Yugoslav Journal of Operations Research
23 (2013) Number 3, 327-341
DOI: 10.2298/YJOR121101007J

LANGUAGES FOR MODEL-DRIVEN DEVELOPMENT OF USER INTERFACES: REVIEW OF THE STATE OF THE ART

Mlađan JOVANOVIĆ
Faculty of Electrical Engineering, University of Belgrade, Serbia
mladjan@rcub.bg.ac.rs
Dušan STARČEVIĆ
Faculty of Organizational Sciences, University of Belgrade, Serbia
starcev@fon.bg.ac.rs
Zoran JOVANOVIĆ
Faculty of Electrical Engineering, University of Belgrade, Serbia
zoran@rcub.bg.ac.rs

Received: November 2012 / Accepted: February 2013

Abstract: In model-driven user interface development, several models are used to describe different aspects of user interface when level of detail varies. The relations between the models are established through model transformations. The Model Driven Engineering (MDE) approach has been proposed in software engineering domain in order to provide techniques and tools to deal with models in the automated way. In this paper, we will review existing user interface languages that gain wider acceptance, and discuss their applicability for model-driven user interface development.

Keywords: User interface, model-driven development, user interface description languages, transformation languages.

MSC: 68N15, 68N19, 68T35, 68U35.

1. INTRODUCTION

Model-driven user interface development (MDUID) uses models to describe static and dynamic system properties on different levels of abstractions, and applies

transformations of one model to another. Model Driven Architecture (MDA) is an official proposal for system specification and interoperability based on the use of hierarchically organized formal models [1]. However, Human-Computer Interaction (HCI) community still has not reached general consensus on models for engineering user interfaces. Therefore, classical MDA-approaches have been lacking solid foundation for user interface development, although some UI development methodologies have been aligned with the OMG (Object Management Group) standards, as for example the UML-based architecture Wisdom [2] and an MDA compliant environment around UsiXML-based tools [3]. The Cameleon Reference Framework [4] proposes types of UI models for different levels of abstractions, namely the task model, the abstract user interface (AUI) model, the concrete user interface (CUI) model and the final user interface (FUI). As in the MDA, transformation tools are used to move from one layer of abstraction to another or to adapt these models to different contexts of use. In practice, we can find a number of approaches for model-driven user interface development. Each of them is specific in terms of involved models, their underlying structure and tools for their manipulation. Here we can speak about different languages used for UI description and transformations between UI models, namely UIDL (User Interface Description Language) and UITL (User Interface Transformation Language).
For this reason, we will review several technologies and discuss their applicability for MDUID. This paper proposes a comparative survey of languages for MDUID with the emphasis on UITLs. The goal of the paper is not to identify the best technology for UI development, but to analyse existing languages and help UI designers to choose the most suitable technology in the specific context of development.

The paper is organized as follows. Section 2 reviews existing UIDLs. Section 3 describes previous works in the field of MDUID and gives a comparative survey on UITLs. Section 4 concludes the paper.

2. USER INTERFACE DESCRIPTION LANGUAGES

User interface description languages are closely related to programming styles and can be divided in two categories – declarative and imperative. Declarative languages indicate *what* to show on user interface, while imperative tell *how* to show user interface component. The key advantage of declarative programming is that you just say what you want, and leave it to an automatic tool to figure out how to produce it. That contrasts with conventional imperative programming, where the programmer has to say, step-by-step, how to reach the desired state. Current declarative languages commonly take forms of markup specifications such as HTML, XML and related languages. Imperative languages include conventional programming languages such as procedural, object-oriented and script languages. Using particular user interface description language is determined by the purpose and application domain of the software system, and also depends on individual preferences of user interface developer. Web environment usually imposes usage of declarative languages. This assumes the existence of an automatic algorithm, built into every web browser, that constructs the user interface from particular language specification. Desktop environments mainly use imperative languages for user interface construction. The latest trend in development of user interface description languages is a hybrid solution where presentation elements of the interface are described by using declarative constructions, while behavior is defined in imperative blocks. In this way, it is

possible to achieve a consistent view and behavior across various Web and desktop platforms. Two most prominent examples of hybrid approach to user interface description language specification are Adobe FLEX and Microsoft WPF (Windows Presentation Foundation) technologies.

Several UIDLs have been developed in last two decades. Most of them are XML-based. As pointed out in [5], the main reasons for emergence of these UIDLs are:

- Capturing the requirement for user interface as an abstract definition which remains stable across variety of platforms. Stability refers to interaction semantics.
- Design of a single user interface for multiple devices and platforms.
- Improvement of reusability of user interface.
- Support to evolution, extensibility and adaptability of user interface.
- Automated generation of a user interface code.

In the rest of the section, we give a short description of user interface description languages used in both commercial and scientific research domains.

2.1. Declarative languages

Current subsection gives review of the existing declarative languages for user interface description.

HYPERTEXT

The basic idea behind hypertext is to create a document read not necessarily linearly but with special connections (i.e. hyperlinks) built in the document's content that enable immediate shift to various resources such as pictures and other documents. However, from its beginning hypertext did not attain widespread use until the advent of the World Wide Web (WWW) system by Berners-Lee and HTML (HyperText Markup Language) in 1990s. HTML was the first commercial hypertext language specification. Some of the elements of the success of the WWW are the simplicity and accessibility of the HTML language used to design web pages, the ease of making pages accessible on the Web, and the embedding of pictures with the text. Convenient tools, called Web browsers, have built in mechanisms for Web page rendering providing access to many of the existing network resources. Programmers use a very simple HTML textual specification to design Web pages. Since the initial proposal up to now, various improvements of the HTML specification have been made in order to achieve more flexibility and capabilities in authoring of web pages.

XUL

The XML User Interface Language (XUL) is an XML-based markup language for description of user interfaces. It is mainly used by the Mozilla Foundation in their products like the Firefox browser or Thunderbird mail client. However, it is increasingly popular in the area of Web applications. For desktop application Java XUL rendering components exist, which allow the usage of XUL for Java applications. XUL is used by several projects involved in development of context-sensitive user interfaces for mobile [6].

XIML

XIML (eXtensible Interface Markup Language) is a language for presentation and manipulation with data related to human-computer interaction, i.e. interaction data [7]. These data include user tasks, domain objects, dialog elements and presentation elements. In this way, it is possible to create user interface models on different abstraction levels. UIFin [8] presents user interface development tool which uses XIML. Tool employs declarative models of user interfaces. User interface development models are separated in two layers – design layer and executable layer. Design models are written in XIML. These models describe semantics of human-computer interaction. Using XSL transformation tools, they are converted into executable models written in commercial declarative languages such as MXML and XAML. So, it is possible to integrate the tool with existing development environments like Adobe Flex Builder and Microsoft Visual Studio.

UIML

UIML (User Interface Markup Language) is declarative user interface language aimed to create platform-independent user interfaces [9]. Platform independence realization is based on using CSS (Cascading Style Sheets) libraries. Language specification assumes the existence of tags for connecting with application logic components. Since it was initially proposed, the language has been constantly improved in order to increase platform independence [10]. MONA project [11] presents software platform for development of multimodal services for mobile devices. Generic user interface descriptions are written in UIML, and then transformed in platform-specific languages such as HTML and VoiceXML.

UsiXML

USIXML (USer Interface eXtensible Markup Language) [12] is a user interface description language built to develop context-sensitive user interfaces. The language is described with several metamodels where each describes particular aspect of user interface according to CAMELEON reference framework for development of context-sensitive user interfaces [4]. This way, it is possible to specify models of user interfaces on different abstraction levels. In addition, transformation model is introduced to enable establishing relations between different models of user interfaces. For the sake of clarity and simplicity, here we describe concrete user interface model. This model presents hierarchical decomposition of concrete interaction objects (CIOs) and relations among them. CIO is defined as a user interface entity perceived by users. Each CIO can be further refined into subtypes suitable for specific modalities (such as visual or auditory). Transformations between different models of user interfaces are realized as graph transformations [29]. In order to enable seamless tool support in development process of context-sensitive user interfaces, a number of tools based on USIXML has been proposed (Fig. 1)

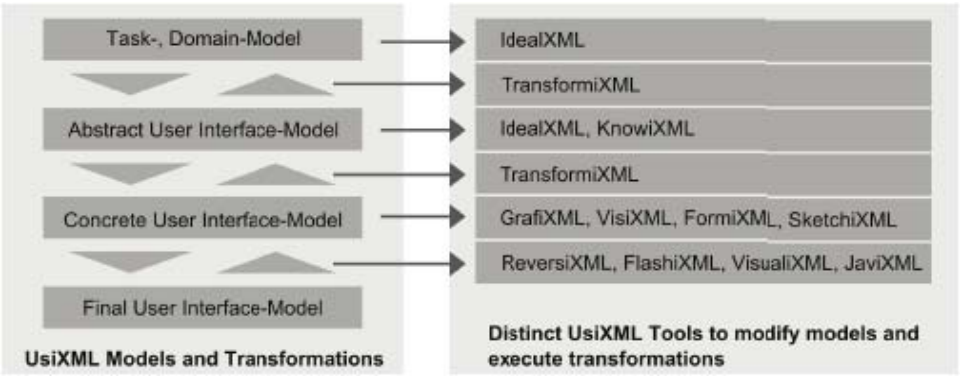

Figure 1. *UsiXML user interface development tool. Taken and adapted from* [3].

COMMERCIAL DECLARATIVE LANGUAGES

MXML (Macromedia eXtensible Markup Language) [13] is a user interface description language developed at Adobe company, and intended to be used in RIA (Rich Internet Applications). Main features of RIA system are rich multimedia content and intensive interaction with end users. MXML is used in combination with ActionScript object-oriented language for definition of application logic. This technology has broad user community, and is used in large number of research and commercial projects [14].

XAML (Microsoft eXtensible Markup Language) is a user interface description language developed used in Microsoft's technologies suite [15], [16].

DOMAIN SPECIFIC DECLARATIVE LANGUAGES

PUC (*eng. Personal Universal Controler*) presents an abstract device for remote control of household appliances [17]. This is accomplished by appropriate user interface that enable communication between devices. Prior to communication, description of aplliances' functionalities is provided using specific language. Based on given description, compatible user interface is automatically generated and used to send control signals to remoted device , and to get feedback information about device's status. In order to describe functionalities of devices, which can be found in household, specification of declarative language is developed. This specification covers around thirty most common household devices, including mobile and sensing devices controlled with voice commands. Upon given abstract devices' descriptions, graphical and speech interfaces are generated, which can be deployed on handheld computers, smartphones and desktop computers. The details of language specification can be found in [17].

Latest SOA (Service-Oriented Architecture) environments integrate rich set of interactive software applications. These environments comprise large number of interactive devices with various applications encapsulated and presented to end users as a set of Web services. The appearance of Web services for interactive applications requires development of suitable interfaces to access them. MARIA (Modelbased lAnguage foR Interactive Applications) is declarative language for user interface description of Web

services [18]. The language is used for the development of multiplatform user inrerfaces while integrating Web services' annotations. It is model-based and has modular structure. This implies the existence of platform-independent metamodel for abstract description of user interface and a number of derived platform-specific metamodels. Abstract description is independent of interaction resources, while concrete description is determined by modality type (in concrete case visual or auditory). Abstract and concrete user interface models are described with XML schemas. Transformations are realized in XSLT, and supported by a set of visual designers. More details about the language can be found in [18].

2.2. Imperative languages

This section discusses imperative languages for user interface description. The following text gives brief and comprehensive survey of these languages and the related tools classified as:

- Window tools and event languages,
- Component systems,
- Object-oriented languages.

The proposed survey presents succinct and modified user interface tools history appearing elsewhere [19].

WINDOWS TOOLS AND EVENT LANGUAGES

One of the main reasons why windows user interface paradigm has been successful is because it helps to manage scarce resources. These include both resources of the computer (e.g., limited number of pixels) and human perceptual and cognitive resources (such as limited visual field, and attention of a user). By allowing the user to control a set of overlapping display objects, the display can be made to suit the focus and attention of the user's task. By paying careful attention to the properties of humans, overlapping windows effectively overcome the limited resources of both available screen space, and human attention and visual field. A number of windows user interface tools provide facilites for drawing and displaying update (an output model) and accepting user input (an input model). Early forms of these tools evolved into interactive graphical tools that allow interactive components to be placed by using a mouse to create windows and dialog boxes. Examples include Microsoft Visual Studio, NetBeans, Eclipse and others. An important reason for the success of interactive user interface tools has been their use of visual means to express concepts of user interface. By allowing interactive specification of user interface elements (rather than conventional programming code), aspects of interface implementation are made available to those who are not programmers.

Event programming models assume that the occurrence of each significant event (such as user input action) is placed in an event record data structure (often simply called an event). These events are then sent to individual event handlers which contain the code necessary for a proper respond to concrete action. Event languages have been successful because they match well to the direct-manipulation graphical user interface style. These systems generate events for each user action with the mouse and keyboard, directed to the appropriate application which then must respond.

COMPONENT SYSTEMS

Component systems follow the idea of creating software systems by dynamically combining independently written and compiled components. Later on, this idea was realized in commercial systems such as Sun JavaBeans and Microsoft OLE and ActiveX. One of the main reasons for wide adoption of component model is its ability of modular application development and usage of existing components, while providing complex functionalities and integration capabilities.

OBJECT-ORIENTED LANGUAGES

From its origins, development of object-oriented programming paradigm has been strongly related to user interface research. Interactions between these fields are mutual. For example, Smalltalk programming language is developed with the aim to ease the development of interactive graphical tools; C++ gain popularity with the need for programming of graphical user interfaces for Windows operating system. Object-oriented programming naturally fits in with development of graphical user interfaces. User interface components, like buttons and menus, are treated as visible objects with their own state, as well as operations enabling state change. This corresponds to object definition in object-oriented systems.

3. MODEL-DRIVEN USER INTERFACE DEVELOPMENT

Introducing variety of new technologies leads to more and more complex interactive systems design. In order to describe these interactive systems, HCI domain uses specific models and tools. On the other hand, the MDE approach provides techniques and tools for dealing with models in an automated way in order to generate executable software. MDE approach is based on models, meta-models and model transformations and aims to increase productivity of software development. In this section, we contribute comparative survey of UITLs. However, prior to this, the overview of the history of model-driven user interface tools is given.

3.1. Model driven user interface tools

In the early 1980's, the concept of a user interface management system (UIMS) was an important area in user interface software research community [19]. A UIMS allows designers to specify interactive behavior in a high-level UIDL that abstracts the details of input and output devices. This specification would be automatically translated into an executable program or interpreted at run time to generate a standard implementation of the user interface. The choice of a UIDL model and methods is a key ingredient in the design and implementation of a UIMS. The goal of user interface management systems was not only to simplify the development of user interfaces but also to promote consistency across applications as well as the separation of user interface code from application logic. However, with standardization of user interface elements, in the late 1980's, on the desktop paradigm, user interface developers were seeking effective and ergonomic mechanisms to control the user interface look and behavior. Thus, although a promising concept, the UIMS approach has been challenged in practice [19].

Subsequently, in the last decade, proliferation of new devices and HCI techniques required next generation user interfaces. UI developers were facing new challenges similar to those GUI developers had been faced with in the early 1980's. Thus, as part of the user interface research community effort to address these challenges, the concept of MDUID reemerged as a promising approach. In addition, significance of model-driven approaches came from practical reasons. Large scale user interfaces composed of different technologies were difficult to implement and maintain. In that regard, detailed models of user interface can benefit implementation and maintenance of processes. Another important incentive to use model-driven principles in user interface software research was the need for device-independent user interfaces. In this sense, significant efforts were made in constructing tools for automatization of user interface development for various platforms.

When talking about MDUID, we can notice several generations of these systems. First generation followed the principle of abstracting components in graphical user interfaces (GUIs). At that time, UI development was based on identifying relevant aspects of visual type of communication [20]. Next generation tried to include end user factors in user interface development process. This was achieved by describing HCI semantics at high abstraction level where task models were introduced as formal descriptions of user goals when interacting with software system [21]. The emergence of new devices and interaction techniques, especially mobile ones, brought new challenges facing user interface research community. An important issue was to preserve usability of user interface across various platforms and interaction styles. A number of researches were devoted to multiplatform user interfaces by identifying relevant information contained in the corresponding models (and languages) [10], [12], [22], [18]. Model-based approaches have initiated official recommendations in the field[1], as well as their adoption by the industry[2]. Nevertheless, the existence of specific, non-standardized languages and tools hampers their integration in standard methodologies for software development.

Current MDE approaches mostly rely on UML (Unified Modeling Language) notation to describe models [23]. UML is widely adopted industrial standard used in a large number of software engineering fields and with rich tool support. On the other hand, HCI field has brought specific notations for describing user interfaces such as task models before UML had been proposed. With the advent and the acceptance of UML, existing notations for user interface descriptions were shaped in UML setting. Thus far, several UML models for user interface description were introduced [24], [25], [26].

3.2. User interface transformation languages

Model-driven engineering of user interfaces assumes that various models describe different aspects of user interface. Relations between these models are established through model transformations. In this way, development of user interfaces can be seen as transformation chain that starts with models at high level of abstraction and ends with executable versions of user interface. An extensive taxonomy of model transformation approaches has been proposed in [27]. Variability of semantics between

---

1 W3C Working Group For Model-based User Interfaces, http://www.w3.org/2005/Incubator/model-based-ui/

2 Working Group NESSI NEXOF-RA IP, http://www.nexof-ra.eu/

different models, their formats and tools caused various transformational approaches in context of model-driven development of user interfaces. Some of them operate directly upon models, while others work with their derived formats. Some are integrated in models, while others are applied externally. Finally, some of them are editable and modifiable, while others are integrated in tools and cannot be modified. Therefore, in the following sections, we discuss existing transformation tools used for model-driven development of user interfaces. At the end of the section, we give comparison of described transformational approaches against selected set of criteria.

GRAPH TRANSFORMATIONS

GT (Graph Transformations) presents a formal, declarative approach for transformations of models with a structure of directed graph [28]. UsiXML is a candidate language to use this type of transformation. The models formed with UsiXML are based on graphs and therefore, the model mappings of UsiXML are specified with graph transformations consisting of a set of transformation rules [12], [29]. Each rule consists of a Left Hand Side (LHS) matching a graph G, a Negative Application Condition (NAC) not matching G and a Right Hand Side (RHS) which is the result of the transformation.Transformation is performed by searching LHS templates in source model and replacing found matchings with RHS, while taking into account NAC. The main limitation of the approach is that it requires models with underlying structure of graph.

ATL

ATL (Atlas Transformation Language) is a hybrid language for transformations of UML models [30]. In this sense, the user can choose whether to use pure declarative features of language, or to employ additional imperative. The declarative approach is realized by the system of matching rules, where a source pattern is described through a set of source types and constraints on provided types. The target pattern is specified in similar way by specifying a set of target types together with a set of bindings used to initialize the target types' features. Declarative aspect offers pretty straigthforward way to specify transformation rules. However, it may be difficult to specify more complex rules. In this case, ATL provides imperative constructions organized in action blocks. These blocks can be added to declarative rules, or even call external code for transformation logic. ATL is a good candidate for model transformations according to the following arguments: an open-source software, large user community, a solid developer support and rich knowledge base of model transformations [31].

TXL

TXL is designed as a general purpose transformation language [32]. Among other things, it allows transformations of programming languages since it is not confined to any source or target format. In general, the language comprises the following specifications:

- Specification of a structure to be transformed based on grammars.
- Specification of transformation rules based on source/target replacement rules.

TXL is intended to transforming models which have syntax tree structure. This is the case of most of the programming languages.

4DML

4DML (Four-Dimensional Markup Language) is a transformation language originally proposed for Web content adaptation to users with special needs [33]. Unlike TXL, 4DML allows transformation of a model with matrix structure. Transformation rules are specified in declarative style by defining source/target pattern matchings. Usage of 4DML may be complicated in case of models organized as trees or graphs, since their conversion to matrix-like structure is not a straightforward task.

UIML Transformations

An important feature of UIML is its capability to define connections to the backend logic, and to provide mappings to other UIML instances or target languages. Therefore, language specification includes transformation features that define explicit mappings of UIML primitives to target format constructs. Separate section defines connections to the application logic. In particular, specification prescribes mappings to VoiceXML and HTML formats. However, these mappings are not necessarily restricted to XML formats, but may also be defined for other languages, e.g., Java. Considering UIML's mapping technique based on explicit matches to target format primitives, it can be seen as declarative. The obvious advantage of the UIML approach is that user interface definition and transformation are specified in the same language. On the other hand, transformation rules are too simple to support more complex transformation tasks.

XSLT

XSLT (eXtensible Stylesheet Language Transformations) is a language for transforming the XML input to textual (in most case XML) output [34]. The input of an XSLT is a XML document. The output is XML, or other textual format. In this way, XSLT can be used to generate documents written in languages different from XML. Transformation is comprised of templates rules. Each rule includes matching pattern, construction element (template) and additional optional attributes. Matching pattern consis of expressions evaluated against currently processed node of input XML document. Transformation executes starting from document's rote node and continues until each node is traversed and processed according to specified rules. When a pattern is matched, the template is recursively executed and target element is generated. Considering rules processing, XSLT provides constants, variables and literals together with conditions, iterations, recursion and sorting as control structures. In addition, XSLT offers a powerful set of built-in string functions for advanced text processing. While the XSLT transformation mostly follows declarative style, it also allows imperative constructs such as conditions, iterations and recursion. Therefore, the language can be considered to be a hybrid.

GAC

GAC (General Adaptation Component) language has been proposed in order to improve adaptation of Web components [35]. Unlike other transformation languages, GAC works with contextual data to control the adaptation process. In this regard, GAC underlying architecture is based on RDF (Resource Description Framework) language for declarative description of Web resources. The language is able to process only HTML and XML-based documents. In this sense, we can talk about adaptation rather than

transformation process. GAC configuration embraces set of adaptation rules with associated conditions. Adaptation rules enable update, deletion, substitution and separation of source model's elements. GAC approach can be considered to be imperative, since the rules provide explicit intsructions and operations to execute when condition holds.

RDL/TT

RDL/TT (Rule Description Language for Tree Transformation) is a language for context-sensitive transformations of XML-based user interfaces [36]. In case of RDL/TT, term context-sensitivity primarily refers to adaptation of Web content to various devices. The language resembles Java sintax to define transformation rules which handling XML document's tree. Transformation rules follow source/target pattern matching style. An important feature is the use of variables for storing contextual information, thus allowing different transformation directions dependant on varying preferences or target devices. Transformation rules are specified in an imperative manner providing control structures like loops and branches together with a set of extensible predefined functions.

TRANSFORMATION LANGUAGES REVIEW

Table 1 gives comparative survey of described languages for UI models transformation. At first, the programming model is considered. Distinction between declarative and imperative style is important from developer's familiarity viewpoint. However, it is not always possible to make a clear distinction between these approches. In this way, many languages combine both styles in order to increase expressivenes and gain UI developers' acceptance. Further, we examined the UIs' transformation abstraction levels. In other words, we looked at capabilities of model-to-model (M2M) and model-to-code (M2C) transformations. Another important aspect is the ability of complex transformations with nonlinear mappings between elements of source and target models. The capability to extend transformation language while keeping it's underlying syntax and semantics is of the key importance in UI development for different domains of use. In the end, existing tool support may be practical reason to decide whether to use a specific transformation language.

Table 1. Review of user interface transformation languages.

Language / Property	ATL	GT	TXL	4DML	XSLT	GAC	UIML	RDL
Declarative	+	+	+	+	+	-	+	-
Imperative	+	-	-	-	+	+	-	+
Model-To-Model Transformation	+	+	+	+	+	+	-	+
Model-To-Code Transformation	+	-	+	+	+	-	+	+
Complex Transformations	+	+	+	+	+	+	-	+
Extensibility	+	-	-	-	-	-	-	+
Tool support	+	+	-	-	+	+	-	+

Looking at M2M transformation capabilites, we can generally claim that each language is capable of performing it. However, only ATL, GT and XSLT are really designed for it. TXL and 4DML require transformed models conforming to specific structure, i.e. BNF grammar and matrix structure, respectively. UIML is not designed to be a transformation language, but explicit mappings to specific platforms are part of a anguage specification. With respect to code generation ability, we can denote TXL, 4DML, UIML, XSLT and ATL. In general, ATL is not designed to generate code; however, certain solutions have already been proposed [37]. With the exception of UIML, all other languages support complex mappings. UIML provides linear one-to-one mappings. Considering the ability to extend the language with additional functionalities, ATL and RDL are good candidates. ATL is described with UML metamodel. In this way, language can be easily extended by assigning additional concepts to existing metamodel. With regard to developers' facilities, ATL distinguishes from other approaches since its use has been extensively supported in EMF (Eclipse Modeling Framework) suite.

In summary, the choice of the UITL largely depends on the models, their representation and the target application‘s domain. Sometimes, a combination of transformation approaches is preferable, where several transformation systems are used at different abstraction levels. Many of the UML tools have the ability to export models in XMI (eXtensible Metadata Interchange) format used for information exchange [38]. This can improve integration at tool level during model-driven user interface development process.

4. CONCLUSION

The goal of this paper is to give a survey of user interface languages in order to help the HCI community chose some of UI technology. Considering existing results in the UI domain, there is a growing need for the MDE approach and languages. With respect to modeling, UI designers use a lot of domain specific models such as task models. However, these models are more descriptive and less productive (concentrated on generative power). Transformations allow to produce new models from existing ones, but also to generate code from models. We can consider two types of transformations, those that generate different models (more general, a file conforming to specific structure that can be manipulated by design tool) and those that produce code (a text file that can be compiled or interpreted by running platform).

Based on these needs, we gave a comparative survey of several UITLs with regard to selected set of criteria. The survey was organized according to the criteria we identified as important for MDUID with the purpose of providing a comprehensive overview of UITLs. These criteria include programming style (declarative vs. imperative), M2M transformation capability, M2C transformation capability, ability to design complex transformations, language extensibility, and tool support.

With respect to the variety of the development tasks, it is difficult to definitely recommend or dismiss one of the compared languages, which provide different strengths and weaknesses for different applications. Nevertheless, some conclusions can be drawn from this comparison.

In terms of UIDLs, there are a number of languages which allow describing domain-specific user interfaces. In terms of UITLs, there is no standard language to be used, but it is important to know the type of transformations language supports, and to

specify if language can be extended in order to meet the requirements of application's domain. Moreover, it is important to know the format of the generated models in order to identify the kind of tools to manipulate them. In this sense, the user interface research community has to incorporate the proposed standards that MDE is using nowadays. We hope this survey will be useful to any HCI designer who wants to select the most suitable UI technology according to the specific requirements.

REFERENCES

[1] Bezivin, J. "From object composition to model transformation with the MDA," *Proceedings of the TOOLS Conference, Santa Barbara*, CA, USA, 2001, 350–354.

[2] Nunes, N.J., and Cuhna, J.F., "Wisdom: a software engineering method for small companies", *IEEE Software*, 17 (5) (2000) 113-119.

[3] Heinrich, M., Winkler, M., Steidelmuller, H., Zabelt, M., Behring, A., Neumerkel, R., and Strunk, A., "MDA Applied: A Task-Model Driven Tool Chain for Multimodal Applications", *Proceedings of the 6th international workshop on Task models and diagrams TAMODIA '07*, Toulouse, France, November 7-9, LNCS, 4849 (2007) 15-27.

[4] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J., "A Unifying Reference Framework for Multi-Target User Interfaces", *Interacting with Computers*, 15 (3) (2003) 289–308.

[5] Paternò, F., and Santoro, C., "UIDLs for Ubiquitous Environments", *ACM CHI 2008 Workshop Proceedings on User Interface Description Languages for Next Generation User Interfaces*, April 6th, Florence, Italy, 2008.

[6] Butter, T., Aleksy, M., Bostan, P., and Schader, M., "Context-aware User Interface Framework for Mobile Applications", *Proceedings of the 27th IEEE International Conference on Distributed Computing Systems Workshops ICDCSW*, June 25-29, Toronto, Canada, 2007.

[7] Puerta, A., and Eisenstein, J., "XIML: A Common Representation for Interaction Data", *Proceedings of the ACM 6th International Conference on Intelligent User Interfaces IUI'02*, January 13-16, San Francisco, California, USA, 2002, 214-215.

[8] Puerta, A., and Hu, M., "UI Fin: A Process-Oriented Interface Design Tool", *Proceedings of the 13th ACM international conference on Intelligent user interfaces IUI'09*, Sanibel Island, Florida, USA February 08 - 11, 2009, 345-354.

[9] Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., and Shuster, J. E., "UIML: An appliance-independent XML user interface language", *Proceedings of the 8th International World Wide Web Conference*, Toronto, Canada, May 11-14, 1999, 617-630.

[10] Helms, J., and Abrams, M., "Retrospective on UI description languages based on eight years' experience with the user interface markup language (UIML)", *International Journal on Web Enginering Technology*, 4 (2) (2008) 138–162.

[11] Simon, R., Jank, M., and Wegscheider, F., "A generic UIML vocabulary for device- and modality independent user interfaces", *Proceedings of the World Wide Web Conference WWW 2004*, May 17–22, New York, USA, 2004, 434–435.

[12] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez-Jaquero, V., "UsiXML: A language supporting multi-path development of user interfaces", *Proceedings of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th International Workshop on Design, Specification, and Verification of Interactive Systems (EHCIDSVIS' 04)*, Hamburg, Germany, July 11-13, 2004, 200–220.

[13] Coenraets, C., 2004, An overview of MXML: The Flex markup language. http://www.adobe.com/devnet/flex/articles/paradigm.html

[14] TourDeFlex, 2010, http://www.adobe.com/devnet-apps/flex/tourdeflex/web/

[15] Microsoft 2006, XAML http://windowssdk.msdn.microsoft.com/en-us/library/ms747122.aspx

[16] Bishop, J., "Multi-platform user interface construction: a challenge for software engineering-in-the-small", *Proceedings of the 28th international conference on Software engineering, ICSE '06*, May 20–28, Shanghai, China, 2006, 751-760.

[17] Nichols, J., and Myers, B., "Creating a Lightweight User Interface Description Language: An Overview and Analysis of the Personal Universal Controller Project", *ACM Transactions on Computer-Human Interaction*, 16 (4) (17) (2009) 37 .

[18] Paterno, F., Santoro, C., and Spano, L. D., "MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments", *ACM Transactions on Computer-Human Interaction (TOCHI),* (16) (4) (19) (2009) 30.

[19] Myers, B., Hudson, S., and Pausch, R., "Past, present, and future of user interface software tools", *ACM Transactions on Computer-Human Interaction*, (7) (1) (2000) 3–28.

[20] Foley, D., and Noi Sukaviriya, P., "History, results, and bibliography of the user interface design environment (UIDE), an early model-based system for user interface design and implementation", *Proceedings of Design, Verification and Specification of Interactive Systems (DSVIS'94)*, Bocca di Magra, Italy, June 1994, 3–14.

[21] Paterno, F., and Mancini, C., "Model-Based Design of Interactive Applications", *ACM Intelligence*, Winter 2000, 27-37.

[22] Mori, G., Paterno, F., and Santoro, C., "Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions", *IEEE Transactions on Software Engineering*, (30) (8) (2004) 507-520.

[23] OMG Unified Modeling LanguageTM (OMG UML), Infrastructure Version 2.3, May 2010 http://www.omg.org/spec/UML/2.3/Infrastructure

[24] Nunes, N.J., and Cuhna, J.F., "Wisdom: a software engineering method for small companies", *IEEE Software*, 17 (5) (2000) 113-119.

[25] Da Silva, P.P., and Paton, N., "User Interface Modeling in UMLi", *IEEE Software*, 20 (4) (2003) 62-69.

[26] Sottet, J-S., Calvary, G., Favre, J-M., Coutaz, J., Demeure, A., and Balme, L., "Towards Model Driven Engineering of Plastic User Interfaces", *Satellite Proceedings of the ACM/IEEE 8th International Conference on Models Driven Engineering Languages and Systems, MoDELS 2005*, Montego Bay, Jamaica, LNCS, 2005, 191–200.

[27] Czarnecki, K., and Helsen, S., "Feature-Based Survey of Model Transformation Approaches", *IBM Systems Journal*, 45 (3) 2006, 621-645.

[28] Czarnecki, K., and Helsen, S., "Classification of Model Transformation Approaches", *ACM OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, Anaheim, CA, USA, 2003, 1-17.

[29] Stanciulescu, A., Vanderdonckt, J., and Mens, T., "Colored graph transformation rules for model-driven engineering of multi-target systems", *Proceedings of the third ACM international workshop on Graph and model transformations GRaMoT '08*, Leipzig, German, 2008, 37-44.

[30] Jouault, F., and Kurtev, I., "Transforming Models with ATL", *Proceedings of the ACM/IEEE 8th International Conference on Models Driven Engineering Languages and Systems, MoDELS 2005*, Montego Bay, Jamaica, LNCS, 2005, 128–138.

[31] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I., "ATL: A model transformation tool", *Science of Computer Programming*, Elsevier Publishing, 72 (1) (2008) 31–39.

[32] Cordy, J.R., "The TXL Source Transformation Language", *Science of Computer Programming*, 61 (2006) 190–210.

[33] Brown, S.S., "Conversion of notations", *Technical report*, University of Cambridge, 2004.

[34] Kay, M.: XSL Transformations (XSLT) Version 2.0, W3C Working Draft. World Wide Web Constortion, 2002.

[35] Fiala, Z., and Houben, G.-J., "A Generic Transcoding Tool for Making Web Applications Adaptive", *Short Paper Proceedings of the 17th Conference on Advanced Information Systems Engineering CAiSE'05*, Porto, Portugal, LNCS,161, 2005

[36] Schaefer, R., Mueller, W., and Dangberg, A., "RDL/TT-A Description Language for the Profile-Dependent Transcoding of XML Documents", *Proceedings of the First International ITEA Workshop on Virtual Home Environments*, Paderborn, Germany, 2002.

[37] MDE Case Studies http://soft.vub.ac.be/soft/research/mdd/casestudies

[38] OMG Meta Object Facility (MOF) 2.0 XMI Mapping Specification v2.4, August 2011 http://www.omg.org/spec/XMI/2.4.1/