Adapting Modeling Environments to Domain Specific Interactions

Vasco Sousa
Université de Montréal
Pavillon André-Aisenstadt
2920 Chemin de la Tour
Montréal, QC H3C 3J7, Canada
dasilvav@iro.umontreal.ca

/EICS '17,/ June 26-29, 2017, Lisbon, Portugal

ACM ISBN 978-1-4503-5083-9/17/06…$15.00
http://dx.doi.org/10.1145/3102113.3102154

Abstract
Software modeling environments are being used by experts in a variety of domains. However, there is no consistent approach to generically synthesize a product line of such modeling environments that also take into account the user interaction and experience adapted to the domain. The focus of my work is the proposal of a solution to explicitly model user interfaces and interaction of modeling environments so that they can be tailored to the habits and preferences of domain experts. We extend current model-driven engineering techniques that synthesize graphical modeling environments to also take interaction models into account. We formalize the semantics of the interaction models with the Statecharts formalism, which also serves as the underlying execution basis. This is done using a novel statechart refinement technique, to provide and promote reuse and semantic consistency.

Author Keywords
Domain Specific Modeling Languages; Human Computer Interaction; Modeling Environments; Statechart Refinement

ACM Classification Keywords
H.5.m [Information interfaces and presentation (e.g., HCI)]: Miscellaneous; D.2.2 [Design Tools and Techniques]: Object-oriented design methods

Introduction

An increasing amount of Model Driven Development techniques focus on providing adaptation to any specific domain, and in particular as a diagrammatic representation. This is done with use of Domain Specific Modeling Languages (DSML). These DSMLs are defined by the concepts captured in the language in the form of an Abstract Syntax (AS) and how these concepts are represented in the form of a Concrete Syntax (CS), specified by the language modeler using common MDE techniques [3].

From these it is possible to generate a modeling environment dedicated to that DSML. But although it provides the adaptation of the CS and AS, the current implementations of such approaches, provide no adaptation of the user interaction. By default this interaction is centered in a point and click behavior, such as [7]. But if we look into domains such as music or home automation for the physically impaired, we can rapidly see the shortcoming of only having easy access to this fixed interaction.

The main goal of my research is to explicitly include the description of user interaction [2] in the specification of graphical MDE tools, so that they can be customized for user needs, habits, and application domain. This inclusion should be integrated in the existing methodologies for creating MDE tools, which rely themselves on MDE, as a way to promote flexibility of development and facilitate their maintenance. In this manner, we focus on providing the following contributions :

1. An explicit modeling language to describe interactions in graphical MDE tools.
2. The automatic synthesis of the graphical MDE tool from the interaction models.
3. A development process to maximize the reuse of existing models of interaction.

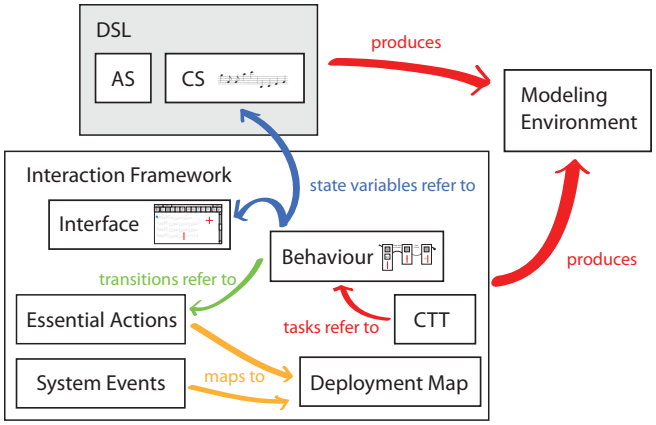


Figure 1: Components of the Interaction specification

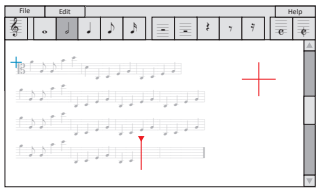
Proposed Solution

Our first step to tackle the specification of user interactions in a DSML environment is to break it into different models, each focusing on a different aspect of user interaction description. This allows us to specify each aspect in its own DSML, with its own concepts and targeted to the appropriate experts. From the knowledge acquired in [4, 6], we propose a set of descriptive models.

Figure 1 outlines the different components of the framework depicting the various models and their relations needed to describe and produce DSML environment interaction.In this framework we have the following models:

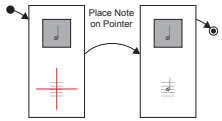
Interface Model

This model is aimed at User Experience (UX) designers to define the static representation of the user interface of the modeling environment for a particular domain. This model specifies all the visual and non-visual artifacts used to convey information to and from a user and how they relate to each other (positioning, overlapping, sound, vibra-

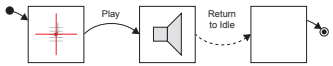


(a) user interface screen model example

Figure 2: Interface Model



(a) Note placement with pointing device example



(b) Note play example

Figure 3: Behavior Model for placing and playing notes

tion. . .). An example of how this is represented for a screen is shown in Figure 2.

For this model we take the concepts from OMG's Diagram Definition [5] and extended it with the following concepts.

Interaction Streams represent forms of interaction with the user that are not representable in the same space, *e.g.,* screen elements, keyboard inputs, and sound.

Layers group diagram elements and provide a semantic segmentation of the interface.

The notion of Anchors, that allow us to have topological constraints in the form of Guards between any two anchors. Through these, we can define positional and scale restriction on elements, minimum and maximum distances between elements, minimum and maximum dimensions, and automatic alignments.

List of Essential Actions
This is a documentation model that consists of a list of all Essential Actions the modeling environment must perform. These are common actions (*e.g.,* copy, paste, save) and actions specific to a modeling language (*e.g.,* instantiate elements in particular ways or to trigger automated operations at key points of the modeling process). This model is populated by a domain expert who define the actions specific to a language and by a UX designer who define the common actions.

Behavior Model
This model expresses the logic of actions expected to take place while interacting with the modeling environment. It defines how the Interface Model reacts to Essential Actions.

We propose to define behavior models in a hybrid formalism composing statecharts with model transformation, with a representation as shown in Figure 3. We use statecharts because it elegantly describes systems that are reactive in nature, such as modeling environments. Model transformation allows us to define these specializations in a rule-based declarative way, in a pre-condition/post-condition fashion, using concrete syntax and Interface Model elements. This makes it easier for UX and domain experts to describe this specialization.

The Behavior Model is aimed at UX designers who tailor the user experience to the domain expert users of the modeling environment.

List of Implementation Specific Events and Operations
This documentation model lists all System Events provided from the target platform (*e.g.,* operating system). For example, for a computer with a touch screen, mouse , and keyboard, events such as OnLeftClick, OnTouch and OnKeyPress can be expected. This list presents platform-specific events that will trigger a particular behavior of the modeling environment when received. The list is populated by the software developer or architect responsible for the deployment on a given implementation.

Mapping of Actions to Implementation Specific Events
To deploy the interaction model on a specific platform, the software developer that provided the Implementation Specific Events and the UX designer shall provide a mapping between the platform-independent Essential Actions (EA) and the platform-specific System Events (SE). This mapping is achieved through the left total function $deploy : EA \rightarrow SE$. This relation can be directly established if EAs are uniquely defined, so that their context information (state of the modeling environment) and user interface elements are accessible through the EA information. Because each action is tied to its own context, the mapping of the same SE (OnLeftClick) can be mapped to different EAs (Select, Place, Play) without conflict.

Concur Task Trees to resolve conflicting flows
One of the results of preliminary analysis of this work is the difficulty of coordinating mutually exclusive interactions. One example of this situation is the disabling of menus when a properties configuration box is open. The difficulty in this situation is to make the interaction that prompts the properties configuration box to open, to also disable other interactions. Taking these conflicting situations into account in every interaction description hinders their understandability and usability, on top of being prone to errors for lack of exhaustiveness. We could introduce control variables that would help this coordination. The problem with this approach is that it introduces elements from a concrete implementation that have no presence in the interface. Therefore, we propose to rely on a model based on CTT to coordinate the interactions without interfering directly in their implementations.

Transformation to Statecharts

The process of taking the previously described models into an executable environment starts by using model transformations to transform the multiple Behavior Models into individual executable statechart models. These statecharts are then ready to be composed through statechart refinement. A final set of refinements introduces the mapping between actions and the events available for a particular implementation, as per Deployment Mapping Model. The process of statechart refinment adheres to a set of operations defined in [8] with the purpose of semantic preservation. Therefore, we can create new interactions on top of existing ones while guaranteeing that new interactions will not interfere with previously defined behaviors. This statechart is then ready to either be used to generate an application through the use of code generation, or used directly in a statechart interpreter.

Conclusion

With this work we open the development process of DSML environments to the inclusion of UX knowledge and in this way further increase the adaptability and reach of DSML approaches, the importance of wich was focal in [1].

From an UX perspective this work allows for a focused description of interaction that is independent from implementation. This in turn, through the specification of DSML environment variations, permits the exploration of new UX.

References

[1] 2016. *19th International Conference on Model Driven Engineering Languages and Systems*. ACM/IEEE.

[2] J. K. Burgoon et al. 2000. Interactivity in human–computer interaction: A study of credibility, understanding, and influence. *Computers in human behavior* 16, 6 (2000), 553–574.

[3] R. France and B. Rumpe. 2007. Model-driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering*. IEEE Computer Society, 37–54.

[4] M. Fraternali and P. Brambilla. 2015. *Interaction Flow Modeling Language*. OMG. Version 1.0.

[5] OMG. 2015. *Diagram Definition, Version 1.1*.

[6] J. M. Rouly et al. 2014. Usability and suitability survey of features in visual ides for non-programmers. In *Workshop on Evaluation and Usability of Programming Languages and Tools*. ACM, 31–42.

[7] D. Steinberg et al. 2008. *EMF: Eclipse Modeling Framework* (2nd ed.). Addison Wesley Professional.

[8] E. Syriani, V. Sousa, and L. Lúcio. 2016. Structural and Behavioral Preserving Statecharts Refinements. *Science of Computer Programing* (2016). (submited).