An Expeditious Approach to Modeling IDE Interaction Design

Vasco Sousa[1] and Eugene Syriani[2]

DIRO, University of Montreal, Canada

Abstract. Software tools are being used by experts in a variety of domains. There are numerous software modeling editor environments (MEs) tailored to a specific domain expertise. However, there is no consistent approach to generically synthesize a product line of such MEs that also take into account the user interaction and experience (UX) adapted to the domain. In this position paper, we propose a solution to explicitly model the UX of MEs so that different aspects of UX design can be specified by non-programming experts. Our proposal advocates the use of multi-paradigm modeling where this aspect of the design of an ME is modeled explicitly and adapted to a specific user expert.

1 Introduction

Software modeling refers to the use of software to model a solution for a problem in a specific domain (e.g., music, finance, biology). As such, there is a plethora of software tools that enable domain experts (e.g., musicians) to represent, manipulate, and simulate models using notations from the domain, specified at a suitable level of abstraction (e.g., a music sheet) [12]. In the programming domain, such tools are often called to Integrated Development Environments (IDEs), like for example Eclipse.

Model-Driven Engineering (MDE) [5] is a generic approach that has proved to be able to generate IDEs in a variety of domains. A main aspect of MDE is characterized by creating a domain-specific modeling language (DSL) that defines the domain, a modeling editor environment (ME) to produce models based on the DSL, and tools that can manipulate these models. The creation of a DSL has been traditionally separated into two aspects: the specification of the abstract syntax (AS) defines the components and structure of the language, and the specification of the concrete syntax (CS) defines symbols and notations associated to AS elements. For instance, if we consider modeling of a music sheet, concepts of note and tempo are part of the AS, and the CS is the way we represent them, e.g., ♩ and 𝄵. The use of a ME provides the means to create and manipulate these domain-specific models, making sure they conform to their language and providing feedback on the modeling process, such as only allowing musical symbols and making sure they are correctly placed on the music sheet. The user experience (UX) [2] is the set of actions, interfaces, and feedback that characterizes the interactions of a user with the software. The creation and manipulation of models, as well as the UX of the ME is the focus of this work.

In most current approaches to ME development, such as the Eclipse Modeling Framework (EMF) [15] and MetaEdit+ [10] the user interaction is restricted to the set of interactions generically built-in the IDE, available as is to all domain-specific environments generated from it. These include, for example, the use of a toolbar with common operations, e.g., copy and paste, and a sidebar with the list of modeling components, in the form of buttons, to select and place them on the main modeling area. Furthermore, in these modeling frameworks, the ME is generated from the AS and the CS. Therefore any further customization requires manual introduction of code into generated code, as pointed out in [4]. However, the user interaction with the ME differs greatly in non-programming domains, which are not the main focus of IDEs like Eclipse. For example, the alignment of the notes is crucial in an ME for music sheets. Thus, the modeler needs to understand the internals of the framework instead of intuitively defining how musicians expect to interact with a music sheet. This is one of the main practical reasons why domain experts continue to use IDEs dedicated to their domains, instead of IDEs generated from generic MDE frameworks.

With the popularity of new computational devices and peripherals at the expense of traditional desktop environments, such as tablets with varying sizes, virtual and augmented reality goggles, and collaborative interactive tables, new human user interactions will keep on appearing. It becomes clear that the interaction with ME, needs to be adapted in new ways. For instance, works such as [1,13] strive to expand user interactions beyond that of traditional means, in ways to tackle UX issues, and grasp other requirements and ways of expressing user interactions.

In this position paper, we propose the idea to promote the UX into an explicit component of ME creation, at the same level as the AS and CS of a DSL. This would allow us to tap into knowledge from the UX domain [2] and be able to define families of interactions that provide bases of adaptation of the ME to different devices and user profiles. To achieve this, we need to express these interactions in a platform-independent manner, with terms that are adapted to UX experts and domain experts in addition to software developers, and to transform these descriptions of interaction into platform-specific interactions for the deployment of the ME on diverse mediums. Our proposal advocates the use of multi-paradigm modeling where is aspect of the design of an ME is modeled explicitly and adapted to a specific user expert.

In Section 2, we briefly discuss about work done on the development of user interfaces. In Section 3, we extend the MDE approach for generating MEs by tackling the specification of the UX. We illustrate our approach with a simple music modeling language example and its editor. Finally, we discuss our roadmap in Section 4.

2 Related Work

The closest work to addressing the interaction issue in some standard or formal way is the recent standardization of the Interaction Flow Management Language (IFML) [6] by the Object Management Group (OMG). IFML was born from works on modeling web applications[3]. Although UX was not intended to be within the scope of IFML and its precursors, we still consider it as related work since it covers some segments of the interaction portion of UX. Its goal is to allow software developers to specify user interactions, by describing its components, state objects, references to the underlying business logic and its data, and the logic that controls the distribution and triggering of the interactions. Despite this distancing from full UX, the accumulated knowledge present in IFML is still valuable to our research.

There are nevertheless several aspects of IFML that do not concord with our approach. IFML describes user interfaces (UI) as constituents without any specification of visual properties or design choices. This limits the description since these properties have a large influence on the UX and should be adapted accordingly. IFML is targeted to Software Developers instead of UX designers that could use their domain knowledge to better adapt the software to its users. It promotes the combined use with other OMG standards, such as Class Diagrams and Business Process Models, but through declaration of specific function calls that effectively bind the user interaction to a specific implementation, obfuscating most of the abstraction effort. It also relegates complex interactions to be specified by the implementation framework instead of being an explicit part of the interaction model. Finally, our evaluation of IFML also concluded that it has some limitations in terms of scalability, where the complexity of the specification increases greatly with each UI element that is introduced, and the use of modules is focused on implementation reuse rather than development simplification.

In our previous work [12], we evaluated the user interaction in 25 MEs from different domains. We provided metrics for measuring the usability and suitability of domain-specific MEs. We also investigated what ME features are needed in which domain and how they should be presented to the user domain. The present work relies on the results of [12] which serves as requirements for our current work.

3 Proposed Solution

Our first step to tackle the specification of user interactions for a DSL 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 domain-specific model, with its own concepts and targeted to the appropriate experts. From the knowledge acquired in [6,12], we propose a description of the UI, a description of the flow of interactions that defines the behavior of the interactions, a list of interaction requirements of the ME for a specific domain, a list of what a specific system provides in terms of interaction events, a mapping that binds all of these models together, and a mapping that provides the link between the interaction descriptions and the information needed for the platform-specific implementation.

3.1 Music Modeling Example

To illustrate our proposal we use a case of a music modeling language akin to MusicXML [7], and the subsequent specification of the interaction and artifacts of this ME. We choose the domain of music writing because its interactions are diverse and complex due to the large work outside computational environments, as well as because it removes the bias of staying in a domain close to computing, which is the predominant domain of application in the MDE literature. This is inspired from the MuseScore ME [14].

The language provides a form for writing music sheets. In this paper, we focus only on a small number of interactions: various ways of placing notes on the sheet, playing a note, and selecting a note length (half note, quarter note ...). The goal here is to demonstrate how we address multiple interaction requirements of a music modeling language, so that the final ME is properly adapted to the music domain expert and the environment he uses.

3.2 Modeling Views

In our approach, domain-specific MEs are not only generated form the AS and CS, specified by the language modeler using common MDE techniques [15], but also from the UX model. This model is composed of the following views.

Interface Model The Interaction Model is aimed at UX designers to define the static representation of the user interface of the ME 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, precedence, sound, vibration. . .). A good candidate to define the

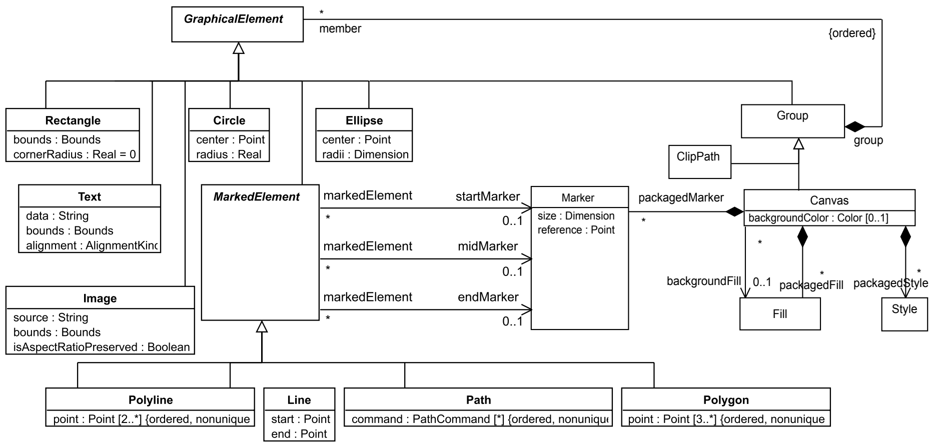


Fig. 1: Diagram Graphics Metamodel Primitive and Group Elements from [11]

Interface Model is the OMG's Diagram Definition (DD) presented in [11]. Fig. 1

depicts an excerpt of the Diagram Graphics metamodel that specifies the definition graphical elements and the segment of the metamodel that allows the specification of element groups and other organizational constructs, such as the canvas. However, these are not enough to define the topological placements of UI elements, in a variety of dimensional spaces from 0D (Numerical Display, Single speaker) to 2D (Screen, binaural Audio, Touch Tables), that we require for UI specification. Thus our proposal is to extend the DD metamodel with constructs that would allow for a better specification of UI, as shown in Fig. 2.

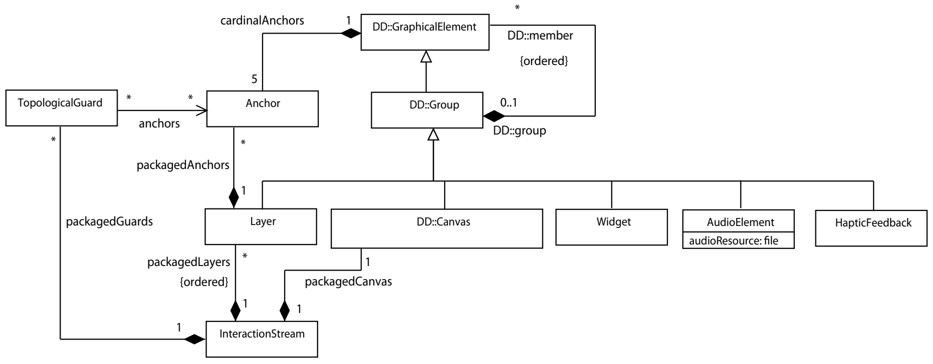


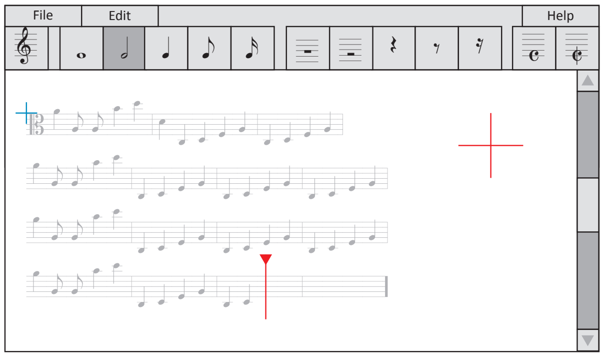
Fig. 2: Extension to the Diagram Graphics Metamodel

We start by organizing these spaces as Interaction Streams (IS). These represent forms of interaction with the user that are not representable in the same space, e.g., screen elements, keyboard inputs, and sound. Fig. 3a shows an Interaction Stream, namely the representation of screen elements for our music ME. Fig. 3b shows a note input stream and a sound stream, both complementing the first stream, but with no direct visual representation as a screen element. These abstract inputs can then be mapped onto more concrete forms of input, such as an electronic piano keyboard, but at this level, we only allow for the inclusion of platform-independent input.

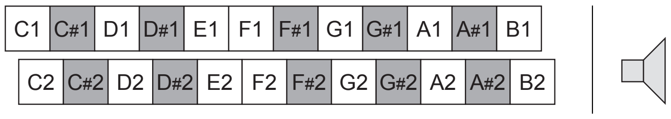
These IS are in turn organized into Layers. Layers group representational characteristics of the UI elements such as shade, size, position, data, audio characteristics, representations of haptic feedback, and other perceptible characteristics, and how to group them, in the form of DD Graphic Elements (GE).

These GE are then specialized into UI specific elements such as buttons, check boxes, drop down menus, and other UI widgets, to simplify the modeling process. Fig. 3a shows the definition of such GE: we can have the typical menu and button widgets positioned at the top, with a representation of a selected button for illustration purposes, and the remaining space as the canvas.

Additionally all of these GE are extended with a set of cardinal anchors and a central anchor. Alongside these automatic anchors, manual unmovable anchors and guides can be placed in the model. These allow us to have topological constraints in the form of Guards between any such anchors, including different anchors of the same GE. Through these topological constraints, we can define



(a) UI screen model example



(b) UI abstract input and sound output model examples

Fig. 3: Interface Models

positional and scale restriction on elements, minimum and maximum distances between elements, minimum and maximum dimensions, and automatic alignments. This is achieved by adjusting the movable anchors pointed by the Guard so that its constraint expression is satisfied. The metamodel presented is further constrained by well-formlessness rules not represented here, such as: the number of Position elements of an Anchor matches the number of dimensions of that particular Interaction Stream.

For representation purposes, we also show the positioning of CS elements from the Music Modeling DSL in the canvas. These CS elements align at the top left anchor in the canvas and also conform to the Music Modeling DSL CS and AS specifications. The two remaining elements on the canvas are the representations of a cursor as a line with a marker over the CS, and a pointing device as a large cross.

List of Essential Actions This is a documentation model, contains a listing of all Essential Actions the ME 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, perform checks at key points of the modeling process). For our example, we only consider copy and paste actions, as well as actions specific to music modeling: placing a note, playing a note, and selecting a note type. This model is populated by a domain expert to define the actions specific to a language and by a UX designer to define the common actions.

List of System Events 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 ME when received. The list is populated by the software developer or architect responsible for the deployment on a given implementation.

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

We propose to define behavior models in a hybrid formalism composing Statecharts (SC) [8] with model transformation [16]. We use SC because it elegantly describes systems that are reactive in nature, such as MEs. Furthermore, SC refinement [8] gives the modeler the possibility to specify a generic behavior of the ME model in the form of a SC with gaps to be specialized and filled-in per DSL and user profile. Model transformation allows us to define these specializations in a rule-based declarative way, using CS and Interface Model elements. This makes it easier for UX and domain experts to describe this specialization.

Behavior Model Formalism We briefly describe the hybrid formalism of SC and model transformation in an informal way. In our approach, we assume there is a SC that models generically any ME and defines generic operations such as opening and closing the ME and other basic operations. Additionally, it contains abstract states that act as placeholders to specify behaviors specific to the domain of the ME. Specializations of this SC is done by embedding well-formed SCs following the rules and conditions in [8]. At this point we opt for a restrictive approach to these specializations to guarantee the soundness of the UX.

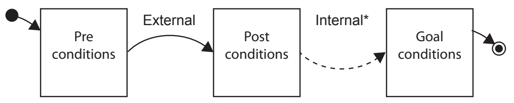
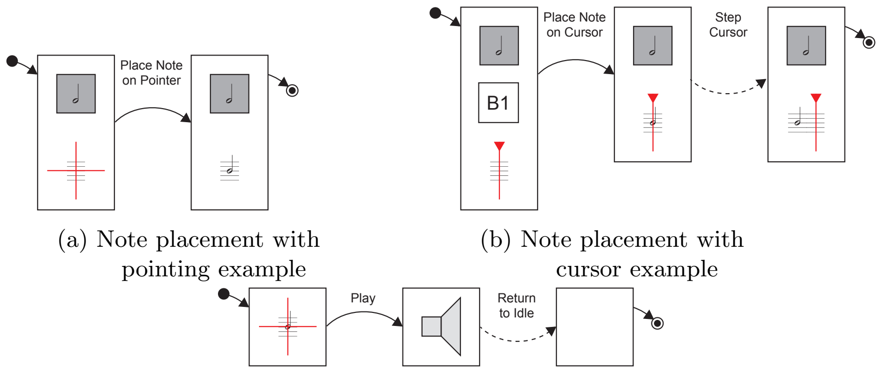


Fig. 4: Abstract representation of specialization model

Fig. 4 illustrates the structure of a specialization model: a SC to be embedded in another generic one for specialization purposes. There are three kinds of states: pre-condition, post-condition, and goal condition states. States contain CS and Interface Model elements that specify the condition for a transition to be triggered. Transitions can be external or internal. External transitions are triggered by Essential Actions performed by the user and/or perform actions that are perceived by the user. These transitions are to be mapped to platform-specific System Events. Internal transitions are triggered by actions internal to the scope of the system and do not result from a human interaction, such as the passage of time and transitions that are immediately triggered as soon as the source state is reached. The semantics of the application of a transition t is as follows: if the pre-condition of t is satisfied and the corresponding event is received, then the system should update itself to satisfy the post- or goal condition. A pre-condition state represents the left-hand side pattern of a model transformation rule that shall be matched over the current state of the ME. A post-condition state represents the right-hand side pattern of a model transformation rule that must be satisfied after the event is received. Post-conditions also serve as pre-conditions of following transitions. Goal conditions special kinds of post-conditions that must be satisfied only when an internal event is received. Specialization models must always start with an external transition. Conditional, hierarchical, and parallelization constructs of SC, such as branching, forking/joining, and OR- and AND-states are all supported [9].



(a) Note placement with pointing example

(b) Note placement with cursor example

(c) Note play example

Fig. 5: Behavior Model for placing and playing notes

This approach allows us to easily define concurrent interactions, through the use of AND-states in the refinement of the generic behavior SC. In Fig. 5a and 5b we have two different interactions sequences to place a note on the music sheet. In Fig. 5a we have the note placement with a pointing device, where a UI specific button element that will relate to note length is selected and the corresponding CS element is placed on the pointed place of the music sheet as an external action. In Fig. 5b we again have the selected length note placement, but instead it occurs on a note activation (note B1 is activated) and the corresponding CS element is placed by the external action wherever the cursor is currently placed, followed by an internal action that advances the cursor in a linear fashion. The note placement of Fig. 5a can then be used with any pointing device, such as a mouse or a touch screen. The note placement of Fig. 5b can be used with any device that allows the direct referencing of a note, such as a musical keyboard.

Fig. 5c shows the interaction of using a pointing device when a note is already placed at that point of the music sheet. Instead of an interaction with the CS by placing a new note as an external action, we play the sound of that note. This means that the play interaction will be placed on all states that stem from the placement interaction. Once the tone for that note has finished playing an internal action triggered by the system advances to an idle state.

The Behavior Model is aimed at UX designers that tailor the user experience to the domain expert users of the ME.

3.3 Deployment Mapping Model

In addition to all the model views of the interaction, we have 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 ME) and UI elements are accessible through the EA information.

SE \ EA	Select	Place Note on Pointer	Place Note on Cursor	Play
OnKeyDown			×	
OnTouch	×	×		×
OnLeftClick	×	×		×

Table 1: Deployment Mapping Model

The mapping is specified in the form of a table, as shown in Table 1. It is produced by the UX designer to establish the relation between platform-independent external actions that interact with the user defined in the list of essential actions and real world actions translatable by the platform-specific system events. For example, in Table 1, we map multiple SEs (OnTouch and OnLeftClick) onto the same EA (Select) to provide alternative interactions and to adapt it to a particular device. Because each action is tied to its own context, the mapping of the same SE (OnLeftClick) can be mapped to different EAs (Select, Place Note on Pointer, Place Note on Cursor) without conflict.

4 Conclusion

In this position paper, we motivated the need to address the issue that generated MEs currently suffer from: the lack of adaptation of its behavior to improve the UX of its domain expert users. Following a multi-paradigm modeling approach, we proposed to explicitly model the UX of a DSL in order to generate a ME where the domain user will feel completely immersed with interactions and utilization of the tool adapted to what he is accustomed to. The AS and CS of the DSL specify the syntax of valid models. The Interface Model represents the representation of the user interface. The CS of the DSL, Interface Model, and Essential Actions are used to define the Behavior Model of the ME. System Events are mapped to Essential Actions. All of these models are necessary to the synthesize MEs where the UX is adapted to domain on a platform using specialized I/O peripherals.

Our future work is to complete the full implementation of a prototype so we can start validating and improving our solution with real-world experts from different domains. We also plan to investigate different interaction streams that go beyond visual screen rendering, such as audio, video animation, and tactile haptic sensing supported by appropriate hardware devices. Furthermore, we plan to formalize the different models and formalisms outlined here as to provide precise feedback to the users for inconsistencies, well-formlessness and other design flaws.

References

1. F. Alonso, J. L. Fuertes, Á. L. González, and L. Martínez. User-Interface Modelling for Blind Users. In *Computers Helping People with Special Needs*, volume 5105 of *LNCS*, pages 789–796. Springer, 2008.
2. B. Buxton. *Sketching User Experiences.* Interactive Technologies. Morgan Kaufmann, Burlington, 2007.
3. S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (webml): A modeling language for designing web sites. *Comput. Netw.*, 33(1-6):137–157, June 2000.
4. A. El Kouhen, A. Gherbi, C. Dumoulin, P. Boulet, and S. Gérard. MID: A MetaCASE Tool for a Better Reuse of Visual Notations. In *System Analysis and Modeling: Models and Reusability*, volume 8769 of *LNCS*, pages 16–31. Springer, 2014.
5. R. France and B. Rumpe. Model-driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering*, pages 37–54, Minneapolis, may 2007. IEEE Computer Society.
6. M. B. Fraternali. *Interaction Flow Modeling Language.* OMG, February 2015. Version 1.0.
7. M. Good. MusicXML for Notation and Analysis. In *The Virtual Score: Representation, Retrieval, Restoration*, volume 12 of *Computing in Musicology*, pages 113–124. MIT Press, Cambridge MA, 2001.
8. C. Hansen, E. Syriani, and L. Lucio. Towards Controlling Refinements of Statecharts. In *Software Language Engineering Posters*, volume CoRR: abs/1503.07266 of *SLE '13*, 2015.
9. D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *Transactions on Software Engineering and Methodology*, 5(4):293–333, oct 1996.
10. S. Kelly, K. Lyytinen, and M. Rossi. MetaEdit+ A fully configurable multi-user and multi-tool CASE and CAME environment. In J. Iivari, K. Lyytinen, and M. Rossi, editors, *Conference on Advanced Information Systems Engineering*, volume 1080 of *LNCS*, pages 1–21, Crete, may 1996. Springer-Verlag.
11. Object Management Group. *Diagram Definition, Version 1.1*, 2015.
12. J. M. Rouley, J. Orbeck, and E. Syriani. Usability and Suitability Survey of Features in Visual IDEs for Non-Programmers. In *Evaluation and Usability of Programming Languages and Tools*, PLATEAU'14, pages 31–42. ACM, oct 2014.
13. A. Savidis and C. Stephanidis. Developing Dual User Interfaces for Integrating Blind and Sighted Users : the HOMER UIMS. In *CHI'95 Proceedings.* ACM, 1995.
14. M. Shinn. *Instant MuseScore.* Packt Publishing Ltd, 2013.
15. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework.* Addison Wesley Professional, 2nd edition, 2008.
16. E. Syriani, Hans Vangheluwe, and B. LaShomb. T-Core: A Framework for Custom-built Transformation Languages. *Journal on Software and Systems Modeling*, 14(3):1215–1243, 2015.