On the Reusability of User Interface Declarative Models

Antonio Delgado, Antonio Estepa, José A. Troyano and Rafael Estepa

Abstract The automatic generation of user interfaces based on declarative models achieves a significant reduction of the development effort. In this paper, we analyze the feasibility of using two well-known techniques such as XInclude and Packaging in the new context of reusing user-interface model specifications. After analyzing the suitability of each technique for UI reutilization and implementing both techniques in a real system, we show that both techniques are suited to be used within the context of today's existing model-based user interfaces.

29.1 Introduction

The automatic generation of user interfaces (UIs) through model-based user interface development environments (MB-UIDEs) [1] is likely to become a candidate technology for reducing the development effort in software industry. MB-UIDEs use abstract models (named user interface models or UIMs), which can be used for developing and automating the generation of user interfaces. Although these models could potentially be expressed through graphical or textual languages, they are commonly coded through XML-based user interface description languages (UIDLs) [2]. In fact, XML has became a de facto standard to specify UIMs potentially suited for MB-UIDEs.

In addition to automation, another way to reduce the development effort is through reuse. Software reuse is a well-known area of study in software engineering, which aims to reduce redundant work. There is a wide range of reusable assets: requirements, test cases, etc. The reuse of UIMs is not a new idea. Some techniques such as UI patterns, multi-device user interface development, etc. have been proposed [3]. However, few efforts have been made to inquire about the reusability of UIs through the reuse of UIMs specifications.

A. Delgado (✉), A. Estepa, J.A. Troyano and R. Estepa
Escuela Superior de Ingenieros, C/ Camino de los descubrimientos s/n, 41092, Sevilla, Spain
e-mail: aldelgado@trajano.us.es

Only a few of current UIDLs provide mechanisms for reusing fragments of the UI models: UIML templates [4], XICL components [5], and XUL overlays [6]. However, the methods used are specific of the UIDL used [4, 5] (i.e., define their own tags) and consequently, incompatible between them, or use special XML processing instructions [6].

The objective of this work is to propose and analyze the pros and cons of two techniques for UIMs reuse: *XInclude* [7] and *Packaging*. Both methods are commonly applied in contexts different from UI reuse but we believe that both are potentially valid in the reuse of UIMs specifications, being UIDL-independent and having a minimum impact in the actual UIDLs syntax. To validate our proposal, we have implemented both techniques in a real MB-UIDE [8] using this experience as a feedback for our analysis.

29.2 User Interface Models: Reuse

MB-UIDEs attempt to formally describe the users, tasks, and presentation for a UI, and then use this set of formal UIMs to guide the full process of developing the UI. To identify what portions of the UIMs are eligible for reuse, we should first describe the models commonly included in MB-UIDEs as well as the objects included in each model. This will let us have a view of the reusability degree of each model. There seems to be a common agreement in defining the following UIMs [1]:

- The Domain Model (DOM): Defines the data objects that a user can view, access, and manipulate through a UI. The elements included in this model are typically classes, entities, attributes, operations, and relationships. Developers are familiar with these objects and the reusability degree depends typically on the model's subject and developer's expertise.
- The User Model (UM): Specifies a hierarchical decomposition of the user population into stereotypes. Each stereotype brings together people sharing the same value for a given set of parameters. Components of this model are users, groups, relationships between them, and their properties. Initially, we do not think that reusing users or groups can be very interesting because they are usually centralized in shared directories (e.g., LDAP repository). However, some parameters from the UM specifications such as user preferences or profiles are potentially interesting for reusing (e.g., accessibility issues).
- The Task-Dialogue model (TDM): Describes the tasks that the users are able to perform as well as its interrelation. The elements commonly used to define the TDM are tasks, goals, actions, preconditions, and postconditions. The definition of tasks is likely to become a reusable asset since some tasks can be repeated in different projects. Tasks are usually defined in a hierarchical way (e.g., Concurrent Task Trees), so full trees of task definitions or portions of them can be defined as reusable components.
- The presentation model (PM): Two different submodels can be distinguished:

– The abstract presentation model (APM) provides a conceptual description of the structure and behavior of the visual parts of the UI. Commonly the APM is composed by views and abstract input objects. Reusing abstract interface specifications is an interesting feature since the abstract definition of a UI can be repeated in other projects (e.g., login dialog).
– The concrete presentation model (CPM) details the visual parts of the UI. The components of this model are windows, concrete interaction objects, and layouts. Reusing CPM specifications is particularly interesting in projects where several applications are interrelated since it provides a consistent presentation across project-wide UIs.

Depending on the target, we might be interested in reusing parts of some UIMs as a whole reusable parameterized asset or subsystem (e.g., an instant message system) or just single model elements such as a user profile, a form, or a widget.

29.3 Approaches for Reuse

Since most MB-UIDEs use text-based languages for UIMs specification, one natural temptation is to just copy–paste reusable pieces of code. This is identified as a bad practice for code development (*cut&paste programming* [9]) leading to an unmaintainable and oversized code. A better solution is to encapsulate the reusable component properly as presented in following subsections.

29.3.1 Model Specification Reuse Using XInclude

XInclude [7] introduces a generic mechanism for merging XML documents in applications that need an inclusion mechanism to facilitate modularity. An obvious requirement to apply XInclude in a UIDL is that it must be XML-based. The use of XInclude provides some immediate advantages. First, large specifications can be modularized in fragments easier to handle. Second, any model component in a specification of the UI can be coded in a XML file and included in any other specification, which requires using the functionalities offered by that component.

XInclude also has some drawbacks that should be noticed. It defines no relationship with the document validation or transformation. XInclude describes an infoset-to-infoset transformation but it does not specify the XML parsing behavior. Therefore, a document can be validated before or after inclusion, or both, or neither. Consequently, one needs to take into account the following:

- If we decide to perform document validation before the inclusion of the components, the validation method (DTD o XML Schema) should support the validation of XInclude tag elements. Consequently, we should add the definition of these elements in the UIDL validation method. Additionally, the included components

should have been previously validated to avoid errors. Also, the validation mechanism can not be used to validate the included components because they are just fragments from other specification and consequently are not well-formed documents.
- If we validate the declarative specification after inserting the included components we need a method to be aware of the correctness of the included code, which is not always possible. In addition, the line numbers that we would obtain in case of error messages would be wrong. Therefore, whatever option we choose would require specific tools for component validation.

XInclude can be a good solution for those UIDLs that specify all the UIMs in one XML-based document (e.g., UsiXML or XIML). However, a number of current UIDLs only support two or three UIMs [2] (e.g., UIML, XUL, or XICL) leaving the rest models out of the XML document.

29.3.2 Models Specification Reuse Using Package Management

Packages are widely used today in computing: from operating systems (e.g., MSI, RPM, DEB) or file management (e.g., TAR, ZIP, CAB) to programming languages (e.g., JAR, WAR). However, to the best of our knowledge, its use for reuse UIM components has never been proposed. To be able to use packaging for reusing MB-UIDE components, it would be necessary to meet the following requirements:

1. The UIMs need to be in a format allowing the extraction of model objects, which are parts of the reusable component (e.g., textual language).
2. A file format must be set for package building.
3. It is necessary to define the metainfo that will be included in the package for each component (e.g., package version, dependencies, etc.).
4. The use of packages in MB-UIDEs creates the need of implementing a package management system to automate the process of installing, upgrading, configuring, and removing component packages.

Packages allow reusing either a complete subsystem or part of its components by including the corresponding fragments in the package content. In addition, the use of packages would allow to include different UI specifications written in different languages (even those which are not XML-based) in the same package.

However, the use of packaging also has some problems to be addressed – first, the problem of dependencies [10]. This problem is not new and it can cause situations such as the commonly known “DLL hell” experienced by Microsoft Windows developers and users. Common problems in software packaging are also conflicts or security issues [11]. Finally, the solutions adopted are typically platform dependent, and errors in package definitions can only be fixed by the package developer.

29.3.3 Testing the Approaches

The previous approaches have been tested in a MB-UIDE (WAINE) [8] oriented to the development of Web-based management applications. The MB-UIDE uses a XML-based language named ASL to specify the UM, TDM, and APM. WAINE uses other languages such as class diagrams or ERDs for DOM specification, and configuration text files, HTML templates, and CSS for CPM issues. Thus, WAINE represents a high-complexity case since it uses both XML and non-XML specification languages in the UI development.

The XML-based UIDL has been upgraded from its original version to allocate the use of XInclude. Currently we are validating the ASL specification before the inclusion of the components. This led us to modify the original DTDs for validation to add the XInclude element. In addition, we have developed new utilities to validate not well-formed documents (UIM components).

We also developed a package management system specifically for this MB-UIDE, which let us reuse components of any complexity. Their main components are the *wpk* files and the *wpkg* tool. The wpk files are tar gzipped files that can, among others, contain an ASL file in addition to other files that can be necessary to describe a complete component (fragments of the DOM, CPM, files, etc). Wpk files also contain a directory with metainformation about the package and two scripts *preins.sh* y *postins.sh*, which are executed before and after the component installation, respectively. The *wpkg* tool allows to list, add, or remove packages to a user interface.

Since our system does not specify all UIMs in XML, the use of XInclude can be useful to modularize a large project or to reutilize single fragments of models, but it lacks the completeness needed to reutilize complete subsystems since they typically need pieces of every UIM. So, in practice we often use WAINE packages instead of XInclude as our main reusing technique. However, the development effort to implement a package management system is bigger than modifying the document validation system (for XInclude use).

To summarize the analysis done in this paper, Table 29.1 shows a brief comparison between the proposed reutilization approaches.

Table 29.1 Comparison between reuse approaches

Feature	XInclude	Packages
Based on standards	Y	N
Allows to reuse isolated components	Y	Y
Allows to reuse complete subsystem	[a]	Y
Needs to modify validation System	Y	N
Needs package management system	N	Y
Independent of UIDL	Y	Y
Independent of MB-UIDE	Y	N
Implementation effort	Very low	High

[a]Valid only for those UIDLs supporting all UIMs

29.4 Conclusions

The reuse of MB-UIDE's specification models can achieve a reduction of the UI development time. We have analyzed two mature standard UIDL-independent techniques for UIM reuse: XInclude and Packaging. Both are typically applied in diverse fields but never been proposed for UIM reuse in the software development process. We conclude that both techniques are well suited for UIM reuse.

References

1. Pinhero, P. (2001): User interface declarative models and development environments: A survey. Interactive Systems – Design, Specification, and Verification: 7th IWS, pp. 207–226
2. Souchon, N., Vanderdonckt, J. (2003): A review of XML-compliant user interface description languages. LNCS. Interactive Systems. Design, Specification, and Verification, 7th International Workshop, DSV-IS 2000, Limerick, Ireland.
3. Feng, S., Wan, J. (2007): User interface knowledge reuse and multi-device user interface development. IEEE International Conference on Automation and Logistics, Shandong, China.
4. OASIS UIML TC (2004): User Interface Markup Language Specification.
5. Gomes de Sousa, L., Leite, J.C. (2005): XICL: An Extensible Mark-Up Language for Developing User Interface and Components. Springer, Netherlands.
6. Mozilla Developer Center: XUL overlay. http://developer.mozilla.org/en/docs/XUL_Overlays.
7. W3C: Xml inclusions, http://www.w3.org/tr/xinclude.
8. Delgado, A., Estepa, A., Estepa, R. (2007): Waine: Automatic generator of Web based applications. Third International Conference on Web Information Systems and Technologies, Barcelona, Spain.
9. Cheng, Y.P., Liao, J.R. (2007): An ontology-based taxonomy of bad code smells. Proceedings of the third conference on IASTED International Conference: Advances in Computer Science and Technology, Phuket, Thailand.
10. Hart, J., D'Amelia, J. (2002): An analysis of rpm validation drift. Proceedings of the 16th USENIX Conference on System Administration, Philadelphia, PA.
11. Mancinelli, F. (2006): Managing the complexity of large free and open source package-based software distributions. ASE 2006, Tokyo, Japan.