Gummy for Multi-Platform User Interface Designs: Shape me, Multiply me, Fix me, Use me

Jan Meskens Jo Vermeulen Kris Luyten Karin Coninx

Hasselt University – tUL – IBBT
Expertise Centre for Digital Media
Wetenschapspark 2, B-3590 Diepenbeek, Belgium
{jan.meskens,jo.vermeulen,kris.luyten,karin.coninx}@uhasselt.be

ABSTRACT

Designers still often create a specific user interface for every target platform they wish to support, which is time-consuming and error-prone. The need for a multi-platform user interface design approach that designers feel comfortable with increases as people expect their applications and data to go where they go. We present Gummy, a multi-platform graphical user interface builder that can generate an initial design for a new platform by adapting and combining features of existing user interfaces created for the same application. Our approach makes it easy to target new platforms and keep all user interfaces consistent without requiring designers to considerably change their work practice.

Categories and Subject Descriptors

H.5.2 [**Information interfaces and presentation**]: User Interfaces – Graphical user interfaces, Prototyping

Keywords

design tools, multi-platform design, GUI builder, UIML

1. INTRODUCTION

There is an increasing need for applications that are available on multiple devices. Today, people tend to read their email or browse the web using their mobile phones or game consoles. This tendency will increase with the move towards ubiquitous computing where users are supposed to have seamless access to applications regardless of their whereabouts or the computing device at hand [20]. People need more means to access their information and applications than just a regular desktop computer. Applications that need to be available on any device at the user's disposal should be able to deploy a suitable user interface (UI) on each of these computing platforms.

We define a *computing platform* as the combination of a hardware device, an operating system and user interface toolkit. Designing a user interface for different computing platforms is far from simple. Each computing platform has its own characteristics such as the device's form factor, the appropriate interaction metaphors and the supported user interface toolkit. In current practice, designers often create a specific user interface for *every* target platform. Even though some technologies are shared between a number of devices (e.g. Java ME[1] or a modern web browser), usually each device still requires specific adjustment. There is a high cost incurred in adding a new target device and keeping all user interfaces consistent using manual approaches. Furthermore, there is no clear separation between the user interface and the underlying application logic.

A common solution to these issues is to specify the user interface in an abstract way by means of high-level models such as task models and dialogue models [9]. The platform-specific user interfaces are then generated automatically from this abstract description. The user interface has to be specified only once, which makes it easier to make changes or add a new target platform. In spite of the fact that these tools solve most of the problems with the manual approach, the resulting user interfaces usually still lack the aesthetic quality of a manually designed interface. Furthermore, the design process is not intuitive since designers have to master a new language to specify the high-level models and cannot accurately predict what the resulting user interface will look like [18].

Gummy combines the benefits of both the manual approach and model-based techniques. Designers create and perform prototyping of a multi-platform graphical user interface (GUI) in the same way as they do when dealing with traditional GUI builders such as Microsoft Visual Studio[2] and Netbeans[3]. Gummy builds a platform-independent representation of the user interface and updates it as the designer makes changes. This allows for an abstract specification of the user interface while keeping the design process intuitive and familiar. An abstract user interface specification avoids a tight interweaving of application and presentation logic.

Gummy can generate an initial design for a new platform from existing user interfaces created for the same applica-

[1] http://java.sun.com/javame/
[2] http://msdn.microsoft.com/vstudio/
[3] http://www.netbeans.org/

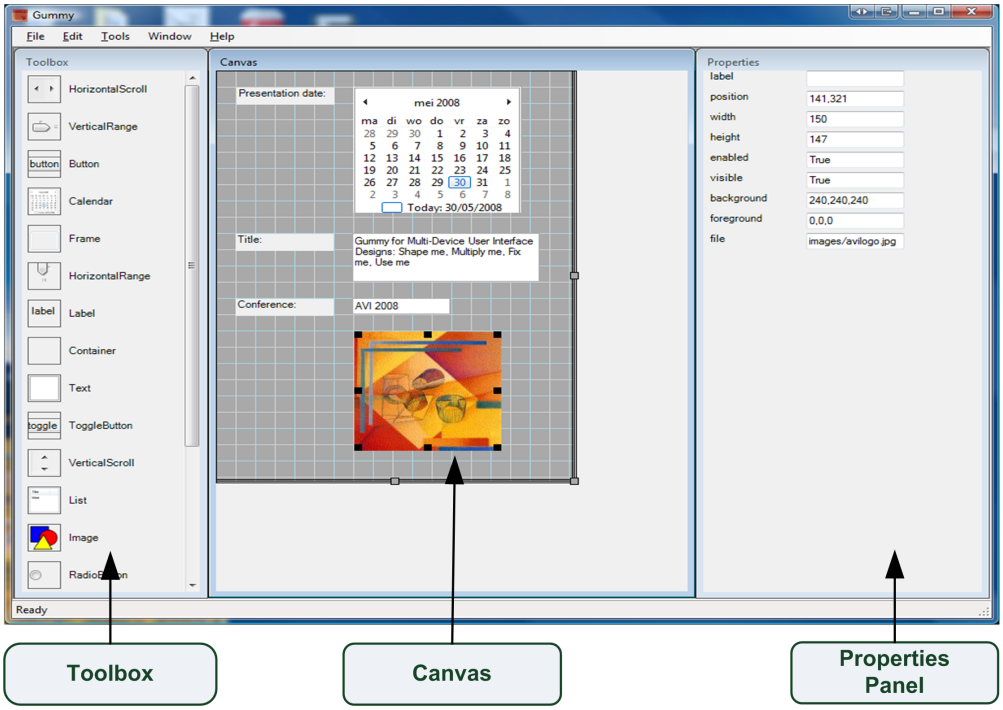

Figure 1: The three main dialogues of the Gummy tool

tion but for other platforms. This makes it it easy to target new platforms and keep all user interfaces consistent without requiring designers to considerably change their work practice. Since designers work with the concrete user interface, GUMMY allows for a true WYSIWYG[4] multi-platform user interface design process.

The main contributions we present in this paper are:

- a *multi-platform design approach* to creating user interfaces incrementally for a wide range of computing platforms while still working on a concrete level (Sect. 3).
- GUMMY, a *generic multi-platform GUI builder* to support the aforementioned design approach. This tool automatically adapts its workspace according to the considered target platform (Sect. 4). A preliminary user study with GUMMY indicated that the incremental design approach was faster than starting from scratch for each computing platform (Sect. 5).

2. BACKGROUND AND MOTIVATION

We feel that most designers prefer to have a concrete representation during their design activities, whether they are working on a single or a multi-platform user interface. This avoids their having to imagine what the final user interface would look like. We can thus conclude that working on a concrete representation reduces the mental burden on the designer [2, 18].

We structured GUMMY in a similar way to traditional GUI builders in order to allow designers to reuse their knowledge of single-platform user interface design tools. Fig. 1 shows the main components of the GUMMY interface: there is a *toolbox* showing the available user interface elements, a *canvas* to build the actual user interface and a *properties panel* to change the properties of the user interface elements on the canvas.

The resemblance to traditional GUI builders also reflects one of the main strengths of our approach: the abstractions used to support multi-platform user interface design are carefully hidden from the designer. This is contrary to existing multi-platform design tools that often expose these abstractions to the designer. In the remainder of this section we give some details about the underlying abstract language that GUMMY uses to represent multi-platform user interfaces and how they are presented to the designer.

The underlying language used in GUMMY is the User Interface Markup Language (UIML) [14], an XML language that contains a platform-independent *user interface description* on the one hand and a *mapping vocabulary* on the other hand. While the former is used to describe the structure, style, content and behaviour of a user interface using platform-independent terms, the latter contains mappings of these terms onto concrete widgets. Fig. 2 gives an example of a mapping vocabulary. The traditional rendering step is to translate the platform-independent user interface description into a concrete user interface using these mappings. For the implementation of GUMMY the opposite was done: the concrete representations were used in the tool and were internally mapped onto the associated abstractions. The designer works with the concrete graphical representations, avoiding the XML language, while the tool maintains a synchronised platform-independent UIML document for

[4]What You See Is What You Get

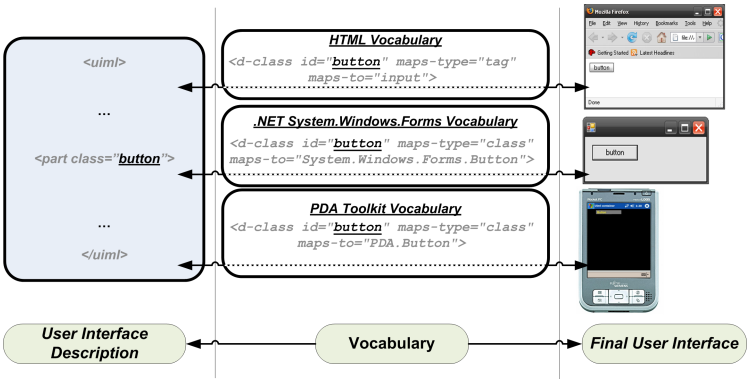

Figure 2: A UIML vocabulary relates generic terms to concrete representations

the concrete design.

3. MULTI-PLATFORM DESIGN APPROACH

Now to delve deeper into the design process supported by GUMMY, as depicted in Fig. 3. The design process is similar to that of traditional GUI builder tools, but includes an additional iteration to refine a design for a specific computing platform.

The procedure to create a user interface design for different platforms can be described in the following five steps:

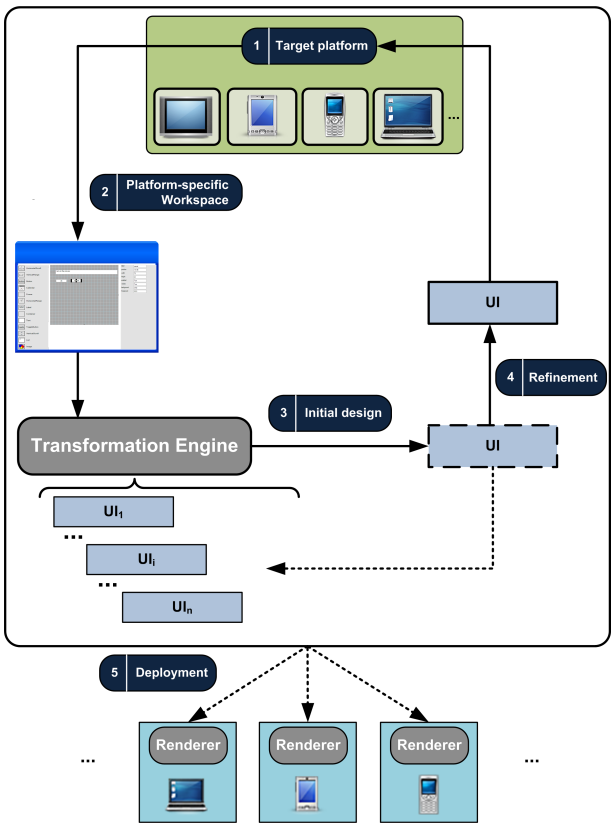

Figure 3: The approach presented in this paper to design multi-platform user interfaces

1. As a first step, the user interface designer specifies the *target platform* for which they want to design a user interface. Some possible platforms are a mobile phone with the Compact .NET framework, a digital TV with DVB-MHP, etc.

2. According to the specified platform, GUMMY automatically loads a GUI builder workspace that is fully equipped for designing user interfaces for this platform.

3. From a set of existing user interfaces that are all created for the same application, GUMMY automatically generates an initial design for the selected target platform. For this generation process, GUMMY relies on a *transformation engine* component. When there are no existing user interfaces available, this step generates an empty user interface for the selected platform. The specifics of the underlying algorithm to perform this transformation is not the main focus of this paper.

4. The designer can refine the initial user interface until it fits their vision. The resulting design is then added to the set of existing user interface designs where it may serve as an input for the transformation engine. After this step, a new iteration is started.

5. When the designer is finished, GUMMY exports all designs as one or more UIML user interface descriptions. Notice that platform-specific user interface descriptions might be needed to achieve aesthetic quality on every target platform. These UIML descriptions can then be rendered on the target platforms. In this paper a *UIML renderer* is defined as a component that can transform a UIML description into a working user interface.

4. A GENERIC MULTI-PLATFORM GUI BUILDER

Three aspects of GUMMY's architecture account for its generic nature:

- a *pluggable rendering architecture* which makes it possible to integrate any UIML renderer into GUMMY with minimal changes;
- UIML vocabularies to automatically *adapt* GUMMY*'s workspace* to a certain platform;

- a *transformation engine* that generates initial designs for new platforms based on existing designs for other platforms.

4.1 Pluggable Rendering Architecture

When a designer alters a user interface design in GUMMY, the underlying UIML description is automatically updated and visual feedback is provided immediately. GUMMY relies on an external UIML renderer to provide the visual representation of this UIML description. Different UIML renderers can be integrated into the tool as plugins.

The communication between GUMMY and a UIML renderer can be viewed as a set of inputs from GUMMY to the renderer on the one hand and a set of outputs from the renderer to GUMMY on the other hand. GUMMY will feed UIML descriptions of parts of the user interface into the renderer. In turn, the renderer will parse these UIML descriptions and render them as off-screen bitmaps. GUMMY then uses these bitmaps to visualise the underlying UIML descriptions.

In theory, every renderer that respects the communication protocol described above can be plugged into GUMMY. However, renderers might be written in other programming languages, run only on specific operating systems (e.g. embedded systems) or have limited communication possibilities. It would be inflexible to require GUMMY to know how to communicate with each of these renderers. To solve this problem, an additional layer of abstraction was added between GUMMY and the different UIML renderers. *Proxy objects* act as local placeholders for the UIML renderers and hide the communication details from GUMMY, as shown in Fig. 4. Every proxy object behaves like a regular UIML renderer but just forwards the rendering inputs to an actual renderer which can be located on any computing device. In turn, the bitmaps produced by the renderer are sent back to the proxy which finally delivers them to the design environment. Proxy objects are free to choose how they communicate with their UIML renderer, e.g. through socket communication, SOAP, etc. GUMMY's proxy objects are based on the *Remote Proxy* and *Adapter* design patterns [13].

4.2 Adapting the Gummy Workspace

In the GUMMY tool, the designer needs to specify the platform for which they want to design a user interface. GUMMY then automatically loads a UIML vocabulary designed for the selected platform. At the same time, it looks for a suitable UIML renderer for this vocabulary. In order to find a suitable renderer, all the available proxy objects need to be placed in a predefined location together with a configuration file that connects each renderer to a set of vocabularies that it can handle.

Once the vocabulary and renderer are loaded, GUMMY adapts its workspace to be fully ready for designing user interfaces for the selected platform. This platform-specific workspace will have a toolbox dialogue that contains only those user interface elements that are available for the selected platform. In order to generate this toolbox automatically, GUMMY uses the relation between generic terms and concrete user interface elements (see Fig. 2) described in the loaded UIML vocabulary. Small versions of all the concrete widgets that are described in the vocabulary are displayed as items in the

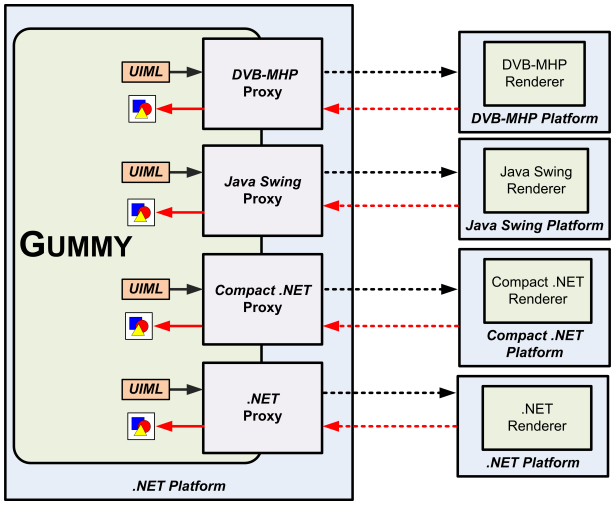

Figure 4: Existing UIML renderers can be easily integrated into Gummy using *proxy objects*

toolbox. The designer can then drag and drop these items onto the canvas. While designers manipulate concrete user interface representations on the canvas, GUMMY maintains a UIML description of the user interface in the background. Every time a designer repositions or resizes a widget through direct manipulation or modifies a property of a widget in the properties panel, the corresponding UIML description is updated and forwarded to the external renderer to update the view.

4.3 The Transformation Engine

GUMMY allows the designer to create several user interface designs for the same application. Each design corresponds to a specific target platform. During this process, a transformation engine is used to generate initial designs for new platforms based on the previously designed user interfaces. Designers can also simply copy one of the previous designs.

To prove the design approach that was introduced in Sect. 3, a basic transformation engine was integrated into GUMMY. This engine transforms existing designs into a new design based on the available screen size, as shown graphically in Fig. 5: Two existing interface designs I_1 and I_2, both representing the same application (a card game) but for two specific screen sizes, are used by the transformation engine to generate an initial design for a new screen size I_x.

The underlying algorithm used by this engine will not be discussed into detail since it is not the main focus of this paper. The transformation engine uses a set of rules to decide which properties (e.g. a widget's size or position) of the already created user interfaces should be selected and adapted according to the available screen size on the target platform. Each rule relates a property with a minimum and maximum screen size between which it is valid. For each property, the designer specifies these rules by manipulating a set of sliders that appear next to the design canvas as shown in Fig. 6. These sliders represent the vertical and horizontal screen size extrema within which the selected property is valid. Specifying these rules from scratch is a time-consuming activity.

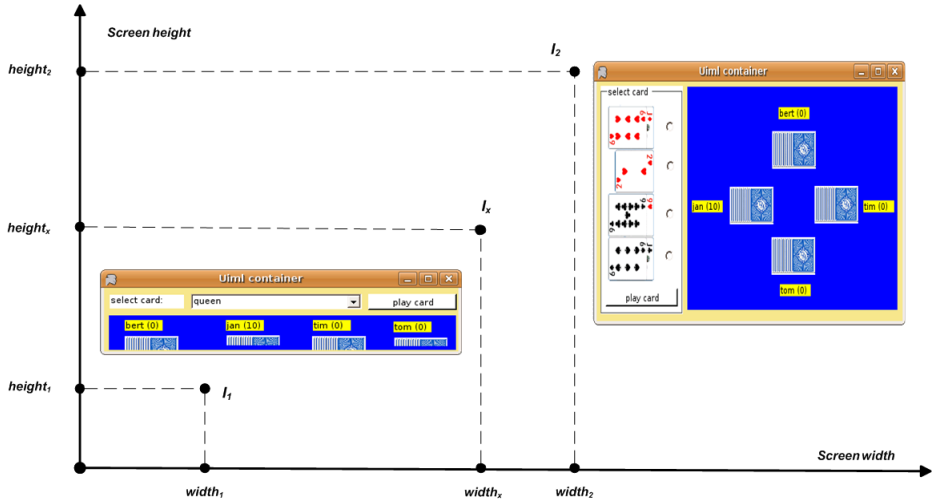

Figure 5: An initial interface for screen space I_x can be generated with the rule-based transformation engine.

Therefore, GUMMY automatically generates initial rules by using a heuristic based on the assumption that a component may only be displayed when it fits within the available screen space. The conceptual user study indicated that designers were faster when using the initial designs generated by this engine than starting from scratch for each computing platform (see Sect. 5.1).

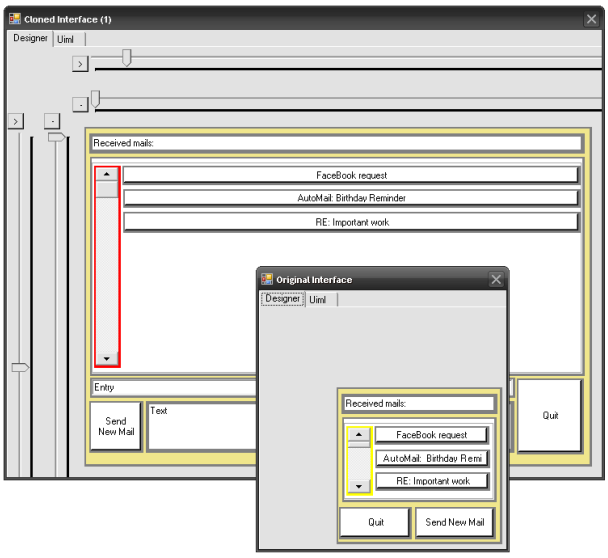

Figure 6: Two mail client user interfaces which are used as input for the rule based transformation engine

5. ANALYSIS

5.1 Conceptual User Study

To get an idea about the usability of the approach a small experiment was organized to assess the user's effectiveness at creating a user interface for multiple computing platforms with GUMMY. Ten test participants with various computer skills were recruited. Six of them were colleagues with good programming skills and a lot of experience with traditional GUI design tools. Four participants did not have a computer science background, of whom three did have experience with graphical drawing tools. The diverse test audience allowed examination of the question of whether the tool would require a certain technical way of thinking. In order to instruct all subjects in the same way before they started, they were provided with a written tutorial explaining the basic workings of our tool.

The test consisted of two parts. The first part of the test evaluated the ease of starting from an initial design for a new platform versus creating one from scratch. The test participants were divided into two groups with the same proportion of technical and non-technical people. Both groups had to arrive at a predefined user interface for a new platform. The first group had to create this user interface from scratch whereas the second one was allowed to base their user interface on the initial design generated by the transformation engine. By performing a one-way analysis of variance, it was determined that the subjects who were able to use the initial design were significantly faster in obtaining the desired user interface than the members of the other group ($F_{1,8} = 15.935, p < 0.005$).

In the second part of the experiment, the participants were asked to manipulate the transformation rules in order to obtain a predefined initial design for a new platform. Subjects rated the difficulty of this assignment on a *Likert* scale from *very easy* to *very hard*. A *Spearman rho* analysis indicated that there was a negative correlation between the programming experience of the test subjects and the perceived difficulty of the task ($p < 0.05$). This suggests that customising transformation rules (see Sect. 4.3) is not very intuitive for non-programmers. However, this does not invalidate the multi-platform design approach presented in this paper. The first part of the experiment gave an indication that changes to the transformation rules were usually not necessary. A more intuitive way of specifying transformations could resolve this issue.

5.2 Evaluation of Effectiveness

As was pointed out by Olsen [6], usability testing in its traditional form is rarely suitable for evaluating UI architectures, toolkits and design tools. Olsen argues that the three basic assumptions of usability testing, (1) minimal required training; (2) a standardised task to compare; and (3) being able to finish a test in short period of time, are rarely met by these systems. For GUMMY, at least the first two assumptions are not met. Because of this, the evaluation was extended with an analysis based on Olsen's evaluation framework.

This framework uses a number of attributes of good tools and methods to demonstrate that a particular tool supports them. The ones considered for GUMMY are:

- *reduce solution viscosity* with flexibility and expressive match;
- simplify interconnection and allow easy combinations to achieve *power in combination*;

5.2.1 Reduce solution viscosity

This implies that a good tool should foster good design by reducing the effort required to iterate on many possible solutions [6].

A tool is *flexible* if it allows the making of rapid design changes that can then be evaluated by users [6]. Since GUMMY allows designers to work on a concrete level they can easily make changes to the user interface using direct manipulation. These changes can be tested immediately by instructing GUMMY to deploy them to the appropriate renderer. Initial designs for new target platforms can be automatically generated using the transformation engine (see Sect. 4.3). These initial designs can again be easily modified (e.g. widgets can be moved, resized, deleted or remapped to another concrete widget) after which the changes can be evaluated. If necessary, designers can easily intervene and correct the tool.

The manual approach offers roughly the same benefits but only within the visual design tool for one specific platform. Designs for other platforms can neither be quickly created nor changed since they have to be recreated from scratch. While most model-based design tools support flexibility by allowing changes to the models and evaluation of the result, these changes take more effort than with GUMMY. For instance, while designers could remap widgets in these tools by altering the transformation model, selecting an alternative widget through direct manipulation is much easier. Due to its better expressive match, GUMMY requires less effort from the designer to intervene.

Expressive match is an estimate of how close the means of expressing design choices are to the problem being solved [6]. GUMMY allows designers to create a user interface for different platforms in much the same way as they do with single-platform visual design tools. A visual design tool is a better expressive match for the task of designing a (multi-platform) user interface than a tool to manipulate abstract user interface models. With model-based techniques, the connection between the abstract models and the resulting user interface is often not clear to the designer [18]. Thus, being able to visually design multi-platform user interfaces lowers designers' skill barrier.

5.2.2 Power in combination

Power in combination refers to a common infrastructure that can support new components to create new solutions [6]. This can be supported mainly by simplifying interconnections and by ease of combination. GUMMY accomplishes both.

Simplifying interconnections deals with reducing the cost of introducing a new component from N (connect to every other component) to 1 (just implement a standard interface) [6]. As we explained in Sect. 4, GUMMY can use any combination of vocabulary and UIML renderer. Every renderer just needs to supply a proxy object that implements a common programming interface in order to communicate with GUMMY. Traditional GUI builders and existing multi-platform design tools usually support only a fixed set of platforms. Adding a new platform to one of these tools often requires specific changes to its internals.

Ease of combination refers to the fact that it is usually not sufficient to be able to connect different components [6]. The connection should also be simple and straightforward. This is clearly the case here. GUMMY only requires renderers to provide a proxy object that conforms to a simple programming interface. We were able to integrate the Uiml.net renderer as well as renderers for Java ME and DVB-MHP without much effort.

6. RELATED WORK

Model-based and automatic techniques have been frequently used for multi-platform user interface design [9]. This approach requires designers to define a high-level specification of the user interface which is then used to automatically produce an appropriate user interface for each target platform. Two examples of tools that rely on this technique are MOBI-D [19] and Dygimes [4]. One of the major drawbacks of this type of tools is that the design process is not intuitive for designers. As discussed in the previous section, GUMMY does not exhibit this problem.

Other tools that try to facilitate the design of multi-platform user interfaces have traditionally focused on low-fidelity or medium-fidelity prototypes. Notable examples include Damask [15] and SketchiXML [5]. Damask lets designers sketch a user interface for one device and indicate the design patterns the interface uses. From this initial design Damask will then automatically generate the other device-specific user interfaces. Although GUMMY and Damask share many concepts (e.g. building an abstract model in the background, generating initial designs which can be refined later, etc.), it is possible to conclude that they serve different purposes. While Damask is mainly targeted towards prototyping, the designs that are created with GUMMY can be directly used as the final user interface and coupled to existing application logic [17]. Recently, Damask was extended with the concept of *layers* for managing consistency between designs for different computing platforms [16]. The motivation for this was a survey among designers that identified consistency as one of the major burdens for cross-device user interface de-

sign. It would be interesting to examine if layers could be used within Gummy to propagate changes between designs for different platforms.

SketchiXML [5] creates an abstract user interface specification from a user interface sketch that can then be deployed on multiple devices. However, the tool has a limited set of abstractions that designers can employ to design a user interface. It builds upon the UsiXML language to describe the abstract user interface which has a predefined set of abstract widgets. With Gummy, the set of abstractions is defined externally in a UIML vocabulary and thus can be changed at any time. SketchiXML has recently been extended to support the entire range of prototype fidelities [5]. After the user evaluation, most participants indicated a preference for medium- or high-fidelity prototyping. This reflects our belief: We feel that most designers prefer to have a concrete representation available during their design activities.

Collignon, Vanderdonckt and Calvary [3] describe an interesting visual tool to specify *plasticity domains* for user interfaces. A plasticity domain defines a range of contexts of use for which a user interface is valid (e.g. a mobile phone and a PDA). Their tool can embed several UIs corresponding to different platforms into one running application. If the context of use changes, the most appropriate of these UIs is automatically selected and used. Contrary to the present approach, this work does not facilitate the design of multi-platform user interfaces. Instead, they focus on defining possible transitions between user interfaces for different platforms and on exploiting these transitions at runtime. The different designs still have to be created by hand.

In the Gummy tool, a basic transformation engine is used that is able to generate an initial design for a new computing platform depending on the available screen size. It was not an aim of the present work to contribute to developments in this area but the transformation engine was implemented to prove the utility of the design approach presented in this paper. Similar but more sophisticated techniques for adapting user interfaces include Supple [11], splitting rules for graceful degradation [10] and Artistic Resizing [7]. Each of these techniques require other types of input to steer the adaptation and are solely used at runtime. It will be interesting to explore these and other transformation algorithms that take into account a more general notion of context than just the available screen size (e.g. different interaction techniques and input devices such as pinch zooming on a multi-touch display). Supple [11] looks promising as it already has basic support for adapting to interaction techniques (e.g. making user interface elements larger to ease interaction on a touch screen), and to the user's preferences and abilities [12]. Modelling input devices and interaction techniques [1, 8] might be useful to cope with the differences between computing platforms and to manage overall consistency.

7. DISCUSSION

This paper presented Gummy, a multi-platform GUI builder that allows designers to easily target new computing platforms without having to give up their current work practices. As designers work on the final user interface, Gummy builds up a corresponding UIML description. The additional abstraction provided by this UIML description allows Gummy to generate initial designs for new platforms based on existing user interfaces created for the same application. Gummy combines many of the advantages of existing multi-platform design tools with those of traditional GUI builders. This design approach lowers the skill barrier to multi-platform user interface design by allowing designers to easily intervene in the process and reuse their existing knowledge of single-platform design tools. Gummy's architecture is flexible enough to easily integrate a wide range of computing platforms and UIML renderers.

We feel that our work opens up interesting possibilities for further research. In particular, the process of empowering domain experts to design user interfaces with Gummy will be examined. Existing tools are not tailored toward non-technical domain experts, even though their input is very important during the design process and helps to shape the final user interface. To provide support for domain experts, the Gummy workspace should not only take into account the target computing platform but also the domain for which the user interface will be designed. UIML vocabularies allow us to do this [14].

At the moment, Gummy does not explicitly enforce consistency. Consistency is only ensured between an initial design and the existing designs it was generated from. Although this is sufficient in most cases, it does not scale well to widely varying computing platforms. Sometimes breaking consistency is desirable because of the specific nature of the target platform (e.g. excluding labels due to space constraints). In the future, Gummy should allow designers to control consistency between computing platforms at a high level of granularity (e.g. exclude labels for both PDAs and mobile phones, but include them for other platforms). As mentioned in the discussion on related work (Sect. 6), Damask's concept of layers [16] in a modified form might be a good way to realise this.

Gummy lacks a number of features that are crucial for its applicability to real-world problems. For one, only user interfaces with a single screen can be designed. Support for multiple dialogues would allow designers to create complex multi-platform user interfaces with the tool. The dialogue flow should be kept consistent when dialogues are split for certain platforms and merged for others. Furthermore, designers currently have no way of specifying how a user interface should behave when it is resized at runtime. The main challenge here is to integrate different platform-specific layout managers in a generic way.

We are aware of the limitations of our current rule-based transformation engine (Sect. 5). Since the focus is mainly on an intuitive multi-platform user interface design approach, our efforts will not concentrate on developing a more advanced transformation engine. Instead, an attempt will be made to extend the Gummy tool to enable the integration of multiple transformation engines. This would allow the use of existing, more sophisticated techniques such as Supple [11], Artistic Resizing [7] or graceful degradation [10]. Designers would then be able to try out several transformation engines to generate an initial design and pick the one they like best. As mentioned in Sect. 6, it is necessary to investigate transformation algorithms that take into account

more than just the available screen size. Interaction modelling approaches [1, 8] might be used to tackle this problem while still preserving the generality of our approach.

More information on GUMMY and an executable of the tool can be found at `http://research.edm.uhasselt.be/~gummy/`.

Acknowledgments

This paper would not have been what it is today without the help of the other researchers at EDM. We warmly thank everyone who helped us test the tool and provided useful insights during the writing of this paper. Part of the research at EDM is funded by ERDF (European Regional Development Fund) and the Flemish Government. The AMASS++ (Advanced Multimedia Alignment and Structured Summarization) project IWT 060051 is directly funded by the IWT (Flemish subsidy organization).

8. REFERENCES

[1] Renaud Blanch and Michel Beaudouin-Lafon. Programming rich interactions using the hierarchical state machine toolkit. In *Proceedings of AVI '06*, pages 51–58, New York, NY, USA, 2006. ACM.

[2] Luca Cardelli. Building user interfaces by direct manipulation. In *Proceedings of UIST '88*, pages 152–166, New York, NY, USA, 1988. ACM.

[3] Bernoît Collignon, Jean Vanderdonckt, and Gaëlle Calvary. An intelligent editor for multi-presentation user interfaces. In *Proceedings of SAC '08*, New York, NY, USA, 2008. ACM.

[4] Karin Coninx, Kris Luyten, Chris Vandervelpen, Jan Van den Bergh, and Bert Creemers. Dygimes: Dynamically generating interfaces for mobile computing devices and embedded systems. In *Mobile HCI*, volume 2795 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2003.

[5] Adrien Coyette, Suzanne Kieffer, and Jean Vanderdonckt. Multi-fidelity prototyping of user interfaces. In *Proceedings of INTERACT '07*, volume 4662 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2007.

[6] Jr. Dan R. Olsen. Evaluating user interface systems research. In *Proceedings of UIST '07*, pages 251–258, New York, NY, USA, 2007. ACM.

[7] Pierre Dragicevic, Stéphane Chatty, David Thevenin, and Jean-Luc Vinot. Artistic resizing: a technique for rich scale-sensitive vector graphics. In *Proceedings of UIST '05*, pages 201–210, New York, NY, USA, 2005. ACM.

[8] Pierre Dragicevic and Jean-Daniel Fekete. Support for input adaptability in the icon toolkit. In *Proceedings of ICMI '04*, pages 212–219, New York, NY, USA, 2004. ACM.

[9] Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta. Applying model-based techniques to the development of uis for mobile computers. In *Proceedings of IUI '01*, pages 69–76, New York, NY, USA, 2001. ACM.

[10] Murielle Florins, Francisco Montero Simarro, Jean Vanderdonckt, and Benjamin Michotte. Splitting rules for graceful degradation of user interfaces. In *Proceedings of AVI '06*, pages 59–66, New York, NY, USA, 2006. ACM.

[11] Krzysztof Gajos and Daniel S. Weld. Supple: automatically generating user interfaces. In *Proceedings of IUI '04*, pages 93–100, New York, NY, USA, 2004. ACM.

[12] Krzysztof Z. Gajos, Jacob O. Wobbrock, and Daniel S. Weld. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of UIST '07*, pages 231–240, New York, NY, USA, 2007. ACM.

[13] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[14] James Helms and Marc Abrams. Retrospective on ui description languages, based on eight years´ experience with the user interface markup language (uiml). *International Journal of Web Engineering and Technology (IJWET)*, 4(2), 2008. To appear.

[15] James Lin and James A. Landay. Damask: A Tool for Early-Stage Design and Prototyping of Multi-Device User Interfaces. In *Proceedings of DMS '02*, pages 573–580, 2002.

[16] James Lin and James Landay. Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. In *Proceedings of CHI '08*, New York, NY, USA, 2008. ACM. To appear.

[17] Kris Luyten, Kristof Thys, Jo Vermeulen, and Karin Coninx. A generic approach for multi-device user interface rendering with uiml. In *Computer-Aided Design Of User Interfaces V*, pages 175–182. Springer Netherlands, 2007.

[18] Brad Myers, Scott E. Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3–28, 2000.

[19] Angel Puerta and Jacob Eisenstein. Towards a general computational framework for model-based interface development systems. In *Proceedings of IUI '99*, pages 171–178, New York, NY, USA, 1999. ACM.

[20] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, September 1991.