Multi-Paradigm Modelling and Synthesis of User Interfaces

Denis Dubé and Hans Vangheluwe
School of Computer Science
McGill University
Montréal, Québec, Canada
denkkar@gmail.com, hv@cs.mcgill.ca

ABSTRACT

In this article, model-based design and synthesis of reactive user interfaces is presented as a particular application of Computer-Automated Multi-Paradigm Modelling (CAMPaM). Multi-paradigm modelling acknowledges the need to model at different levels of abstraction, using appropriate formalisms. It also gives transformations first-class model status. In the CAMPaM UI development process, a class of user interfaces is modelled. This includes models of the abstract syntax of the user interface, of the concrete visual syntax of the user interface (including layout) and of the semantics of the application (its reactive behaviour). From these models, an interactive modelling environment is synthesized. This environment allows the modeller to experiment (analyze, simulate) with different instances in the modelled class of user interfaces. Once a single element of the set of possible user interfaces is chosen, the final UI application is synthesized. This process will be demonstrated by means of a digital watch application. Code is synthesized for execution within a web browser using an AJAX client-server architecture.

1. INTRODUCTION

Recently, model-based approaches to complex (software) systems development have gained popularity. Putting models (rather than code) central in the development process does indeed offer many advantages. It raises the level of abstraction; it enables formal analysis (through model checking for example), simulation (for performance analysis), as well as automated, consistent code synthesis for multiple target platforms. Modelling complex systems is a difficult task, as these systems often have components and aspects which cannot be described in a single formalism (such as Class Diagrams, Statecharts, or Petri Nets). User interfaces are a very pertinent example of such complex systems, in particular as there are many facets to their structure and behaviour. Multi-Paradigm Modelling [9] captures the notions that (1) models may have components described in different formalisms, and span different level of abstraction and that (2) model transformations are used to map models onto domains and formalisms where certain questions can be easily answered. The sequel demonstrates by means of a digital watch example how CAMPaM principles can be consistently applied to the design and synthesis of User Interfaces.

2. MODELLING LANGUAGES

The two main aspects of any model are its *syntax* (how it is represented) and its *semantics* (what it means). The syntax of modelling languages is traditionally partitioned into *concrete* syntax and *abstract* syntax. In textual languages for example, the concrete syntax is made up of sequences of characters taken from an alphabet. These characters are typically grouped into *words* or *tokens*. Certain sequences of words known as *sentences* are considered to the language. The (possibly infinite) set of all valid sentences is said to make up the language. Costagliola et. al. [2] present a framework of visual language classes in which the analogy between textual and visual characters, words, and sentences becomes apparent. Visual languages are those languages whose concrete syntax is visual (graphical, geometrical, topological, ...) as opposed to textual. For practical reasons, models are often stripped of irrelevant concrete syntax information during syntax checking. This results in an abstract representation which captures the essence of the model. This is called the abstract syntax. Obviously, a single abstract syntax may be represented using multiple concrete syntaxes. In programming language compilers, abstract syntax of models (due to the nature of programs) is typically represented in *Abstract Syntax Trees* (ASTs). In the context of general modelling, where models are often graph-like, this representation can be generalized to *Abstract Syntax Graphs* (ASGs). Once the syntactic correctness of a model has been established, its meaning must be specified. This meaning must be *unique* and *precise*. Meaning can be expressed by specifying a *semantic mapping function* which maps every model in a language onto an element in a *semantic domain*. Often meaning is given in an operational way by specifying how to execute models in a language. In our approach, we consider an application with a reactive, visual User Interface to be a single element of a language. The application has concrete visual syntax, abstract syntax (the essential structures) and semantics (the behaviour). The Cameleon framework [1] makes a similar distinction focused on the user interface domain between final UI, concrete UI, abstract UI, task and domain. Figure 1 shows concrete and abstract syntax as well as the relation between them explicitly. Semantics is distributed over all of these as reactive behaviour may need to be specified for both the concrete syntax entities and for the abstract syntax entities. The fig-

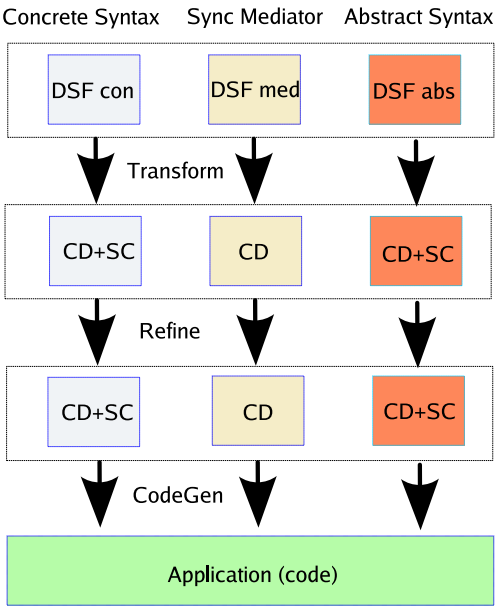

Figure 1: The Process

ure shows three levels. The top level will not be used in our digital watch example. It uses Domain-Specific Formalisms (DSF) to optimally model structure and behaviour of the different parts of the application. These models are subsequently transformed into models in more general-purpose formalisms such as Class Diagrams (CD) and Statecharts (SC). The models thus obtained specify not a single application but rather a *class of* applications. For example, in a class diagram, a multiplicity `*` may be used to indicate the number of allowed buttons in a digital watch. The models may be used to synthesize a visual and interactive analysis and simulation environment. Using such an environment, the modeller may come up with a refinement of the model, thus specifying a smaller language. The above multiplicity may for example be refined to `4` indicating that the final application must have exactly four buttons. This refined model can then be used to synthesize application code.

3. DIGITAL WATCH EXAMPLE

In this section, we use the example of a digital watch application to illustrate the various steps in the process. On the abstract syntax side, the essential parts of structure and behaviour of a digital watch are depicted in Figure 2. Due to space restrictions, the figure does not explicitly show behaviour for all elements of the model. Only for the `Button` class, Figure 3 shows the associated behaviour in the form of a Statechart. The simple statechart has two states: `ButtonOn` and `ButtonOff`. Transitions between these states are triggered by an event or trigger `T:`, but only if the guard `G:` is True. When taken, a transition has an action `A:` as side-effect. Guards and actions can refer to attributes of the class (and at instance level object) whose behaviour is described. Note that as this model is on the Abstract Syntax side, it does not contain nor refer to any concrete (visual) information. Rather, a `press` or `release` event received (from the concrete syntax) is forwarded to the `Watch` object of which the button is part. The `Watch` (known under the role name `watch` here) will subsequently take appropriate action. On the concrete syntax side, a button needs to

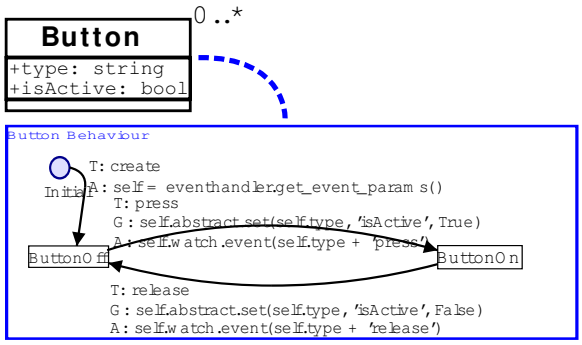

Figure 3: `Button` Structure and Behaviour

be visualized. This is done by means of a `Button2D` visual object which in this case can turn gray or green to indicate the state of the abstract `Button`. The actual changing of colour is done by calling upon the methods `setGray` and `setGreen` of `Button2D`. In our concrete prototype implementation, we use Scalable Vector Graphics (SVG) to render visual objects. As such, `Button2D` specializes `SVGObject`. As such, the `setGray` and `setGreen` are actually implemented by means of SVG instructions. Figure 4 shows structure and behaviour of `Button2D`.

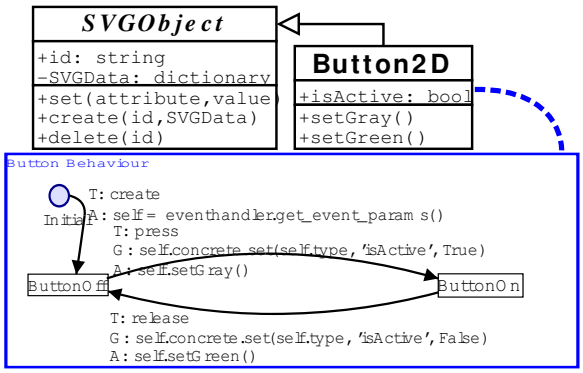

Figure 4: `Button2D` Structure and Behaviour

Figure 5 depicts how abstract and concrete models are linked using a `ButtonA2CS` mediator which keeps both views consistent. Consistency most be guaranteed in both directions as (1) interactively, the concrete syntax may be manipulated which must be propagated to the abstract level and (2) changes may occur at the abstract level (such as time update) which need to be reflected/visualized at the concrete

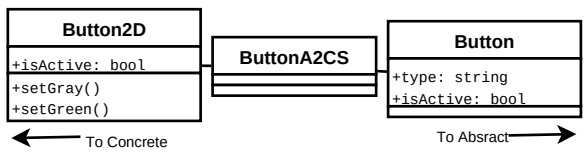

Figure 5: Abstract and Concrete Syntax in Synch

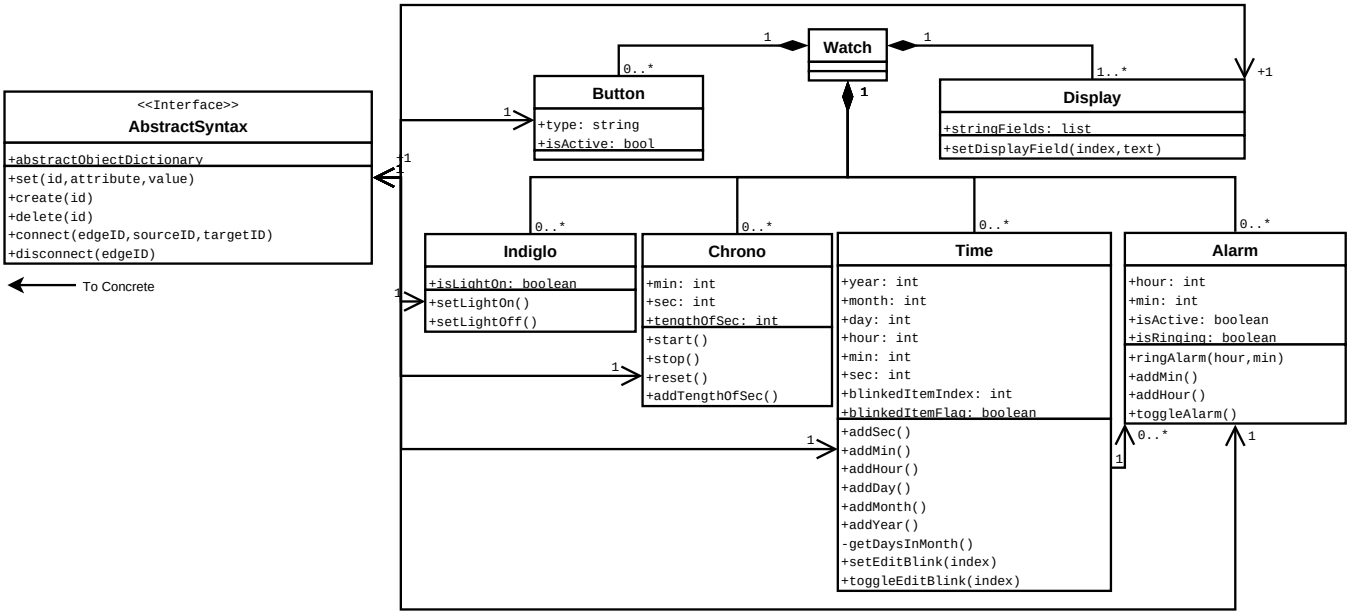

Figure 2: Digital Watches Abstract Syntax

level. Some of the classes in the abstract syntax have no concrete representation. Other classes are related to concrete visual representations (vector graphics drawings). Some associations in the abstract syntax are related to connection splines connecting the visual representations of the classes. Another alternative is to relate abstract syntax associations to geometric or topological relations such as insideness or relative positioning on the concrete side.

Once the above models have been built, an appropriate meta-modelling and model transformation tool such as AToM^3 [3, 4] can be used to synthesize an interactive modelling and simulation environment as shown in Figure 6. Synthesis

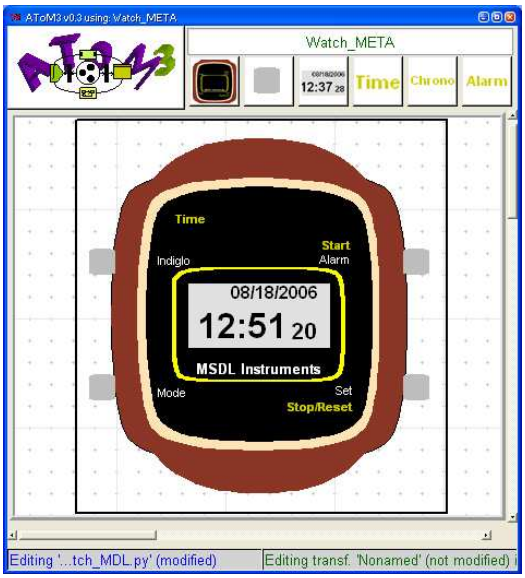

Figure 6: Simulating language elements in AToM^3

is possible thanks to the information available in both the Class Diagrams and (Rhapsody) Statecharts. Note that our current code generator generates Python code which explains the syntax of guards and actions in the Statecharts.

As the models upto now still leave a lot of freedom (in multiplicities at the abstract side and in visual layout on the the concrete side), the modelling and simulation environment allows the modeller to experiment with various model *refinements*. A modeller might for example decide to create a watch which only shows time, and has no chrono nor alarm. Referring back to our discussion about modelling languages, the various alternative models are all sentences in the language defined by the design model. As such, the synthesized modelling and simulation environment is a highly domain-specific visual modelling environment (DSVME). It is interesting to note that the modeller refines the design by manipulating concrete visual (instance) objects.

Eventually, after refinements are complete, an actual application can be synthesized as shown in Figure 7. In our prototype we synthesize a real-time, reactive application whose user-interaction part runs in an SVG-rendering capable browser such as `firefox`. We use an "Asynchronous JavaScript + CSS + DOM + XMLHttpRequest" (AJAX) [6] framework supporting both push and pull interaction as shown in Figure 8 As `XMLHttpRequest` can be initiated from the browser side, we need a means of "pushing" information from the abstract side to the browser. This is particularly necessary for the autonomous, timed digital watch application where time gets updated every second on the abstract side (as specified by the Statechart of the `Time` class in Figure 2). This "push" is achieved by polling the server (every 50ms) in the Javascript `JS_Eval_Poll` inside the browser.

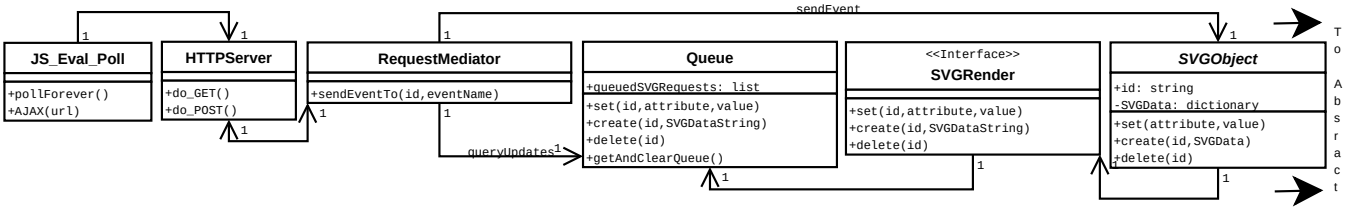

Figure 8: AJAX Framework

Figure 7: Synthesized Application in Browser

4. RELATED WORK

An example of the use of various formalisms for the specification of contex-sensitive interactive applications is given in [12]. Behaviour and structure of the UI are modeled and then XForms and XHTML is generated for the final application. Myers [10] describes the UI challenges and the difficulty of UI and behaviour seperation. This problem gets exacerbated when in addition the application logic is reactive due to complexity of the callback structure. [5] gives a brief high-level overview of a UI synthesizer whereas [8] discusses abstract UI to concrete UI synthesis on multiple platforms. [11] goes deeper into the issues of having different models at the presentation, dialog, and application levels. The closest to our approach is reactive animation [7] which links application behaviour to rendering and reactivity to interactive input. It uses Flash rather than SVG.

5. REFERENCES

[1] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.

[2] G. Costagliola, A. D. Lucia, S. Orefice, and G. Polese. A classification framework to support the design of visual languages. *J. Vis. Lang. Comput.*, 13(6):573–600, 2002.

[3] J. de Lara and H. Vangheluwe. AToM^3: A tool for multi-formalism and meta-modelling. In *FASE '02: Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, pages 174–188, London, UK, 2002. Springer-Verlag.

[4] J. de Lara and H. Vangheluwe. Defining visual notations and their manipulation through meta-modelling and graph transformation. *J. Vis. Lang. Comput.*, 15(3-4):309–330, 2004.

[5] J. Falb, R. Popp, T. Rock, H. Jelinek, E. Arnautovic, and H. Kaindl. Using communicative acts in high-level specifications of user interfaces for their automated synthesis. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 429–430, New York, NY, USA, 2005. ACM Press.

[6] J. J. Garrett. Ajax: A new approach to web applications. http://www.adaptivepath.com/publications/essays/archives/000385.php, 2005.

[7] D. Harel, S. Efroni, and I. R. Cohen. Reactive animation. In *FMCO*, pages 136–153, 2002.

[8] G. Mori, F. Paterno, and C. Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Trans. Softw. Eng.*, 30(8):507–520, 2004.

[9] P. J. Mosterman and H. Vangheluwe. Computer Automated Multi-Paradigm Modeling: An Introduction. *Simulation: Transactions of the Society for Modeling and Simulation International*, 80(9):433–450, September 2004. Special Issue: Grand Challenges for Modeling and Simulation.

[10] B. A. Myers. User interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 2(1):64–103, 1995.

[11] K. Stirewalt and S. Rugaber. Automating ui generation by model composition. In *ASE '98: Proceedings of the 13th IEEE international conference on Automated software engineering*, page 177, Washington, DC, USA, 1998. IEEE Computer Society.

[12] J. Van den Bergh and K. Coninx. Cup 2.0: High-level modeling of context-sensitive interactive applications. In *Proceedings of ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems*, LNCS, Genoa, Italy, October 2006. Springer. Accepted.