Int. J. Human-Computer Studies 86 (2016) 48–62

Contents lists available at ScienceDirect

Int. J. Human-Computer Studies

journal homepage: www.elsevier.com/locate/ijhcs

Reusing UI elements with Model-Based User Interface Development ☆

A. Delgado [a,*], A. Estepa [a], J.A. Troyano [b], R. Estepa [a]

[a] Department of Telematics Engineering, University of Seville, Camino de los descubrimientos s/n., 41092 Seville, Spain
[b] Department of Computer Languages and Systems, University of Seville, Reina Mercedes s/n., 41012 Seville, Spain

ARTICLE INFO

Article history:
Received 5 February 2014
Received in revised form
13 August 2015
Accepted 3 September 2015
Communicated by Fabio Paterno
Available online 11 September 2015

Keywords:
MBUID
Reuse
Software engineering
User Interface

ABSTRACT

This paper introduces the potential for reusing UI elements in the context of Model-Based UI Development (MBUID) and provides guidance for future MBUID systems with enhanced reutilization capabilities. Our study is based upon the development of six inter-related projects with a specific MBUID environment which supports standard techniques for reuse such as parametrization and sub-specification, inclusion or shared repositories.

We analyze our experience and discuss the benefits and limitations of each technique supported by our MBUID environment. The system architecture, the structure and composition of UI elements and the models specification languages have a decisive impact on reusability. In our case, more than 40% of the elements defined in the UI specifications were reused, resulting in a reduction of 55% of the specification size. Inclusion, parametrization and sub-specification have facilitated modularity and internal reuse of UI specifications at development time, whereas the reuse of UI elements between applications has greatly benefited from sharing repositories of UI elements at run time.

1. Introduction

Model-Based User Interface Development describes the user interface (UI) through a collection of models which guide the UI development process. Such models can also drive the UI generation process in a (semi) automated fashion such as in model-driven engineering (MDE), which should reduce the work needed to develop UIs, resulting in a more productive software development process (Viana and Andrade, 2008; Meixner et al., 2010).

MBUID environments (MBUIDEs) have proliferated over the last decades, giving rise to a number of different toolsets and specification languages (Pinheiro da Silva, 2001; Guerrero-García et al., 2009; Meixner et al., 2011). However, in spite of the plethora of available approaches, no MBUIDE has experienced wide adoption by the software industry (Trætteberg, 2008; Molina, 2004). As envisaged by some authors (Pinheiro da Silva, 2001; Ahmed and Ashraf, 2007) the poor reusability of UI model specifications can be argued to partially explain this lack of interest from the industry. More recently, Meixner et al. (2011) have supported this idea, pointing also to the lack of harmonization between the MBUID and MDE worlds, as well as the absence of real-world usage and case studies as important challenges to be faced by MBUID systems in the future. This paper addresses some of these challenges, focusing specifically on reusability.

In general, software reuse increases productivity and software quality as reported in many industrial cases (Mohagheghi and Conradi, 2007), such benefits should also be present in the context of MBUID. Although some environments have been equipped with techniques to support reuse, issues and methods associated with the reuse of UI components and the benefit/cost associated have been shortly addressed in the MBUID community. One possible reason for this could be the complexity of the subject, which involves several inter-related questions such as (a) which UI fragments or models can be subjected to reuse; (b) what technical approaches can be used; and (c) how to assess the benefits of reusing. As stated in Meixner et al. (2011), the answer to these questions becomes even more complicated since emerging standards such as W3C task (Paternò et al., 2014) or Abstract UI models (Vanderdonckt et al., 2014) are not widely adopted yet, and there are scarce real-world usage and case studies that quantify the potential benefits of reusing UI assets.

This paper aims to be a first step in gaining insight into how elements from UI models can be reused. We provide a real-world usage case through the development of six applications with an environment that supports some standard reuse techniques such as parametrization and sub-specification, inclusion or shared repositories. We describe our experience and quantify the benefits

☆This paper has been recommended for acceptance by Fabio Paterno.

* Corresponding author.
E-mail addresses: aldelgado@us.es (A. Delgado), aestepa@us.es (A. Estepa), troyano@us.es (J.A. Troyano), rafaestepa@us.es (R. Estepa).

http://dx.doi.org/10.1016/j.ijhcs.2015.09.003
1071-5819/

of intra- and inter-project reuse of UI specifications. Based on our experience, we provide lessons learned that can be valid for other contexts and provide some advice on the development of future MBUIDEs with potentiated reuse features.

The remainder of this paper is as follows: Section 2 briefly introduces the main concepts of MBUID. Section 3 summarizes current reuse approaches already present in the context of MBUID. Sections 4 and 5 describe the main characteristics of the MBUIDE used in our case study and the techniques supported for reusing UI elements respectively. Section 6 introduces our study case. Section 7 addresses the results based upon our experience, discussing the use of each reuse technique and providing quantitative results of the benefits obtained. Section 8 provides an analysis of the main points of our results. Finally, Section 9 concludes the paper.

2. MBUID overview

MBUID offers an environment for developers to design and implement UIs in a professional, consistent and systematic way (Pinheiro da Silva, 2001; Meixner et al., 2011, 2010). MBUID is based on the idea that the UI can be fully modeled by a set of declarative models each addressing particular facets of the UI such as tasks and presentation. The specification of each model consists of an abstract description of the aspects pertaining to its domain by means of a so-called UI Description Language (Guerrero-García et al., 2009). Model specifications lead the UI development life-cycle and provide the basics for automatic UI generation.

Consensus on the set of models and languages for UI description has remained elusive in the past. Most MBUIDEs have defined their own languages and models (Guerrero-García et al., 2009; Meixner et al., 2011). However, some models were recurrently used by a number of environments in the early 00s (Pinheiro da Silva, 2001; Vanderdonckt et al., 2003):

- The *User model*, which specifies a hierarchical break-down of users in stereotypes that share a common role.
- The *Domain model*, used to define the objects accessible to users via the UI.
- The *Task model*, which describes the set of tasks that users are able to accomplish, its hierarchical decomposition and its temporal relations and conditions.
- The *Presentation model*, devoted to presentation aspects of the UI. It can, in turn, be decomposed into the *Abstract Presentation model*, dealing with abstract level descriptions of the structure and behavior of the UI objects, and the *Concrete Presentation model* that describes in detail the parts of the UI using modality-dependent (i.e. graphic and haptic) concrete interaction objects (Vanderdonckt and Bodart, 1993).
- The *Dialog model*, which defines the set of actions the user can carry out within various system states and the transition between these states. It links tasks with interaction elements forming a bridge between the Task and the Presentation models.

According to Meixner et al. (2011), the Task, Presentation and Dialog models can be considered the core models since they have direct influence on the content and appearance of the UI.

Different MBUID approaches can be related using the Cameleon Reference Framework (CRF) (Calvary et al., 2003). Since its definition in 2003, the CRF has become widely accepted in the HCI community as a reference for classifying UIs supporting multiple targets, or multiple contexts of use on the basis of a model-based approach (Meixner et al., 2011). The framework describes different layers of abstraction related to the model-based development of user interfaces:

- *Concepts-and-Tasks*: Specifies the hierarchies of tasks that need to be performed on/with domain objects (or domain concepts) for a particular interactive system (Meixner et al., 2010). Traditional Domain and Task models belong to this abstraction layer.
- *The Abstract UI*: Expresses the UI in terms of interaction units without making any reference to implementation in terms of interaction modalities or technological space (e.g. computing platform, programming or markup language) (Vanderdonckt et al., 2014). Abstract presentation or dialog models belong to this layer of abstraction.
- *The Concrete UI*: Describes concretely how the UI is perceived by the users using concrete interaction objects (Vanderdonckt and Bodart, 1993). These objects are modality-dependent but implementation-language-independent. Concrete Presentation models belong to this layer.
- *The Final UI*: Expresses the UI in terms of implementation-dependent source code. It can be represented in any UI programming or mark-up language (e.g. Java or HTML).

The CRF distinguishes between development and run time phases. In the development phase, initial model specifications are refined in successive steps. Ultimately, a Final UI expressed in source code is generated in a manual or automatic fashion from the concrete UI (Fonseca et al., 2010). Final UIs can then be interpreted or compiled as pre-computed UIs targeted for specific contexts of use (i.e. user, platform and/or environment) and plugged into an environment that supports dynamic adaptation to multiple targets at run time (Calvary et al., 2003).

3. Related work

Improving the reusability of model specifications in the context of MBUID has been addressed in the past. The main approaches found in the literature can be classified as:

- Reuse based on *the UI Description Language*: Some languages have foreseen the need to reuse fragments of specifications and have defined specific technical methods to deal with it. For instance, Hyatt et al. (2001) allows referencing specification fragments defined in the same document or included from an external document using the special processing instruction $<?xul-overlay?>$. In XICL (Sousa and Leite, 2005; de Sousa and Leite, 2006) the UI is made up of components which are somehow similar to classes. XICL allows the inclusion of components which can be extended through certain language tags and attributes (e.g. *extends*, *child*) in an analog way to object-oriented programming. In UIML (Abrams and Helms, 2004), frequently used specification fragments can be defined as templates using the $<$template$>$ tag, and reused in a flexible way (e.g. cascade, replace or union the referenced element). In addition, templates can receive parameters whose value will be passed when referencing. All previous approaches allow the developer to create a library of reusable assets, enabling the scope of the reuse to be internal (i.e. intra-project) or external (i.e. inter-projects).
- Reuse through *multiple transformations*: Third (TERESA Mori et al., 2003, XMobile Viana and Andrade, 2008) and fourth generation (e.g. MARIA Paternò et al., 2009, GUMMY Meskens et al., 2008) MBUIDEs are capable of generating multi-target UIs (Meixner et al., 2011). Therefore, a single UI model specification can be transformed multiple times targeted to different context of use which are defined for a set of users, hardware and software platforms, and physical environment (Calvary et al., 2003). This can be viewed as a kind of generative programming (Mohagheghi and Conradi, 2007). The scope of reuse is normally