying applications on mobile devices. This includes the development of a graphical modelling tool, which will then transfer to a formal XML model, and then processed to code for the specific platforms. This slightly differs from [18] which mainly concentrated on the production of User Interfaces for Visual Basic and embedded Visual Basic platforms by using specific models to each platform, instead of a single model for many platforms. The more recent DSLs in other areas include [6] which concentrates on the abstraction of web applications to lower the overall complexity of the application and boilerplate code. Further work on this DSL lead to the creation of Platform Independent Language (PIL) [19]. PIL was developed as an intermediate language, to provide a scalable method for developing for multiple platforms. A drawback of this method is currently it lacks support for mobile platform development. Other efforts for making mobile application development easier include Googe Simple[1] , a BASIC dialect

[1] http://code.google.com/p/simple/

for creating Android applications, and more recently the Google App Inventor[2] which is based on Openblocks [20] and Kawa[3]. Particularly Google App Inventor has vastly abstracted app development, but only supports development of Android applications.

III. Domain Analysis

Before designing and implementing any DSL, sufficient domain analysis needs to be undertaken. In this section we discuss different technologies commonly used in mobile applications, and then discuss the case studies we shall be using for the language validation in later sections of this paper.

A. Case Studies

This language was conceived through the use of two iPhone application case studies, that were created for a local SME. These applications do have differing features, but essentially do not consist of overly complicated logic and user interaction, making them good candidates for being used in the domain we targeting.

1) Tour de France (TDF2009): This application was created to help support people in following the 2009 series of Tour De France. Firstly the application required a method of transferring and receiving data from an external server for two different reasons. Firstly for the stage results, and secondly for the general data including information about the Teams/Riders and all the Stages involved in that year, this helped us achieve a very small installation size. The data communications were done via XML files parsed using the iPhone SAX-XML Parser, one created with the static data, and one generated every day with the current results. The static data was viewable by entering the particular sections of the application. Inside the stages section, fly-through videos to help illustrate the course and terrain, large high resolution gesture controlled pictures were incorporated.

Figure 1. TDF Screen Transitions

[2]http://appinventor.googlelabs.com/about/
[3]http://www.gnu.org/software/kawa/

2) Lyrical Genius: This application was created as a game that consist of quiz questions relating to different lyrics in songs. This game though still using the Apple Cocoa Framework is quite different in many ways. Firstly this application does not use XML files as persistance of data, but uses a SQLite database for storing level and question data. Other features of this game include music that is played in the background that can be switched on/off and sound effects for if the user chooses the correct or incorrect answer. These features require threading, which is one issue we must consider in the DSL. The game also includes a timer, for which the user must get a number of correct answers within a time limit. This makes use of threading again, and also another important area as the use of timing can be needed in many different contexts. Finally, though the use of Cocoa was made, with the use of button images and background images the application was made to not look like a conventional "apple style" iPhone app.

Figure 2. LG Screen Transitions

B. Domain Features

Based on the case studies above, we can define a set of features that the dsl must support. In the case of GUI implementation, in the iPhone and Android development OpenGL can be used. OpenGL is a cross-platform graphics language which supports the ability to draw 2D and 3D objects, but in this paper we are concentrating on the platform framework for the GUI.

Screen Size: mobiles support only a limited size display. This size leads to a relatively small number of GUI features, therefore there is more scope for building these features into a common language. The standard iPhone resolution is 480 by 320 pixel and the iPad supports a 1024 by 768 resolution. This compares to the Android screens, which vary by hardware vendor but resolutions range to about 480 by 800 pixel. Apple have currently settled the differences in screen display resolution by the use of graphic scaling. This method can seem an effective way to allow iPhone apps to run on an iPad, but this comes with its flaws. Graphic scaling of very small low quality images can make them look unappealing to the user.

Also UI design on the iPad, because of its size difference will be slightly different than on the iPhone. This will require developers to create applications with interfaces to suit that device.

Layout Control: layout control is an important consideration. Android controls layout through the use of XML files, supporting different layout styles. The main style types consisted of linear, relative and absolute. Android now has deprecated absolute positioning, due to the fragmentation in different hardware vendor screen resolutions (see above). This compares to iPhone, which can do programmatic layout and prebuilt XML type interfaces using Interface Builder. Interface Builder can help the user easily create UIs, but these layouts would be less dynamic than programmatic ones.

GUI Element Containership: both iPhone and Google Android platforms use a form of GUI element containership. In iPhone development, the emphasise is on the application *Window* and it's *Views*, with *Subviews.* These are then 'stacked' onto each other to create anything from a simple to complex interface. With the Android a similar model is used, except with *Views* and *ViewGroups*. Interface control on both platform have similarities and differences. On the iPhone, views are normally controlled by the use of View Controllers, which are where widget event handlers are implemented. In comparison Android development uses *Intent*s and *Activity*s. Example of a ViewController interface for iPhone:

```
@interface AboutViewController : UIViewController
```

Event Driven Applications: largely the applications we are targeting are event driven. In implementation, registering methods for event handler is done dynamically, not statically. This method means there is a lack of checking at compile time to prevent an application crashing. An example of a event listener for iPhone:

```
[btnMenu addTarget:self action:@selector(backToMenu)
forControlEvents:UIControlEventTouchUpInside];
```

Hardware Features: modern day mobile devices come equipped with many different features. These features include microphones, accelerometers, GPS, camera, and close range sensors. These features tend to be fairly standard in their behaviour if they are supported by the platform. The use of these features is done using the platform specific framework.

Concurrency: the use of concurrency in mobile applications is paramount. This is carried out by the use of threads, for instance a UI thread starts with the execution of an iPhone or Android app. Because this thread is used for the UI elements of the application, heavy or concurrent tasks should be allocated in its own thread. This can help avoid UI halts and a 'laggy' experience for the user. On the iPhone platform, threads can be implemented in various ways including POSIX Threads and NSThread. The difference between the two are that the pThreads are a C/C++ library and NSThread is a Cocoa-native thread. On Android, concurrency can be implemented through the use of Thread Classes, just as you would do it in Java. Example of a thread in iPhone:

```
[NSThread detachNewThreadSelector:
@selector(playMusic) toTarget:self withObject:nil];
```

Object-Orientation: mobile development also is done using Object-Oriented (OO) languages. In the iPhone the main language used is Objective-C, though support for C++ and the non-OO C can also be used. This compares to Android, which uses Java, but with different libraries and uses the Dalvik Virtual Machine (VM) instead of the Java VM, because its characteristics support mobile devices more. Applications are built by constructing new and extending existing class/object types.

Transitional Behaviour: state machine transitional behaviour is very common in mobile device applications, and can be found on the Android platform. Each Activity can be viewed as a state machine that stores state and actions by the user, which then causes transitions between different views or activities.

The language itself is not designed to be an approach for multi-platform deployment in all types of mobile applications, but more designed to support a particular domain/area. The type of applications this language is designed to support are applications not requiring extensive user-interaction, but are driven by data stored in a external server.

C. The Mobile Application Domain

In the first part of this section we discussed different implementation technologies used in mobile application development. Furthermore with the case studies described many of the technologies were used and will be required in the DSL. In the DSL there should be ability to produce applications similar to the case studies, but not be capable of solving every computational problem.

IV. MobDSL

Largely, programming languages are made up of two basic concepts, the syntax or the actual expressions that are compiled or interpreted, and the semantics that bring meaning of the language. In this section, this will be explained with the inclusion of the underlining calculus and examples to help express key features. Following this code snippets of the case studies implementations will be given with explaination.

A. A Mobile Application Calculus

$$
\begin{array}{lll}
E & ::= & & \text{expressions} \\
 & & V & \text{variables} \\
 & | & \textbf{let } V = E \textit{ in } E & \text{local defs} \\
 & | & \textbf{letrec } V = E \textit{ in } E & \text{recursive defs} \\
 & | & \textbf{fun}(V...V).E \mid E(E,...E) & \text{funs, apps} \\
 & | & \{V = E; \ldots; V = E\} \mid E.V & \text{records, ref} \\
 & | & [E, \ldots, E] & \text{lists} \\
 & | & \textbf{widget}(V)EE & \text{widgets} \\
 & | & \textbf{if } E \textbf{ then } E \textbf{ else } E & \text{choice}
\end{array}
$$

Figure 3. Calculus Definition

The basic calculus for the mobile applications language is shown in figure 3. It is based on the λ-calculus extended with widgets for managing mobile application components. The formal semantics of the language is out of the scope of this

paper, however the operational semantics reduces expressions in the normal way until a widget is encountered. A widget names an external library feature that is displayed on the device and selects a handler based on events raised by the device platform. The rest of this section describes the key features.

B. Domain Features Support

This section describes how the mobile calculus supports the key features required by mobile applications. The features are described with respect to the following example which impliments a slightly simplified version of the quiz application shown in figure 2:

```
fun(Bs,Qs,S)
 letrec
  main(S)=widget(Main)
   {image=Bs.main}{play()=levels(S)};
  levels(S)=widget(Levels)
   {level=S.level;image=Bs.levels}
   {menu()=main(S);level()=level(S,S.level)};
  level(S,level)=widget(Level)
   {level=level;image=exp.Nth(Bs.level,level)}
   {play()=play(S,level);back()=levels(S)};
  play(S,level)=widget(Play)
   {question=nth(nth(Qs,level),nth(S.Qis,level))}
   let is=S.Qis in
   let i=nth(is,level)
   in {answer()=play(S.Qis:=(is[level]:=i+1,level));
       timeout()=main(S)}
 in main(S)
```

The definition contains examples of the general calculus features: function definition and application; (recursive) local definitions; records; lists; basic values. The quiz is defined as a function with three arguments: `Bs` a record of jpeg background images, `Qs` a list of question lists, each question is a string; `S` a record of the current state of the player containing the current player-level and the number of questions attempted by the player at that level.

Displays: a display element is defined as a widget. In the quiz example, `Main`, `Levels`, `Level` and `Play` are all widgets. A widget consists of a name, its state and event handlers. The name refers to an external platform specific library component that expects to be supplied with the state and which can raise any of the named events. For example, `Main` expects to be supplied with an image (indexed in `Bs`) and can raise an event `play` that causes a transition to the `Levels` widget.

Events: each widget has handlers. When an event occurs, it is supplied to the widget on which it occurs. If it is handled there then that widget is responsible for returning a replacement widget. Otherwise the event is passed up the container tree. Events can be re-raised. The handler for an event must return a widget that is used as a replacement for the receiver on the physical platform. For example, when `Main` receives a `play` event it returns a `Levels` widget that is viewed by the user as a transition from one screen to the next.

Hardware Features: built-in widget types. If a platform does not support a type then this will raise a type-error at compile-time.

Containership: not all widgets need to correspond to visible screen elements; some are used as containers to group widgets. However, all widgets can receive, process and raise events.

Concurrency: each widget is active in its own thread. The following widget is a container of two media players both of which will start independently and either of which may raise a `terminated` event where the container is replaced with an empty container (i.e. either player terminates the other):

```
widget(Container){contents=[
  widget(Player){media='media1';state='play'}{},
  widget(Player){media='media2';state='play'}{}]}{
  terminated(mediaPlayer)=
    widget(Container){contents=[]}{}}
```

Object-Orientation: the state of a widget is a record that can contain both data (widget attributes) and functions (widget methods).

Transitional Behaviour: each widget is a state machine that handles events by making a state transition to a new widget that defines a replacement for the receiver.

C. Proposed Architecture

To help implement this DSL for use on multiple platforms, an architecture for making applications platform-independent needed to be defined shown in fig 4. In the proposed architecture, essentially there are 3 tiers: (1) the application, written and compiled using the DSL; (2) the DSL specific engine, and implemented libraries; (3) the running platform, be that Java, C#.NET, Android or iOS (iPhone). For each of the target platforms, the virtual machine will comprise of two major parts. Firstly there will the platform libraries (MobLib) which will contain the specific platform api calls. This library will contain the callable display, interface, and underlining methods of that platform. Secondly the engine, that will run the compiled code and make the appriopriate platform calls using the the bundled platform library set.

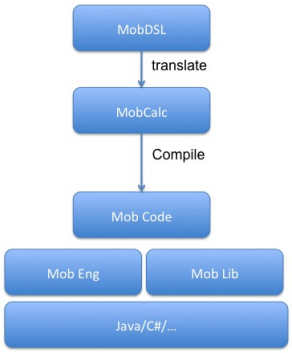

Figure 4. Proposed Architecture

V. Discussion

Attempting to make the development on mobile platforms is far from an easy task, and in most cases different approaches come with their advantages, and their disadvantages. With the DSL and the creation of virtual machines on targeted platforms, one particular benefit would be the avoidance of application installation source lock-in, which is applicable to the many users of iPhones/iPads. Though with lock-in can

come increased security for end-users through the use of application validation by that platform vendor, and can also be seen as a method to allow that vendor to decide on the types of applications it believes are right for that user/device. Through the creation and use of a VM and downloadable programs (in the form of DSL program definitions), this can be overcome after the user has the VM installed on their device. This though does mean that the VM needs to be downloaded from the appstore in the first place, which is highly unlikely because of Apple's stance on VM development. Other examples of this issue occurring includes the failed attempt by Sun Microsystems to create and make available a iPhone version of the Java VM, which includes JavaME, a branch of Java designed for mobile and embedded devices. If there was the ability to run a VM on an iPhone/iPad, development of these applications will no longer require a certain platform, as the tools will be operable in multiple desktop platforms.

Other advantages of a VM would also include the decrease in application size. This is because all the libraries and methods that are needed for any app to run are stored in the VM, and the applications are merely application definitions, leading to much smaller sizes and faster downloading times for the user. This smaller size will be also seen as an advantage to the developer through a higher abstraction, as this DSL is not targeted as producing every single type of mobile applications, but more a select type of applications similar to the two case studies above.

VI. Current State and Further Work

The calculus language has been prototyped as an interpreter in Java and the next step is to develop a mobile Virtual Machine (VM) for platforms including Android and iPhone. Because of the policies in place regarding VM development for the iPhone discussed earlier, the VM may not be accepted by Apple to be on the App-Store. One method of possibly getting around the issues with the App-store policy on VM development, could be incorporating the XMLVM [11] and instead of following a VM approach, use a compilation approach for iPhone/iPad.

Because of the problems that can occur from dynamic event and event handler association, we hope to implement a type checking system. In iPhone applications, certain UI classes in the UIKit framework can create events, which the developer then can associate and link with a particular event handler. At compile time, these associations are not checked, and can cause an application to hang and crash if and when that event handler does not exist or meet the requirements of the event. By the use of a type checking system, this can be avoided thus preventing the developer from making such a mistake in the first place. Part of a type checking system has been prototyped currently, and we hope in the future this feature will be complete.

Other areas of future work include the ability to connect to external services. This ability, will be intended as an abstract method of connecting to RSS/XML feeds including a method for parsing the documents. Because of the nature of the language, we intend the implementation of this abstraction to be in a widget.

References

[1] H. J. De La Vergne, C. Milanesi, A. Zimmermann, R. Cozza, T. H. Nguyen, A. Gupta, and C. Lu, "Competitive landscape: Mobile devices, worldwide, 4q09 and 2009," tech. rep., Gartner, 2010.

[2] A. Puder and I. Yoon, "Smartphone cross-compilation framework for multiplayer online games," *Mobile, Hybrid, and On-line Learning, International Conference on*, vol. 0, pp. 87–92, 2010.

[3] J. W. Backus, R. J. Beeber, S. Best, R. Goldberg, L. M. Haibt, H. L. Herrick, R. A. Nelson, D. Sayre, P. B. Sheridan, H. Stern, I. Ziller, R. A. Hughes, and R. Nutt, "The fortran automatic coding system," in *IRE-AIEE-ACM '57 (Western): Papers presented at the February 26-28, 1957, western joint computer conference: Techniques for reliability*, (New York, NY, USA), pp. 188–198, ACM, 1957.

[4] D. D. Chamberlin and R. F. Boyce, "Sequel: A structured english query language," in *SIGFIDET '74: Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, (New York, NY, USA), pp. 249–264, ACM, 1974.

[5] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger, "Report on the algorithmic language algol 60," *Commun. ACM*, vol. 3, no. 5, pp. 299–314, 1960.

[6] D. M. Groenewegen, Z. Hemel, L. C. Kats, and E. Visser, "Webdsl: a domain-specific language for dynamic web applications," in *OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, (New York, NY, USA), pp. 779–780, ACM, 2008.

[7] E. Maximilien, H. Wilkinson, N. Desai, and S. Tai, "A domain-specific language for web apis and services mashups," in *Service-Oriented Computing – ICSOC 2007* (B. Krämer, K.-J. Lin, and P. Narasimhan, eds.), vol. 4749 of *Lecture Notes in Computer Science*, pp. 13–26, Springer Berlin / Heidelberg, 2010.

[8] M. Shtelma, M. Cartsburg, and N. Milanovic, "Executable domain specific language for message-based system integration," in *Model Driven Engineering Languages and Systems* (A. Schürr and B. Selic, eds.), vol. 5795 of *Lecture Notes in Computer Science*, pp. 622–626, Springer Berlin / Heidelberg, 2009.

[9] P. Miravet, I. Marín, F. Ortín, and A. Rionda, "Dimag: a framework for automatic generation of mobile applications for multiple platforms," in *Mobility '09: Proceedings of the 6th International Conference on Mobile Technology, Application & Systems*, (New York, NY, USA), pp. 1–8, ACM, 2009.

[10] A. Puder, "A code migration framework for ajax applications," in *Distributed Applications and Interoperable Systems, 6th IFIP WG 6.1 International Conference, DAIS 2006, Bologna, Italy, June 14-16, 2006, Proceedings* (F. Eliassen and A. Montresor, eds.), vol. 4025 of *Lecture Notes in Computer Science*, pp. 138–151, Springer, 2006.

[11] A. Puder, "An xml-based cross-language framework," in *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops, OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, GADA, MIOS+INTEROP, ORM, PhDS, SeBGIS, SWWS, and WOSE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceed* (R. Meersman, Z. Tari, P. Herrero, G. Méndez, L. Cavedon, D. Martin, A. Hinze, G. Buchanan, M. S. Pérez, V. Robles, J. Humble, A. Albani, J. L. G. Dietz, H. Panetto, M. Scannapieco, T. A. Halpin, P. Spyns, J. M. Zaha, E. Zimányi, E. Stefanakis, T. S. Dillon, L. Feng, M. Jarrar, J. Lehmann, A. de Moor, E. Duval, and L. Aroyo, eds.), vol. 3762 of *Lecture Notes in Computer Science*, pp. 20–21, Springer, 2005.

[12] A. Puder, "A cross-language framework for developing ajax applications," in *PPPJ '07: Proceedings of the 5th international symposium on Principles and practice of programming in Java*, (New York, NY, USA), pp. 105–112, ACM, 2007.

[13] R. B. Kieburtz, L. McKinney, J. M. Bell, J. Hook, A. Kotov, J. Lewis, D. P. Oliva, T. Sheard, I. Smith, and L. Walton, "A software engineering experiment in software component generation," in *ICSE '96: Proceedings of the 18th international conference on Software engineering*, (Washington, DC, USA), pp. 542–552, IEEE Computer Society, 1996.

[14] J. Gray and G. Karsai, "An examination of dsls for concisely representing model traversals and transformations," in *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences*

(HICSS'03) - Track 9, (Washington, DC, USA), p. 325.1, IEEE Computer Society, 2003.

[15] D. Batory, J. Thomas, and M. Sirkin, "Reengineering a complex application using a scalable data structure compiler," in *SIGSOFT '94: Proceedings of the 2nd ACM SIGSOFT symposium on Foundations of software engineering*, (New York, NY, USA), pp. 111–120, ACM, 1994.

[16] J. R.M. Herndon and V. Berzins, "The realizable benefits of a language prototyping language," *IEEE Transactions on Software Engineering*, vol. 14, pp. 803–809, 1988.

[17] F. T. Balagtas-Fernandez and H. Hussmann, "Model-driven development of mobile applications," in *ASE '08: Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, (Washington, DC, USA), pp. 509–512, IEEE Computer Society, 2008.

[18] J. Stocq and J. Vanderdonckt, "A domain model-driven approach for producing user interfaces to multi-platform information systems," in *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, (New York, NY, USA), pp. 395–398, ACM, 2004.

[19] Z. Hemel and E. Visser, "Pil: A platform independent language for retargetable dsls," in *Software Language Engineering, Second International Conference, SLE 2009, Denver, CO, USA, October 5-6, 2009, Revised Selected Papers* (M. van den Brand, D. Gasevic, and J. Gray, eds.), vol. 5969 of *Lecture Notes in Computer Science*, pp. 224–243, Springer, 2009.

[20] R. V. Roque, "Openblocks: an extendable framework for graphical block programming systems," Master's thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science., Massachusetts, USA, 2007.