
Meta GUI Builders: Towards A Model for Generating Domain Specific Interface Builders

Kris Luyten, Jo Vermeulen, Karin Coninx

Hasselt University - Transnationale Universiteit Limburg

Expertise Centre for Digital Media - IBBT

Wetenschapspark 2

3590 Diepenbeek, Belgium

{kris.luyten,jo.vermeulen,karin.coninx}@uhasselt.be

Abstract

We present a new generation of graphical user interface builder tools (GUI-Builder) that allow for the creation of flexible GUI designs and use the vocabulary of the domain for which the GUIs are created. This domain vocabulary contains a set of abstractions that are commonly used in such a domain and a corresponding presentation for each abstraction. This vocabulary is used to generate a GUI builder, so for each different domain the GUI Builder tool encapsulates the domain specific elements in the designer's tool palette. Over time, a vocabulary can be extended or changed when the designer evolves and becomes more familiar with a particular domain.

Keywords

ACM Classification Keywords

H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

Introduction

The design of user interfaces is tedious work and involves many stakeholders. To ensure the interface that results from a [vul aan]. A consistent mapping of

Copyright is held by the author/owner(s).
CHI 2007, April 28 – May 3, 2007, San Jose, USA
ACM 1-xxxxxx

domain-specific elements on a graphical representation is important to increase the recognizability for the designer, and by consequence the consistency and usability of the graphical interface w.r.t. the usage domain.

In this work, we present a meta graphical user interface builder that is generated for a specific domain: a visual builder tool is presented to the designer according to the domain objects that are to be found in the problem domain.

The User Interface Markup Language

The user interface markup language (UIML, [1]) is the cornerstone of our approach. UIML is a canonical XML-based user interface description language that allows a custom naming scheme according to the problem domain. A UIML document expresses the structure, style, content and behavior of a interactive user interface independent of platform, widget set or even programming language. For this purpose a *mapping vocabulary* containing mapping rules from domain objects onto concrete representations is defined.

We provide a highly dynamic rendering engine, UIML.net [2], that transforms a UIML document into a concrete working user interface. This rendering engine will query the mapping vocabulary and instantiate the appropriate widgets from the selected widget set at runtime. This means that a change in the vocabulary does not render the user interface specification unusable: the user interface will be automatically adapted according to the changed mapping rule and presented with the altered concrete widgets.

The same technique as described above to generate concrete user interfaces is also used to generate a Graphical User Interface Builder (GUIBuilder) tool.

[Stuk toevoegen over flexibiliteit, multi-device en patterns/parametrized templates: later nog nodig om goed te motiveren]

Mapping Vocabularies

The mapping vocabulary is the part of UIML that allows to use a custom naming scheme while creating the user interface description. Some changes needed to be made to the OASIS UIML standard to have better support for UIML-based design tools. Since the mapping vocabulary already contains a concrete representation for each domain object, they can be presented directly in the GUI Builder. However, the standard vocabulary only allows to map one domain object onto one *widget class* but does not allow mapping a domain object onto a *set* of widget classes.

There are two approaches to overcome this limitation: (1) extend the mapping vocabulary with more semantics; or (2) add the required information in a separate file. An iconic representation for each mapping rule is required to serve as a candidate representation in the designer tool [gebruik RDF hiervoor]. We choose the first option and allow the mapping vocabulary to be extended while the designer gains more knowledge in the problem domain, thus discovers a set of patterns that she or he considers candidates for reuse. The benefit of having an iconic representation is the flexibility with which we can represent the mapping in an "abstract" way, such as the Canonical Abstract Prototypes [3], while still enabling the designer to generate a high-level prototype.

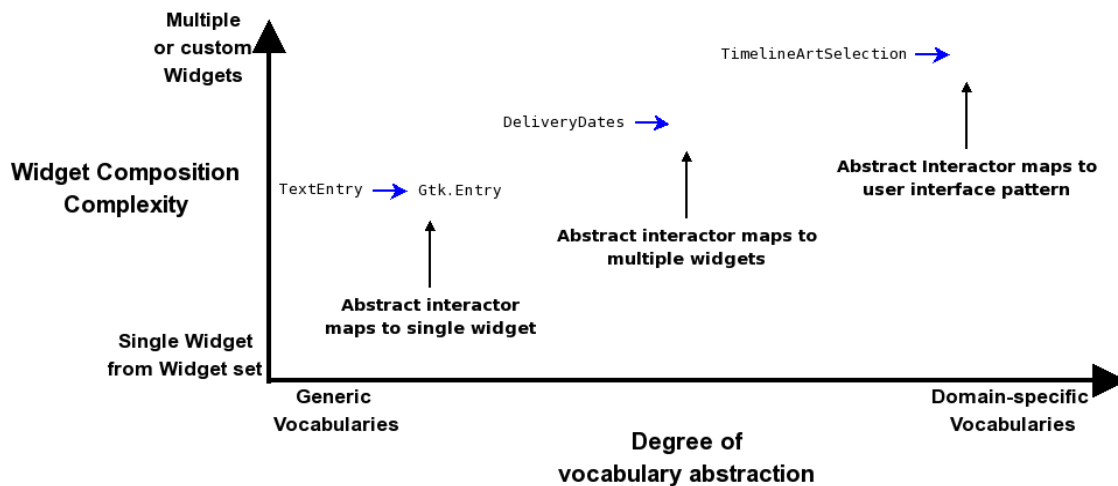


figure 1. The Vocabulary Space: as the vocabulary includes more abstract mappings it becomes more specific to a certain domain and evolves into a body of domain-specific interface patterns.

This also implies the level of abstraction that is supported by the tool can differ. Figure XXX shows a GUI Builder tool that is generated by a vocabulary intended for a designer that is used to work with “regular” design tools such as Glade, Visual Studio Forms Editor, ... Alternatively, a vocabulary that contains a naming scheme for a “car navigation system” designer can be created that maps objects from the source domain onto a suitable user interface patterns.

The most powerful property of this approach is that a vocabulary that only provides a low degree of abstraction can evolve into a vocabulary of a high degree of abstraction as the designer gains more knowledge of the a domain. Figure 1 depicts this

abstraction dimension: a vocabulary evolves from the bottom left region of the vocabulary space towards the upper right region.

Generation of a GUI Builder tool from domain objects

A GUI Builder tool typically allows the designer to compose a user interface by using drag-and-drop operations that move objects from a toolbox into a graphical canvas. The most common dialogs (be it integrated in a single view or a multi-window view) are the *toolbox*, the *canvas*, the *property dialog* and the *treeview*. The toolbox contains a set of representations of domain objects that can be dragged onto the canvas. The canvas represents the graphical representation of how the user interface should look like. The property dialog shows the properties of the domain objects: editing these properties will result in the corresponding change in representation in the canvas as defined by the vocabulary. The treeview gives a hierarchical view on the design: the leaf nodes are the actual widgets, and non-leaf nodes serve as different types of containers for the widgets.

The above description of the different dialogs is identical to the traditional structure of a GUI Builder tool. The toolbox and properties dialog are generated from the vocabulary however, and the information contained in the other windows is contained in a UIML document.

Extending the GUI Builder “by practice”

The main difference is the way the designer can “save” patterns and add them to the toolbox. This can be done as follows: the treeview will be expanded while the designer adds objects to the canvas. If there is a part

of the design that can be reused later on (e.g. a “speed control ahead dialog”) in other designs the designer can create a new pattern by selecting that part of the user interface and naming it. The selection can be done in the treeview (a subtree can be selected) or on the canvas (a set of widgets can be selected) and a name for this pattern must be provided by the designer. This information is saved into the mapping vocabulary as a template and, if the designer wishes to do so, an iconic representation can be added for this pattern. The properties of the different subparts of this pattern are merged so it is considered as a new domain object with its own specific identity now.

Tools and Processes

The tool framework presented in this paper is part of a process we gradually implement to support user-centered engineering of interactive systems. This process is depicted in figure [].

Our approach should bring designers, end-users and domain experts closer together by combining three

important aspects that support more efficient and unambiguous communication:

- a naming scheme that is known to the end-users and domain experts
- a high-level prototype that can be consulted early during the design process
- smooth integration with the application logic and target platform(s)

Future Work

This paper presented the first proof-of-concept we created of a GUI Builder generation tool, or a meta-GUI Builder model.


Conclusions


We have currently a first proof-of-concept implementation of a meta-GUI-builder.


Acknowledgements


The authors would like to thank Jan Meskens who implemented an initial version of the system described in this paper as part of his MSc thesis.

Example citations

 Abrams, M. and Helms, J.,

 Luyten, K. and Coninx, K., Uiml.net: an Open Uiml Renderer for the .Net Framework, CADUI 2004, Funchal, Madeira, 2004

 Constantine, L. L., Canonical abstract prototypes for abstract visual and interaction design, DSV-IS 2003, Funchal, Madeira, 2003

 Myers, B. A., User Interface Software Tools, *ACM Transactions on Computer-Human Interaction*. vol. 2, no. 1, March, 1995. pp. 64-103