Formalizing Markup Languages for User Interface

Luis G. M. Ferreira
Grupo Sistemas de Tecnologia e Informação, IPCA-EST
Barcelos
lufer@ipca.pt

Resumo

This work focus on the formalization process of user interfaces specification using eXtended Markup Language (XML) description. As an instance of this process, we explore a VDM-SL *specification of* UIML - *User Interface Markup Language. The main results are tested in a particular graphical* tabular *OLAP features case study, towards a definition of a Visual Component Library, with UI components composition and reuse.*

Palavras-Chave

User interfaces, XML, VDM-SL, Formal methods, OLAP.

1. Introduction

For the companies of today it is crucial the full integration and consistency of all the information which flows or is stored in its databases. The number of specific and different applications, manipulating this information, is necessarily large. Maintenance and support of these applications requires expensive team work.

The (conventional) *Relational Database Management Systems (RDBMS)* no longer can guarantee timely efficiency in the answers to complex queries. The treatment of information in bidimensional format (tables and spreadsheets, for example), the inability of the analysis, transformation and consolidation of such information, restrict the overall application of these systems.

The advantages of formal specifications of complex systems using formal methods are well known. It is known and accepted by a considerable group of persons as a way to avoid ambiguities and clearly guarantee correctness and consistency of the developed programs. In this context, it is useful to experiment these guaranties on user interface development, exploring some kind of merging processes of new specification methods with existing graphical objects specifications.

As a consequence of this large set of graphical elements, many of them only found in pure commercial application, its formal specification process, towards a *Visual Component Library - VCL*, could become complex. This complexity can even increase during the tentative abstraction of real problems.

This paper is organized as follows. Section 2 presents the motivations and main technical guidelines followed on this work. This is followed, in Section 3, by the brief presentation of - at time of writing - technologies on modelling, specifying, designing and programming user interfaces, as well as OLAP concepts and Markup Languages applied to interfaces description. Section 4 discusses the archived UIML VDM-SL specification, followed by resultant table model and its specification on VDM-SL on Section 5. Section 6 describes the developed prototype and main supporting tools. Finally, Section 7 concludes the paper and foretell future developments.

2 Problem Statement

This work stands from the thoughts of Brad Myers [Brad 98], where the area of the UI is an example of the strong influence of academic research in the industrial way of doing things. In its essence, this research intends to apply formal semantics techniques to the user interface development, in order to impose scientific rigor on the whole process, from interface specification and validation to its transformation ("transcoding"). However, the application of formal methods should never surpass the limits of versatility, trying whenever possible to build the specification through the composition of components previously specified.

By applying formal methods, UI development becomes a rigorous discipline with focus on higher abstraction levels (relative to implementation), using visual components such as *ListBox*, *Button*, *TextBox*, *Menu*, etc.) and a semantic set of rules which describes the process.

The *Sets Theory* [Oliveira 03] semantic specification of each component will resort to semantic models, describing their distinct internal states, associated to execution of each operator.

The process usually starts from a data model, properly (or not) normalized, to describe the intended information system. Using *VDM-SL*, developers will try to specify its data

model, by identifying the interface components, necessary for its information visualization. To guarantee coherence in the final placement of the components in the interface layout, it should be interesting the use invariants which, in the later phase of layout placement, should prove its relative position.

3. State-of-the-Art

This work, being specially focused with data Visualization in application interfaces, gave priority to the analysis of the existing technologies in the area. In [Phanouriou 00], *Constantinos* summarizes the main steps that can be identified in this process and, as the *Seeheim model* [Pfaff 85] advocates, the importance on maintaining a real separation between the presentation layer and the remaining application layers.

Amongst the known user interface Models, from the "ancient" to the recent ones (*Arch* Model, *MVC Model*, *PAC Model*, *XForms*, *N-Tier Model*, *MIM Model*[Phanouriou 00]), perhaps the first and the most significant UI model was *Seeheim Model*, presented at the Seeheim Workshop (Berlin, 1985). As this model describes, as well as on recent architecture models (such as *N-Tier*), there are evident concerns in separating responsibilities between interfaces and the rest of the application.

The *MIM -Meta-Interface Model*, which extends the level of abstraction of the *Slinky Model*, was created with abstractions mechanisms to describe interfaces that can map into multiple and distinct types of devices. It divides the interface into three major components: *presentation*, *logic* and *interface*.

The *UIML*, markup specification language for generic interfaces description explored on this work, is based upon MIM model.

3.1 Multidimensional Analysis

Once we intend to explore OLAP features applied to tabular object as a case study, the Multidimensional Analysis technique over multidimensional information was carefully explored.

Multidimensional data models categorize data as being either *facts*, which means data values and eventual attributes, or as being *dimensions*, that categorize the facts and are mostly textual. The model of a Multidimensional Database is an array of n-dimensions (often called *Hypercube* or collection of related cubes). Although the term "cube" implies three dimensions, a cube can have any number of dimensions.

Each dimension is associated to a hierarchy of data levels consolidation. For example, a dimension "*time*" can have a hierarchy with levels *day*, *month* or *year*. A dimension works as an index of identification of values in the array. Each array position, corresponding to the intersection of all dimensions, is a *cell*.

Operations like *Rotation, Ranging, Rolling-Up, Drill-Down, Hierarchies, etc*. should be available, supporting OLAP technologies.

3.2 Markup Languages for User Interface description

Over the past few years, there have been a number of industrial and academic initiatives to standardize many data types towards application interoperability. However, it seems that the same did not happen in the interface design area.

It is relieving to know that the *W3C - World Wide Web Consortium* and *OASIS - Organization for the Advancement of Structured Information Systems* have worked towards the great dissemination of XML and of some standards for application interoperability (like SOAP[1]) and several XML applications for multiple distinct areas. Focusing on our main goal, there are already several XML applications aiming user interface specifications[2].

Noting the scope, requirements and structure of *XIML - eXtensible Interface Markup Language*, *XUL - XML-based User-interface Language* (by *Mozilla* project), *AUIML - Abstract User Interface Markup Language* and *UIML - User Interface Markup Language*, we focus our attention on the last one, mainly because its extensibility and facility.

We can also confirm that *UIML* does not walk alone on this path, ignoring all other initiatives. For instance, in the current *UIML* version (3.0) it is even better to use XForms with *UIML* than to work directly with HTML

4 *UIML* Formal Specification

The *UIML* should not be considered a pure VCL because it "works" away from any concrete representation. However it can be understood as so in the sense that it describes visual components that can be represented in a concrete graphical environment.

All UIML elements were formally specified. <uiml>, <head>, <structure>, <peer>, etc., using VDM-SL notation as following code excerpts try to represent.

The <uiml> specification was:

```
Uiml :: head: [Head]
        members:Member*
inv uiml = uniqueIds(uiml) and
validProperties(uiml);
```

where *inv* represent the invariant which must be respected.

The <interface> element specification was,

```
Interface ::    intele: InterfaceElements*
                id: [ID]
                source: String
                how: [SourceModes]
                export: [ExportOptions]
inv i = uniqueIDs(i);
```

The <structure> element specification was,

[1] SOAP - Simple Object Access Protocol

[2] http://xml.coverpages.org/userInterfaceXML.html

```
Structure ::    parts : Part*
                id: [ID]
                source: String
                how: [SourceModes]
                export: [ExportOptions]
inv s = uniqueIDs(s);
```

In order to certify semantic rules on operators utilization, several VDM-SL *invariants* were created, as described in the code excerpts. The code below represents an excerpts of one of such definitions.

```
UniqueIDs: UIMLElements -> Bool
    UniqueIDs(t) =
        case t:
         mk-UIML(-,-) ->
                rng allIDs(t)={1},
         mk-Peers(-,-,-,-,) ->
            rang peersIDs([t],{->})={},
         ....
```

All the VDM-SL specifications respect the actual UIML XML Schema[Harmonia 02]. Due to the extension of *UIML* language, some particularities were not considered on this work, mainly rules associated to *template* and *property* elements.

The resultant *VDM-SL* specification, will be tested using a table object.

5 Table *VDM-SL* specification

To better understand OLAP methods behavior (functions, operators or transformers), they must be considered as **attributes** if they support column and row characteristics (ex. *column name*, *column width*, etc.); **operators** if they work with table structure (*addRow*, *delRow*, etc.); **functions** if they support data calculus (*sum*, *avg*, *sort*, etc.) or even **transformers** if they manipulate the original table structure (*hideCol*, *drill-down*, *rotation*, etc.).

Following excerpts of VDM-SL code specify some implemented table operators. For instance, the operator responsible from table creation - *mkTable* - was defined as:

```
mkTable : Style x Rows -> T
    mkTable (s,r) = mk-(s,r);
```

For example, the *dT* default table creation, with no rows and no graphical information (background color, cell spacing, etc.), could be obtained by the following expression:

$$mkTable(\texttt{"dT"}, \{\}, \{|\ \texttt{-}>\});$$

Another table *t* could start its construction using:

$$mkTable(, \texttt{"r1"}\ |\ \texttt{-}> mk\_(, \texttt{"Mark"}\ |\ \texttt{-}> mk\_(, \texttt{"Ford"})));$$

The *hideRow* operator, which hides a particular row:

```
hideRow : T x Rid -> T
    hideRow (t,rid) =
    mk-(t.#1,markRow (t.#2,rid,"h"))
pre rid 2 dom (t.#2);
```

The *delRow* operator, which delete a particular table row,

```
delRow : T x Rid -> T
delRow (t,rid) = mk-(t.#1,{rid}<-t.#2);
```

In our example, the expression

$$delRow(t, \texttt{"r1"})$$

will remove all *Black Ford* information concerning.

A similar process was done for all others implemented operators.

6 Prototype and Supporting Tools

In order to experiment the animation of VDM methods, mainly *OLAP* functions, we have decided to create an HTML prototype whereby we can visualize all table transformations. This application uses our VDM specifications, *UIMLSpec* and *UIMLSpecTab*. From CGI HTML forms behavior, the *VDM-SL* methods are called, then *UIML* is generated and rendered again to HTML. Figure 1 depicts the prototype architecture.

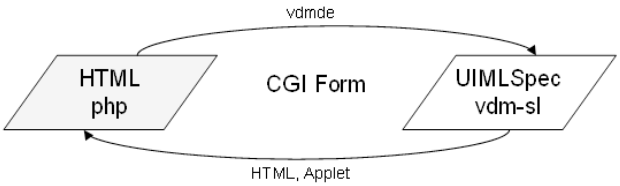

Figure 1. Prototype architecture

As supporting tools used on the four phases of the process, on Phase 1 - *Transcoding* UIML *to VDM-SL*, we generated a XML StyleSheet (*uiml2vdm.xsl*)to transform UIML into VDM-SL. Next we created directly in VDM-SL, a Pretty-Print tool - *vdm2uiml.vdm* - which works as a script and executes a parsing process over *VDM-SL* specifications. It generates a correspondent *UIML* syntax for each specified element.

In Phase 2 - *Verifier*, was created a verifier to support the *validation* phase of our process. This validation allows to test the consistency of the VDM-SL generated code along the transcoding process, verifying the "similarity" between both *UIML* codes.

The Phase 3 - *Abstraction*, has not been completely developed. All reasoning implicit in it is sketched in Future Work. The process starts from *VDM-SL* table abstraction towards the creation of a visual components library.

This work should show that it is possible to perform transformations by calculation. Then it is necessary to find different candidate implementation objects. This is a natural property of adaptable interfaces.

In our case study, the idea stays as "it is possible to get a new *UIML* interface description" which represents the same.

In Phase 4 - *Rendering UIML*, we used the actual *UIML* rendering engines, from Harmonia[3], which supports

[3]http://www.harmonia.com

several platforms, with several programming languages (JAVA, HTML, WML, VoiceXml, etc.), rendering widgets and events specified in *UIML* <presentation> elements. The goal was to generate UI from an UIML specification.

For instance, in order to convert our table specification (*UIMLSpecTab*) to our base specification (*UIMLSpec*), we created the function *TabUIML2UIML*. Considering this, the expression

$$UIMLSpecTab`TabUIML2UIML(UIMLSpecTab`t)$$

will result in an *UIML* element corresponding to our table t.

Once in the *UIML* code, the following shell commands will respond for the rest of the process:

1. $u2h\ table.uiml\ table.html$

   Generates $HTML$ code for browser platforms. The render $u2h$ works over $HTML$ vocabulary.

2. $u2ji\ table.uiml\ table.java$

   Generates $Java$ code associated to $UIML$ descriptions, ready to be used on Java platforms. The $u2ji$ render works over $Java$ vocabulary.

3. $u2w\ table.uiml\ table.wml$

   Generates WML code for WAP devices. The render $u2w$ works over WML vocabulary.

7 Conclusions and Future Work

The main focus of this work is the user interfaces graphical design which support current software applications. For the sake of rigor, it addresses the application of formal methods to recent UI markup language support technologies. The application of formal semantics techniques to user interface development, imposes scientific rigor on the whole development process: specification, validation and transformation.

Despite the existence of several research projects focused on this, there are still questions which remain unanswered. One of these questions is as basic as follows: are markup languages, even developed with important objectives, the best idea to support this process? Further than a portable and platform independent description technique, XML subscribes to an enormous vocabulary set, which demands parser and validation mechanisms.

Our *UIML VDM-SL* specification leads us to believe that there is room for some conceptual simplification. This conclusion is sustained by observing the achieved abstract specification, where several replicated elements and repeated patterns can be identified.

Considering this, an interesting path for future work is that of UIML's refactoring process towards a simpler specification.

Another interesting focus topic for future developments is that of creating a formal *Visual Component Library - VCL*. Our table specification is but the beginning of one such repository of useful graphical interaction objects (IO). The aim should be the construction of new interfaces by reusing existent components. Once made available, a VCL would contribute to answering another quite common question: "What is the best IO to represent this particular piece of data?", Ideally, a "matching" process should become available for searching the library and retrieving all adequate components.

As a matter of fact, our research has included some work in these two vectors - *UIML* abstract syntax refactoring and visualization. However, the outcome cannot be regarded as finished work. Despite their incompleteness, perhaps our results can still be useful to anyone wishing to pursue them.

8. ACKNOWLEDGEMENTS

I would like to thank my supervisor Professor José Nuno Oliveira, member of DI (Department of Informatics, Minho University[4]), who encouraged all formal methods research and initiatives at the University, for his useful support and advice during this work. I am also grateful to Marc Abrams and James Helms *Harmonia* members, for their enthusiastic support along this research as well for the availability of their *UIML* supporting tools.

References

[Brad 98] Myers Brad. A brief history of Human Computer Interaction Technology, 1998. http://citeseer.nj.nec.com/myers98brief.html.

[Harmonia 02] Inc. Harmonia. User Interface Markup Language UIML Specification. Technical report, Harmonia Inc., 2002.

[Oliveira 03] J. Nuno Oliveira. An introduction to Data Refinement. Formal Methods II, 2003.

[Pfaff 85] G.E. Pfaff. User Interface Management Systems: Proceedings of the Seeheim Workshop, 1985.

[Phanouriou 00] Constantinos Phanouriou. UIML: A Device-Independent User Interface Markup Language, 2000.

[4] http://www.di.uminho.pt