GUI Generation from Wireframes

GUI Generation from Wireframes

Óscar Sánchez Ramón, Jesús García Molina
University of Murcia
Murcia, Spain
{osanchez,jmolina}@um.es

Jesús Sánchez Cuadrado
Autonomous University of Madrid
Madrid, Spain
jesus.sanchez.cuadrado@uam.es

Jean Vanderdonckt
Catholic Universiy of Louvain
Louvain-La-Neuve, Belgium
jean.vanderdonckt@uclouvain.be

ABSTRACT

Wireframes are useful for discussing and refining the user interface of a new application. After the client has validated the GUI, frequently developers have to spend time on recreating the GUI in a development environment for a specific language, and then the created wireframes are discarded. We propose a model-based approach to infer the high level layout of the GUI based on wireframes in order to be able to generate a proper final GUI for different technologies.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

Keywords

Wireframing, Layout inference.

1. INTRODUCTION

Sketching and wireframing are useful as a communication medium between clients and developers in early stages of designing the user interface (UI) of applications. Sketching is rapid, freehand drawing to represent initial design ideas that are not intended of becoming part of a finished product. Wireframing is a visual guide we use to suggest the contents and structure of the views as well as the relationships between those views, which is refined until an agreement is reached. Both allow developers to focus purely on functions and user interactivity, laying aside irrelevant details (such as visual themes). Both techniques are complementary and supported by computer tools. After the refinement phase, the user interface becomes more or less stable. Then, most wireframing tools can generate an interactive mockup which can be used for demos and usability testing.

The mockups created with wireframing tools can serve as a starting point for creating the final user interface. In fact, there are tools[1] that take mockups and generate code for different target technologies and platforms. This relieves developers from spending time on recreating the UI in a development environment. However, the generated code does not always have the desired quality since frequently the implicit graphical relations between the UI elements are neglected.

Our work aims at generating quality code from wireframes, and more specifically, it is focused on the layout inference of the UI. We present a model-driven approach to reverse engineering a wireframe in order to extract implicit relations between the elements and be able to generate a proper UI for different target technologies. The approach has the following benefits: i) the solution is integrated with existing wireframing tools, so developers or event clients can use tools they are used to; ii) the solution can be used with independence of the source (wireframing tool) and target (GUI toolkit); iii) the wireframing tool user does not have to take care of perfect alignments since they are inferred; iv) the target code generated automatically will follow the best practices in engineering.

We have implemented a prototype that supports our approach for WireframeSketcher as a wireframing tool, and Java Swing as a target toolkit.

In the next section we briefly present the related work. Section 3 shows the scenario of use. In Section 4 the layout inference approach is presented, and the prototype that results from this approach is commented in Section 5. The paper finishes with the conclusions and related work.

2. RELATED WORK

Sketching and wireframing techniques have been proved to be useful in early stages of UI design. In [5], the authors show the benefits of sketching for leveraging the user creativity when designing. In [7] it is claimed that sketching allows taking one design idea at a time and then work it out in details. Sketching is also useful to explore a set of alternatives which are narrowed after several iterations [6]. There are other works, like [13] with state that sketching can be useful to detect usability issues.

Though sketching and wireframing can be applied manually, computer-assisted tools are frequently preferred due to their benefits for editing and deleting the created elements. Therefore, there is a large amount of software tools devoted to sketching [9] and wireframing [2].

Given that wireframing tools are used to refine the concepts that come out of sketching, they frequently often some

Actas del XIV Congreso Internacional de Interacción Persona-Ordenador. Celebrado en el Marco del CEDI 2013, del 17-20 de septiembre de 2013 en Madrid, España.
Congreso promovido por AIPO – Asociación para la Interacción Persona-Ordenador.

[1]For example, *Reify* [10] generates code for Balsamiq [2] mockups.

facilities to generate mockups and prototypes. In [4] a tool for creating UIs at different fidelity levels is presented. This tool supports the creation of prototypes and outcomes a UI description in terms of a platform-independent User Interface Description Language (UIDL) called UsiXML. There are available tools to generate a final UI for different platforms based on UsiXML models, such as mobile or web platforms.

As to the field of layout discovery, our work is partly based on the model-driven approach introduced in [12] for inferring the layout of Rapid Application Development environments (RADs). In [11] authors propose an approach to generate a web interface from mockups. In their approach, the user must select the layout type to be used, whereas in our case the layout types are inferred and the best layout arrangement is selected. Another related work in this area is that of presented in [8], which proposes the use a mathematical model based on linear programming, to represent the GUI layout constraints.

3. SCENARIO OF USE

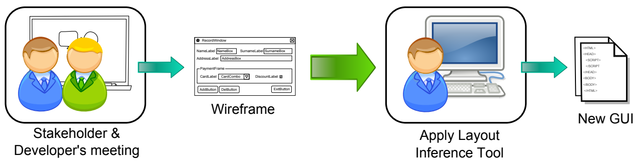

Figure 1: The scenario of use

Figure 1 show the scenario of use of the proposed approach. Firstly, stakeholders and developers have a series of meetings to discuss about some user interface concerns such as the contents to show, the overall layout of the contents, or the interaction required. Developers will iteratively refine the interface with the wireframing tool they take to or they are used to, and after the meetings a validated wireframe will be available.

In this step wireframing tools are expected to be used, thought sketching tools can also be used in an initial phase of brainstorming.

Then, a developer loads the file in the development environment (if the wireframing tool is not already integrated in that enviroment) and it automatically generates the GUI code for a specific target technology. The code can then be manually tuned if needed and it is integrated in the new system.

4. LAYOUT INFERENCE

Figure 2 shows the transformation chain that we apply to obtain a final GUI for a concrete technology from a wireframe designed with a wireframing tool (i.e. the second step in Figure 1). Please note that the chain has been made specific for a concrete wireframing tool (WireframeSketcher) and a concrete target widget toolkit (Java Swing), though any other source and target technologies are also possible. Next we explain the steps (arrows in the picture) involved in the process.

Source model normalisation

The first step normalises the output obtained from the wireframing tool so it can be used as an input of our process. It consists of transforming the concepts of the wireframing tool to a generic model that contains common GUI concepts such as windows, panels, text boxes, buttons and so forth. With regard to the output of the wireframing tool, two cases are possible. The first one is that the tool outcomes a model that conforms to a defined metamodel, so a model transformation is required to perform the mapping to the generic wireframing model. The second case is that the wireframing tool generates a file that does not conform to any available metamodel. In this case a model injection from the source file can be achieved by using tools such as Gra2MoL [3].

Note that the normalisation step is rather straightforward since in most cases there is a one-to-one mapping between the source and the generic elements. The normalisation step brings us independence of the source technology, so the rest of the approach can be easily reused no matter the wireframing tool.

Region extraction

A view (such as a window or a web page) can be seen as a composition of parts (maybe implicit) that give a structure to the widgets that are in the view. From now on we will refer to these parts of the views as *regions*. The region extraction goal is twofold, identify regions and make the containment explicit:

- **Region identification**. It consists of explicitly create regions for those graphical elements that are used to visually group widgets. For example, think of a group of widgets which are in a panel surrounded by a border.
- **Explicit containment**. In some cases elements are not actually contained in a container, but they are overlapped. For example, there could be a panel with a border that visually contains some widgets but they are all overlapped on a window at the same level without any nesting.

Both, region identification and explicit containment enable matching the layout 'physical' and visual structure, what greatly simplifies the reverse engineering algorithms.

The outcome of this step is a Region Model. This is a model that adds additional information to a Generic wireframe Model to make explicit the visual containment relationships between widgets. In a Region Model, every GUI element is represented by a region, that is a rectangular area of the GUI which is defined by means of coordinates. Moreover, aditional regions are created to group spatially-related widgets. Region models have three main features: i) every GUI element is associated with a region defined by two pairs of coordinates, ii) container and non-container widgets must not exist at the same level (i.e. a region annotating a container cannot be a sibling of a region annotating a non-container), and iii) overlapped regions are not allowed. Further explanation about the region identification can be found in [12].

Make positioning relative

A coordinate-based positioning system is not desirable, since coordinates are technology-dependant and are not well displayed across technologies. Moreover, they are not flexible

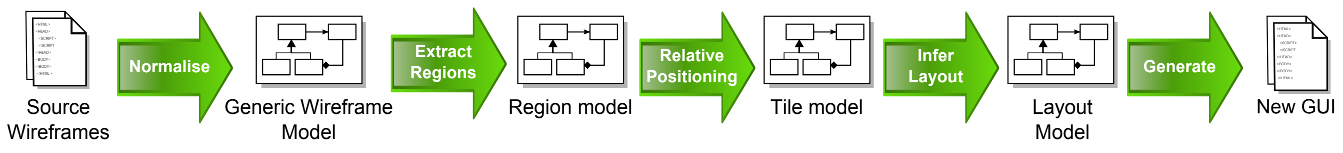

Figure 2: The layout inference approach

enough to perform beautification-wise actions, such as dynamic resizing. A relative positioning-system that represents relationships between elements would be preferable.

For this reason, this step aims at moving the positioning system from coordinates to relative positions. The outcome of this step is the Tile Model, which is used to represent the relative positions among the elements by means of a directed acyclic graph. In this graph, the nodes (called **tiles**) represent GUI elements (regions or widgets such as text boxes) and the edges (**relations**) represent the relative position between a pair of tiles.

When a tile represents a region, then it contains a graph composed of the regions or widgets they belong to that region. Tiles that represent widgets will not contain any graph. Aditionally, tiles include information about the percentage of width and height they take out of the container dimension, and the percentage of separation from the container boundaries.

Relations have three main attributes to represent the spatial relations. The first attribute is the Allen interval for the X-axis, the second one is the Allen interval for the Y-axis and the third one is the closeness value. The Allen intervals [1] express the spatial relationships for a pair of segments in one dimension. For example, the *MEETS* interval indicates that the end of the first segment meets the beginning of the second segment, and the *CONTAINS* interval means that the second segment is strictly contained in the first one. Be aware that the relations are directed, i.e. the source and the target nodes of the relation are distinguished, and this indicates how to interpret the Allen intervals. The closeness value is a discrete estimation (e.g. VERY_CLOSE, CLOSE, FAR, etc.) of the separation of the pair of connected elements, so all the relations with a more or less similar distance will be marked with the same closeness value. Note that the Allen intervals also express the alignment between a pair or related tiles.

Layout inference

Based on the relative positioning graph (the Tile Model), the layout inference algorithm is applied to get a high level representation of the layout in the form of a Layout Model.

The main idea of the devised algorithm consists of generating all the possible permutations of a predefined set of layout patterns and checking for each sequence if we can meet a solution by applying the layout patterns in the order specified by the sequence. A solution sequence is a composition of layouts that covers all the nodes of the graph representing the position of the elements of a portion of the view. Then each different solution that is found is assessed by a fitness function. The fitness function returns a score based on the layout types and how they are combined. For example, it will score a grid layout better than a simple flow layout. The best solution will be the solution with the best fitness value. This approach has the advantage of offering a list of alternative solutions, which could be interesting to know different implementation options and manually guide the later restructuration process.

As we have mentioned, the pattern matching engine matches layout patterns against a graph. The layout patterns implemented are four:

- **Horizontal / Vertical flow**: select the nodes that are connected by only one outgoing edge with the xInterval / yInterval equals to BEFORE or MEETS.
- **Grid**: searches recursively for 2×2 node-subgraphs connected among them so they form a rectangular grid topology of $n \times m$ nodes. There is a constraint that the nodes inside the grid cannot contain edges that point to nodes outside the grid, only the border nodes of the grid are allowed to have connections to the nodes outside the grid.
- **Border**: analyses the graph looking for subgraphs containing some of the following areas: north, south, east, west, centre. For example, a subgraph with just a part aligned to the left (east) and a part aligned to the right (west) would match.

The process of matching the layout patterns of the sequence is performed according to the closeness levels. The algorithm defines a current closeness level which is used to limit the relations which the patterns are matched against, so only the relations with a closeness level equals or lower than the current level are candidates to be matched. Therefore, at first the current level is the lowest level, so the layouts in the sequence are applied to the relations with the lowest level. If there are no matches, then the current level is increased and the sequence is applied to the relations marked with the lowest level and the next one. If there are no matches, the and so forth. Note that this makes a partition of the graph in connected components, so each connected component is a subgraph of the original graph where all the edges have a closeness level equals or less than the limit.

When the sequence has been tried with all the closeness levels and there have beeen no matches, the algorithm stops since no solution can be found by applying such sequence.

As a result of this step, a Layout Model which represents the design of the views in terms of composite layouts is achieved.

GUI code generation

The information contained in the Layout model is rich enough to generate the GUI layout for any technology. This last step is aimed at generating the GUI code for the target technology or platform. This can be done either by directly generating the final GUI code with a code generator or by performing a M2M transformation prior to the generation phase. Given the semantic gap between the Layout model and the technology concepts is reduced, both approaches are suitable in most cases.

Another option for generating code for a concrete technology or reusing available third-party tools would be to transform the Layout model into a third-party representation, such as an UIDL defined by a metamodel. For example, we can implement a M2M transformation from the Layout metamodel to the UsiXML metamodel to take advance of the available code generators.

It is worth noting that the Layout model makes the approach independent of the target technology. For each new target technology to be supported, a code generator (or M2M transformation plus code generator) will be required. Fortunately, implementing such generator will not often require a big effort but it will be relatively straightforward.

5. THE TOOL

We have developed a prototype of the tool that runs as an Eclipse plugin. For the time being, the wireframing tool supported is WireframeSketcher [14] and the target toolkit is Java Swing, though in the future other source tools and target APIs/toolkits will be available. WireframeSketcher is a wireframing tool that helps designers to quickly create wireframes, mockups and prototypes for desktop, web and mobile applications. It can be executed as a desktop application as well as a plug-in for any Eclipse IDE. WireframeSketcher outputs an Ecore model conforming to a predefined metamodel which is available in the distribution. Figure 3 shows an example of the WireframeSketcher view in the Eclipse IDE. Stakeholders and developers meet and create an initial GUI (with the help of WireframeSketcher) which is discussed and iteratively refined until it is validated. Then, the developer puts the GUI model in a Eclipse project and runs our plugin after selecting the Java Swing toolkit as a target technology. Finally, the final GUI code is automatically generated in that project. Figure 4 shows the Swing window as a result of the execution of the generated Java class. Note that the generated code respects the layout, the gaps between the widgets and the alignment with regard to the window.

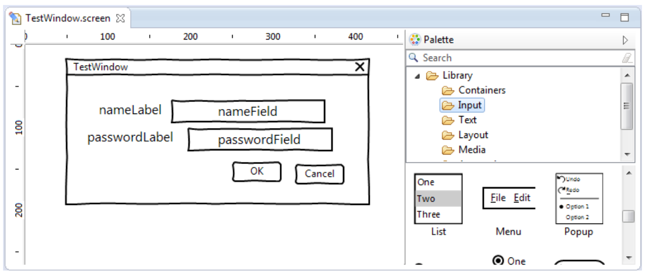

Figure 3: WireframeSketcher test window

6. CONCLUSIONS AND FUTURE WORK

In this paper we have outlined the model-driven approach we have devised to perform reverse engineering of the layout of wireframes, which can be integrated with existing

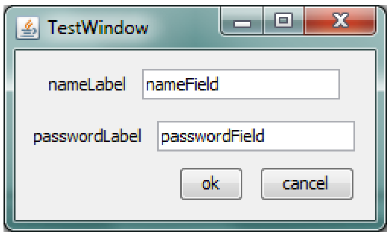

Figure 4: Java Swing test window

wireframing tools and allows us to generate new GUIs for different toolkits. We have implemented a prototype of the approach that takes wireframes created with WireframeSketcher and generates Java Swing GUIs.

As a future work we will improve the prototype of the tool to support additional source and target technologies. We will conduct an experiment with users to check whether the generated layout matches the idea that these users had in mind when designing a suggested test case.

7. REFERENCES

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[2] Balsamiq. http://www.balsamiq.com.

[3] J. L. Cánovas and J. García. A domain specific language for extracting models in software modernization. In *ECMDA-FA'09*, 2009.

[4] A. Coyette, S. Faulkner, M. Kolp, Q. Limbourg, and J. Vanderdonckt. Sketchixml: towards a multi-agent design tool for sketching user interfaces based on usixml. In *TAMODIA '04*. ACM, 2004.

[5] A. Coyette, S. Kieffer, and J. Vanderdonckt. Multi-fidelity prototyping of user interfaces. In *INTERACT'07*. Springer-Verlag, 2007.

[6] B. Hartmann et al. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In *UIST '08*. ACM, 2008.

[7] G. Johnson, M. D. Gross, J. Hong, and E. Yi-Luen Do. Computational support for sketching in design: A review. *Found. Trends Hum.-Comput. Interact.*, 2(1):1–93, Jan. 2009.

[8] C. Lutteroth. Automated reverse engineering of hard-coded gui layouts. In *AUIC'08*, volume 76. ACS, 2008.

[9] M. W. Newman et al. Denim: an informal web site design tool inspired by observations of practice. *Hum.-Comput. Interact.*, 18(3):259–324, Sept. 2003.

[10] Reify. http://www.smartclient.com/product/reify.jsp.

[11] J. M. Rivero, G. Rossi, J. Grigera, J. Burella, E. R. Luna, and S. Gordillo. From mockups to user interface models: an extensible model driven approach. In *ICWE*, ICWE'10. Springer-Verlag, 2010.

[12] O. Sánchez Ramón, J. Sánchez Cuadrado, and J. García Molina. Model-driven reverse engineering of legacy graphical user interfaces. In *ASE'10*, 2010.

[13] M. Walker, L. Takayama, and J. A. Landay. High-fidelity or low-fidelity, paper or computer? choosing attributes when testing web prototypes. In *HFES 46th annual meeting*, 2002.

[14] Wireframesketcher. http://wireframesketcher.com.