PULP Scription: A DSL for Mobile HTML5 Game Applications

Mathias Funk, Matthias Rauterberg

▶ **To cite this version:**

Mathias Funk, Matthias Rauterberg. PULP Scription: A DSL for Mobile HTML5 Game Applications. Gerhard Goos; Juris Hartmanis; Jan van Leeuwen. 11th International Confernece on Entertainment Computing (ICEC), Sep 2012, Bremen, Germany. Springer, Lecture Notes in Computer Science, LNCS-7522, pp.504-510, 2012, Entertainment Computing - ICEC 2012. .

HAL Id: hal-01556161

https://hal.inria.fr/hal-01556161

Submitted on 4 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

PULP Scription: A DSL for Mobile HTML5 Game Applications

M. Funk and M. Rauterberg

Department of Industrial Design, Eindhoven University of Technology,
Den Dolech 2, 5600MB Eindhoven, The Netherlands

Abstract. As applications and especially games are moving to the web and mobile environments, different tools are needed to design these applications and their behavior. HTML5 in combination with JavaScript is a promising basis for such applications on a wide range of platforms. Content producers and designers often lack the tools for such developments, or the expertise to operate existing, but too complex tools. This paper presents work in progress about a novel domain-specific language (DSL) PULP that aims at closing this gap. The language allows tying content such as images and media files together by modeling the dynamic behavior, movements, and control flow. The DSL helps abstracting from asynchronous JavaScript, state machines, and access to cross-platform media playback, which is generated in a final model-to-text transformation. The DSL and tooling were created and evaluated in close cooperation with content authors.

1 Introduction

The application domain targeted in this paper is simple educational games for children from 3–7 years. These games aim for instance at improving foreign language skills in a playful way. The games support simple touch gestures and allow for guiding the players with sounds and videos.

Producers of interactive content and applications for mobile devices face a huge diversity of platforms to develop for: from Apple's iOS or Google's Android platforms to Windows Mobile, Symbian, WebOS, Blackberry, and others there is a variety of systems available. This often involves major additions and changes to the user interface and thus leads to huge efforts spent on device specific code and testing. Apart from simply selecting a subset of platforms to target, a recent trend is to target the devices' browsers with HTML5, which is commonly supported on most platforms. HTML5 in connection with CSS3 not only supports transitions, animations, often hardware-accelerated, but also allows for native media playback and local storage of application data. The browser as a platform also requires programming in addition to the content authoring, and although the programming model is relatively light-weight (consisting of HTML5 markup, styling and JavaScript programming), this expertise is often beyond the abilities of content producers.

The central problem statement can be formulated as: how can the authoring of interactive content on the web for next generation platforms and novel input methods be supported by modeling in the form of domain-specific languages as means to enable end-user programming with rapid iterations? Tackling this problem by designing a language should mainly involve the content author and her expertise as a domain expert. What is behind this problem is the assumption that content authors and designers can benefit from domain-specific languages which represent a formal description of the final product, but allow to specify in a way that resonates well with the working habits and prior expertise of the domain experts.

The paper introduces the domain-specific language PULP that allows for scripting all structural and behavioral aspects of an interaction HTML5 application. While images and media files represent the content basis of the application, PULP provides a structural skeleton which is enriched with behavior and styling. In the end, a fully working interactive application is generated.

2 Related work

Different techniques exist to *author* dynamic HTML5 content: on the one hand programming techniques based on either front-end scripting or domain-specific languages with code generation, and on the other hand there are light-weight prototyping techniques based on image slices that are annotated with links to support page transitions. Examples for the first techniques which involve modeling are the WebDSL and Mobl languages[2, 3].

While WebDSL enables the development of generic web applications with a DSL, the Mobl language focuses on mobile application development. Both DSLs are supported by generators that ease quick iterations by generating respective HTML and JavaScript artifacts on-the-fly. The generated artifacts assemble UI elements and add interaction possibilities. The second technique of visual rapid prototyping is based on image slices that are assembled to form the screens of the application, and their annotation to allow for limited interaction support[5, 4]. PULP's usage domain, however, is situated in-between: the focus is on applications without predefined interface elements but with image objects that serve as actors which can be manipulated and which structure the user experience. PULP aims at designing for (novel) interfaces with touch and gesture support, as well as native sound and video output. At the same time, PULP is a textual DSL and allows for more control over the application structure, collaborative working and version control.

3 The DSL

The design of an appropriate domain-specific language PULP for the purpose of rapid prototyping of interactive (touch-based) applications needs to involve

concepts of the domain as first-class citizens, but also the user of such a language having a distinct set of skills. The language concepts are explained in the following, before turning towards the necessary skills to master the language.

3.1 Language concepts

As mentioned before, the users of the PULP language are designers and content creators for different interactive media. They are trained, sometimes even formally, to think in terms of canvases and objects within. Behind this is the notion of coordinate systems and pixels as the basic measure of distance within such systems. Often, these users have good understanding of the size of screen objects and can reliably estimate the pixel size of objects or distances. The PULP DSL draws on these skills as all screen objects and the screens are measured and positioned according to pixel counts. For the sake of simplicity, only absolute positioning is used as relative positioning of elements could potentially lead to obscure dependencies between screen objects. The second important skill of language users is the ability to not only position elements in 2D space, but also according to a z-index, i.e. the notion of screen layers. The language supports this by first the straightforward convention to place objects in order of appearance, and second by layers which may group objects on the same (layer) z-index.

A PULP script contains essentially a number of screens, therein, screen objects, and definitions of interactions upon these screen objects. These interactions are expressed as action chains. The language was designed such that the number of structural tokens such as "=>" or "{" and "}" is kept to a minimum, whereas operative (or more verbal) keywords have a direct counter part in observable functionality like "show", "hide", "move", and "play". The general terminology originally aligned with the analogy of a theater stage with background, foreground, actors and requisites. It later evolved to a more open theme of screens, screen objects, and interactions.

3.2 Screen primitives

An application in PULP is first defined with a few global constants such as "title", "width", "height" and background color. Furthermore it is structured into different *screens* which can contain screen objects and associated interactions. Screen objects are divided into layers, images, and text. Layers enable grouping other screen objects and expressing a z-order on the screen. Images and text are defined with coordinates and size parameters, as well as optional styling (according to CSS3). In addition to screens, different media objects can be defined in a PULP application: videos, sounds, and voices. The difference between sounds and voices is the convention that sounds can be played in parallel, whereas voices are played exclusively to ensure understandability. Finally, a start screen is given for starting up the application.

3.3 Action chains

An action chain consists of at least one action trigger and one action result. More than one action trigger define either an ordered sequence of actions that have to be performed by the user to reach an action result, or an arbitrary sequence of actions that are needed to trigger the action result. In general, sets of combined action triggers are useful for setting with *scripted* behavior that is expected from the user, such as learning environments, whereas single actions triggers are incorporated most for user interface prototypes that strive for simplicity and good usability.

An action chain is defined as:

```
Trigger (=> Trigger)* => Result (=> Result)*
```

An example is a click action on a predefined image that triggers sound playback followed by a screen change. Note that the image defines "click" as a possible action:

```
image showResults (images/btnShowResults.png) width 200 height 40 click
sound buttonSound (media/btnSound.mp3)
click showResults => play buttonSound => screen resultScreen
```

Multiple triggers within an action denote a simple form of a state machine: the results are only triggered in case all specified triggers are performed in the right order by the user. An example is entering a code ("4321") on a graphical numeric pad to gain access to the result screen:

```
image numPad1 (images/buttons/numPad1.png) click
image numPad2 (images/buttons/numPad2.png) click
...
click numPad4 => click numPad3
               => click numPad2 => click numPad1 => screen resultScreen
```

This last aspect of action chains enables to quickly prototype interactive applications with complex interaction patterns that would require larger efforts to achieve with manual JavaScript programming. PULP hides this complexity, and instead presents a designer-friendly, more application focused view on HTML5 scripting.

4 Tool implementation and the HTML5 generator

The PULP editor tool (cf. Fig. 1) is based on the Eclipse platform[1], and makes use of the Xtext domain-specific language toolkit[2] to provide a textual editor with syntax coloring, auto-completion, and validation. The generation step is a model-to-text generation of HTML and JavaScript code and it is enabled by the Xtend2[3] tools. The editor allows browsing image and media files while creating PULP language elements.

[1] Eclipse: http://eclipse.org
[2] Xtext: http://eclipse.org/Xtext
[3] a part of Eclipse Xtext

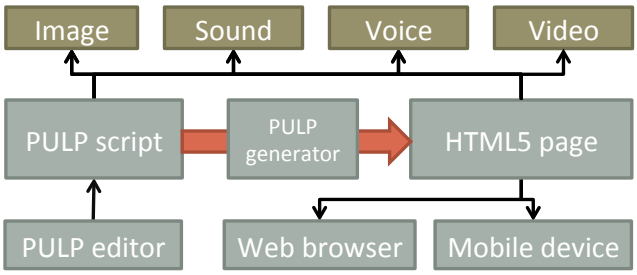

Fig. 1: Architecture of PULP tool chain with media and script artifacts

When a PULP file is saved in the editor, the PULP generator (for HTML5) runs automatically and creates a hierarchy of files in an `/html/` subfolder where also the images and media files reside. The generator creates a single HTML file which first contains the declaration of all screens and screens object as HTML `<div>`s and images, then the associated media files as HTML5 `<audio>` and `<video>` objects. Finally, JavaScript is generated according to the given interaction capabilities and defined interaction of screen objects. The generated JavaScript makes use of jQuery[4] and a few custom library functions.

5 Evaluation

A short informal evaluation of the tool has been carried out. In general, the possibility to achieve small games with comparatively rich interaction and touch gestures was received positively. During the evaluation, the participants were presented first with an exemplary PULP script that demonstrates most of the functionality and uses few pictures and media files to script an application. Different items of the PULP script were explained and participants could ask questions if necessary. This step took at most 15 minutes. In the next evaluation phase the participants were asked to change minor parts of the PULP script, such as positioning of images, and styling of text. Subsequently, the focus shifted to modifying the interactions that were attached to the screen objects, e.g. changing the click action on an image to a mouse-over action and triggering different sounds after a drag and drop action. This second phase took at most 20 minutes. In the third and last phase, participants were asked to be creative with their own material that they brought to the evaluation. This last phase did not involve extensive coaching anymore, and took about 25 minutes.

A more formal follow-up study is planned and will be carried out with a later version of the tool. Interesting aspects of this evaluation will concern the creation of more complex control flows, visual positioning of elements, and embedding videos. Furthermore, tool iterations based on user feedback are expected to motivate not only functional improvements, but also improvements of language syntax and semantics.

[4] jQuery: http://jquery.org

6 Conclusions

This paper introduces a novel domain-specific language (DSL) for design and content authoring stakeholders in the domain of mobile games and other interactive applications. The potential range of applications is larger than shown in this paper, however, the gaming domain was chosen due to its focus and availability of professional users and testers. The emphasis is on enabling to script interaction flows, gestures and behavior in general. Combined with appealing visuals and supportive media such as object sounds and ambient atmospheres (potentially also video), applications created with PULP can convey a user experience that adequately matches the final product in its main lines. This rapid prototyped experience is important for showing the product vision as a clear goal during development, but also for demo purposes and early user feedback.

The presented DSL aims at simplifying programming tasks that would normally involve parallel and event-based programming paradigms, gesture handling, and handling of browser compatibility issues. The associated generator component takes a PULP file and linked media files as input and generates an entire HTML5 web application, which can be directly tested with a web browser.

Behind the PULP language and tools is the question whether DSLs can be applied successfully in the domain of design where the use of textual tools is not as common. DSLs naturally relate to modeling and formalization - therefore the question is extended to the applicability of (more or less formal) modeling in the design domain. The research on the PULP language aims at investigating this problem domain in the future. Further future work can be also derived from the evaluation, namely to ease the positioning of screen objects and to allow for prototyping other, more elaborate user interfaces that make use of gestures. Finally, prototyping efforts should be aligned with means to observe use flows[1] for the collection of user feedback during early stages of development.

References

1. Mathias Funk, Piet van der Putten, and Henk Corporaal. Analytics for the internet of things. In *CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, pages 4195–4200, New York, NY, USA, 2009. ACM.
2. Danny M. Groenewegen, Zef Hemel, Lennart C. L. Kats, and Eelco Visser. Webdsl: a domain-specific language for dynamic web applications. In Gail E. Harris, editor, *Companion to the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2008, October 19-13, 2007, Nashville, TN, USA*, pages 779–780. ACM, 2008.
3. Zef Hemel and Eelco Visser. Declaratively programming the mobile web with mobl. In *Proceedings of the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2011*, Portland, Oregon, USA, 2011. ACM.
4. Ali Hosseini-Khayat, Teddy Seyed, Chris Burns, and Frank Maurer. Low-fidelity prototyping of gesture-based applications. In *Proceedings of the 3rd ACM SIGCHI*

symposium on Engineering interactive computing systems, EICS '11, pages 289–294, New York, NY, USA, 2011. ACM.

5. Anders P. Jørgensen, Matthijs Collard, and Christian Koch. Prototyping iphone apps: realistic experiences on the device. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, NordiCHI '10, pages 687–690, New York, NY, USA, 2010. ACM.