Chapter #

A HYBRID TOOL FOR USER INTERFACE MODELLING AND PROTOTYPING

Hallvard Trætteberg
Dept. of Computer and Information Sciences (IDI), Norwegian University of Science and Technology (NTNU). Sem Sælands v. 7-9. 7491 Trondheim, Norway. hal@idi.ntnu.no

Abstract: Although many methods have been proposed, model-based development methods have only to some extent been adopted for UI design. In particular, they are not easy to combine with user-centered design methods. In this paper, we present a hybrid UI modelling and GUI prototyping tool, which is designed to fit better with IS development and UI design traditions. The tool includes a diagram editor for domain and UI models and an execution engine that integrates UI behaviour, live UI components and sample data. Thus, both model-based user interface design and prototyping-based iterative design are supported.

Key words: model-based user interface design; user interface prototyping.

1. INTRODUCTION

With the standardization of UML and specialised UML profiles, modelling of systems and software artefacts seems to be accepted in the engineering community. Model-driven architecture (MDA) and engineering (MDE) and service oriented architecture (SOA) all rely on models for representing the structure and functionality of software. For many, if not most, information systems (IS), models are crucial to managing the complexity of both the outside world and the computerized information system (CIS).

The field of model-based user interface design (MBUID) has a long tradition [1][2][3], but it has been noted that UI modelling is not very

common [4]. One reason is that many actors and activities that are part of user-centered, iterative design are not well supported by models, so alternative representations and means of communicating ideas must be used. E.g., most tools take a top-down approach to UI design by requiring task and dialog models before moving to concrete design, while iterative design methods often require support for bottom-up design.

In this paper we present our work on a hybrid UI modelling and GUI prototyping tool. First we give a brief overview of the "standard" model-based UI design approach and compare this to prototyping-oriented iterative design. The notion of complementing representations is used to motivate our hybrid modelling and prototyping tool, which is then presented. We finally compare our approach and the tool with others' work.

2. MODEL-BASED UID MEETS UI PROTOTYPING

Research in model-based UI design has resulted in many specific modelling languages methods. Although there is no UML of UI design, there seems to be a general agreement of three classes of models [1][3][6]. *Task* and *domain* models focus on the user's and designer's knowledge of the user, their goals and the tasks they need to perform. *Dialog* models focus on input and output of information, Abstract Interaction Object (AIO) structure and dynamic behaviour. Dialog models abstract away details of interaction style and client platform. Models of *concrete design* describe the structure and details of Concrete Interaction Objects (CIOs) and is specific to interaction style and client platform. In the model-based approach, design progresses from task models, through dialog models to concrete interaction design, in a top-down process. Parts may be generated, based on logical dependencies, using formal systems like predicate logic, process algebra, petri nets, state machines, etc [2].

There is no *conflict* between model-based UI design and user-centered design, since modern model-based design has a strong focus on task, domain and user models. E.g. Mobile [5] claims to supports user-centered design by including all three classes of models. There is however, a *tension* between these approaches since their respective processes and techniques are different. User-centered design is by nature iterative, meaning that design is moved forward by cycling through UI (re)design, prototyping and evaluation. Testing by end-users is considered important and will in most case rule out using (only) abstract and formal prototypes.

Prototypes are one of many design representations that are used in a user-centered process, others are sketches, scenarios, storyboards and mock-ups. Common for all these are that they are concrete and informal, as creative

processes are best supported by flexible representations, and formal representations are found to be constraining, as are their tools [12].

There are many important characteristics of models in general, that makes it desirable to combine model-based and user-centered approaches. A hybrid approach and tool should integrate complementing representations, rather then force designers to select one, to be less constraining and more useful. The tool should support moving between abstract and concrete representations, and change the level of completeness and formality according to the needs of the process.

The following sections will describe two complementary representations, CAP and Diamodl, and a hybrid model-based prototyping tool, designed to improve support for the movements between informal and formal representations of abstract dialog and concrete interaction.

3. THE CAP AND DIAMODL NOTATIONS

In this section we present the two notations CAP and Diamodl, that our tool supports.

3.1 Canonical Abstract Prototypes

Prototyping is a crucial part of iterative design and prototypes are used throughout the design process both alone, as a medium for evolving and communicating design ideas, and with end-user testing to resolve design issues. In [7], Constantine notes that prototypes can be placed along an “abstraction continuum” (section 2, second paragraph). To support “smooth progression from abstraction toward realization” he introduces his Canonical Abstract Prototypes (CAP). Simply stated, a CAP is a low-fidelity prototype consisting of nested rectangular regions representing user interface components, where the components’ interactive function is indicated by an icon and a name. The icons are drawn from a standardized set, hence the term “canonical”. Material components roughly correspond to output of information, while tool components correspond to input/editing and actions.

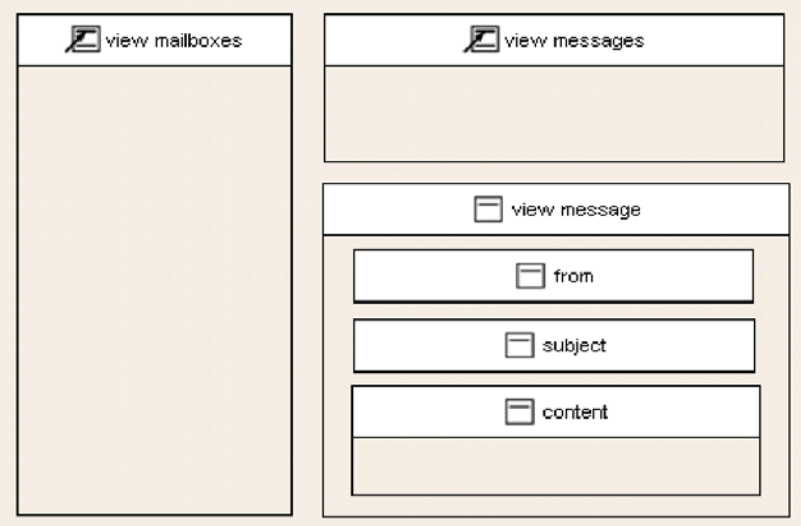

Figure #-1. Canonical Abstract Prototype example

Figure 1 shows a CAP example, with two components for viewing mailboxes and messages, labeled “view mailboxes” and “view messages”, respectively. The interactive function is indicated by the icon and includes selection, i.e. input of an element, in addition to output of a set of elements. A third component labelled “view message”, contains three sub-components labelled “from”, “subject”, and “content” and is used for viewing individual messages,. As can be seen, the nesting and layout makes the diagram look fairly similar to a quickly assembled widget hierarchy in an interface builder.

3.2 The Diamodl notation

The Diamodl notation [8] is a diagram language for modelling abstract dialog, both information handling and activation logic, which is particularly suited for business applications. The *interactor* and *gate* concepts specify the input and output function of an AIO. The *variable*, *computation* and *connection* concepts are used for specifying how data is stored, transformed transported among interactors. Figure 2 shows a simple model where these five concepts are used. The upper left box is a variable and is used to store data. This particular one is limited to holding a String and currently contains “Hallvard”. The two boxes with title bars to the right are interactors and the attached triangles are gates. As indicated by the gates, the upper interactor may output and input a String, while the lower one may output an Integer. The variable is connected to the output gate of the former interactor, while the input gate is connected to a computation, the output of which is further

connected to the output gate of the latter interactor. The name and the input/output types suggest that the computation computes the length of the input String. Hence, this particular model specifies a UI through which the user may edit a String and see its length.

In addition to these five concepts, states (an interactor is also a state) and transitions are used for specifying the activation and deactivation of interactors, while UML class and object diagrams are used for data modelling.

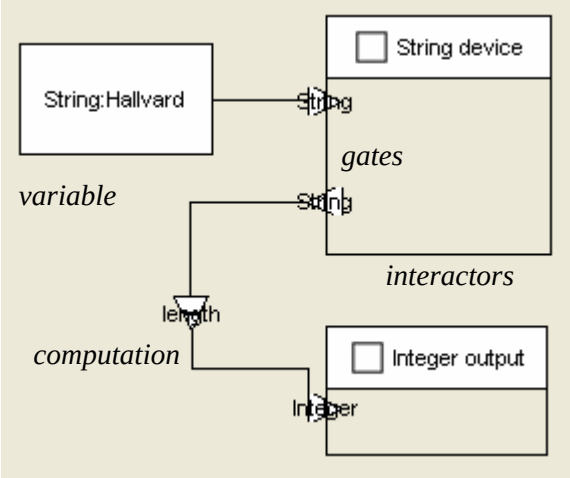

Figure #-2. The variable, interactor, gate, computation and connection concepts

In Figure 3, a Diamodl example corresponding to the CAP example in Figure 1 is shown. As above, interactors are shown as rectangles with gate triangles on their left side, either pointing into for user output or out for user input. Variables are labelled with class names and multiplicity. Computations are shown as free-floating triangles, and are labelled by attribute, association or operation names from the UML model (not shown). Connections are shown as edges and may optionally be labelled with attribute or association names, to indicate an embedded computation. This particular model is a formalisation/explication of the corresponding CAP diagram, where the data types and input/output functions of the interactors are made explicit. The “view mailboxes” interactor outputs a set of mailboxes and lets the user select (input) one of them. From the selected mailbox a set of messages is computed and used as output by the “view messages” interactor. One of these messages may be selected and output by the “view message” interactor. As can be seen, there is a correspondence between input and output gates on interactors in Figure 3 and the CAP

components and icons in Figure 1. However, since CAP has no formal semantics, this is by no means a mechanical process.

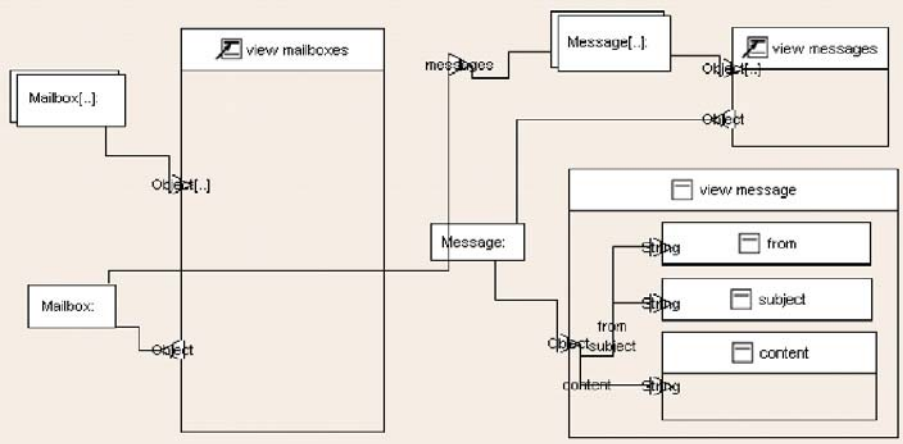

Figure #-3. Diamodl example

The innermost interactors in a diagram will correspond to CIOs like text fields, buttons, sliders, lists etc, determined by the interactor's input and output gates. A single selection list widget will for instance be capable of implementing the "view mailboxes" interactor in Figure 3.

4. THE HYBRID MODEL-BASED UI PROTOTYPING TOOL

Our hybrid diagram editor supports a mix of a small subset of UML's class and object diagrams, UI modelling with CAP and Diamodl and GUI design with "live" widgets. A document may be split into diagrams, and each diagram may include an unconstrained mixture of model elements. In this section we will first describe and discuss one way of approaching the design process, while indicating relevant variants. Later, we will briefly describe the Eclipse-based architecture and implementation.

4.1 The CAP diagram

The CAP notation has its strength in early design, because it simple and informal. There is essentially only one kind of object, the CAP component, depicted as a rectangle with an icon and a name. The icon indicates the interactive function of the component, and since there are no hard constraints

on how components are composed, the icon may be changed at will. The creation tool palette (left in Figure 4) includes the most common kinds and the icon may later be changed to a more specialised one, by using the contextual menu. The component name typically indicates the kind of information a component handles, like “send message” or “view messages”. If the name is recognized as a <verb> <noun> phrase with a CAP-specific vocabulary, the interactive function/icon will change automatically. E.g. changing the name of a component to “edit element”, will change the icon to correspondingly.

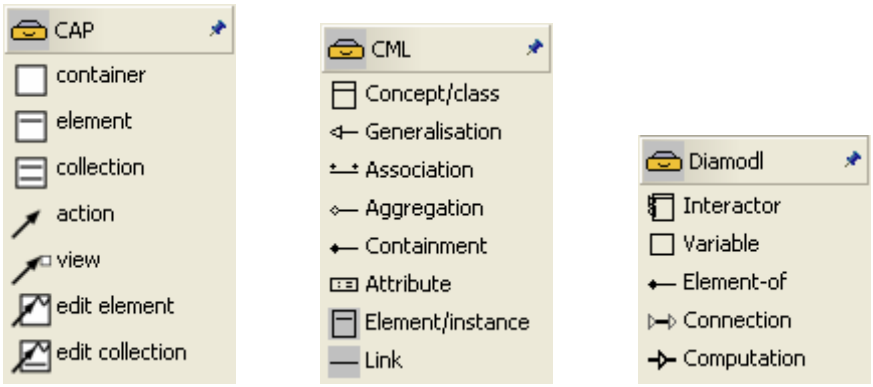

Figure #-4. The CAP, domain and Diamodl modelling tools

Although informal, CAP diagrams can provide important information about tasks, domain objects and actions. At some point, however, it will become important to be more explicit about what is left implicit in CAP.

4.2 The domain model

A natural second step is to define the meaning of the terms used on the CAP diagram(s) in e.g. UML. In the editor we support the following object-oriented concepts (middle in Figure 4): class, generalisation, attribute, association (with roles) and aggregation/containment, element/instance and link.

Non-verb terms in a CAP diagram will typically show up in the domain model as names of classes, attributes, associations and roles. E.g., the terms “mailbox” and “message” give rise to two classes, the term “messages” refers to both a class and an association between the two classes, and terms “from”, “subject” and “content” are attributes of “message”. It is possible to define the classes, attributes and associations side-by-side the CAP components, and/or in a separate diagram.

One way of validating the domain model is to create sample data based on a user story or scenario, to ensure that the data are meaningful to the end-user. In the editor, instances of both classes and associations (objects and links in UML terminology) may be created as diagram elements, while attribute values are edited in a property sheet. Later in the process, the sample data will be used to populate a "live" user interface, so it's important to provide sensible data to avoid taking the focus away from the user interface structure and logic. To avoid constraining the designer, we support making the sample data first, and then create the class diagram. For instance, if ethnographic methods are used to collect field data from a workplace, these may be "transcribed" into objects and links in a diagram. The classes and association may be introduced when our understanding of the domain is more complete. There is currently no tool for doing this automatically, i.e. deriving a complete class diagram from a set of instances and links, but we hope to provide some support easing this process in the future.

4.3 The Diamodl dialog model

Although the CAP notation is fairly intuitive, it has no defined formal semantics, so most aspects of the dialog are left implicit. E.g., a CAP component named "view mailboxes", suggests that a set of mailbox objects are presented to the user. However, where the set of mailboxes come from is not expressed. E.g., is it the set of all mailboxes or is it a particular set like those containing unread messages? Similarly, if a CAP component is classified as view with selection, there is no way of expressing how the selection is used by other parts of the user interface. To explicitly represent these aspects of the user interface design, Diamodl must be used.

Since both CAP components and Diamodl interactors represent user interface containers or components, the CAP component hierarchy usually corresponds to similar Diamodl interactor hierarchy. After creating the CAP diagram, it may be *evolved* into a Diamodl diagram, by adding gates, variables, connections, states and transitions. For instance, the "view mailboxes" component in the CAP diagram in Figure 1, may be formalised as an interactor that outputs a set of mailboxes and lets the user input (select) one of them. This means that the interactor must have both output and input gates, and that their multiplicities are "unbounded" and 1, respectively, as shown in the Diamodl diagram in Figure 3. As mentioned, the CAP components are drawn from a standardised set, and a similar set of standard interactors has been added to the creation tools palette. Hence, gates may either be added manually, or the CAP components be replaced by their interactor counterpart.

Variables, computations and connections may also be added, by means of the Diamodl specific tools palette (right in Figure 4). In the example model in Figure 3, we have added variables to hold both the output and input data for the “view mailboxes” interactor. The input data, i.e. selected mailbox, is connected through a computation to another variable, holding sets of messages. The computation is labelled “mailboxes” and refers to the association from the domain model. In this case the association is used as a function, and we can see that the variable types are correct, as the multiplicity of the association is 1-n, i.e. the “messages” function maps from a single mailbox instance to a set of messages. Both attribute and association names may be used as labels/functions on computations and connections.

Transitions between interactors and/or states are typically introduced last, to express when the various interactors are active and what triggers their activation or deactivation. The need for transitions and states depends on characteristics of both the style of interaction and client platform. E.g. speech interfaces are usually more sequential than forms based business applications, and interfaces for PDAs with small screens have more navigation than their desktop counterpart.

4.4 Combining interactors and concrete widgets in a testable GUI prototype

The dialog model describes the abstract logic, but says nothing about what the GUI will look or feel like. To be able to communicate with end-users it is important to give it a concrete visual representation, and to further evaluate the design by means of end-user testing, it is also necessary to make a testable prototype with real data and functionality. To support this, the editor lets the designer add “live” user interface components to the diagram, link the interface components to the inside of the corresponding interactors and execute the dialog model.

As mentioned in Section 3.2, the innermost (leaf) interactors of a dialog model correspond to or may be implemented by specific CIOs. E.g. the “view mailboxes” interactor in Figure 3 may be implemented by a single selection listbox. The editor includes a palette of tools for inserting standard widgets into interactors and means for defining how the logical gate model for output and input is mapped to the event and property model of the widgets. To make the widgets easier to combine with the interactors, the editor includes a palette of predefined interactors with embedded widgets, that may be added as replacements for the ones already present in the diagram.

The interactors model in Figure 2 is shown left in Figure 5, augmented by concrete widgets. A text field has been inserted in the “String device”

interactor while a spinner has been chosen for the “Integer output” interactor, and both are connected to the gates of their respective interactors. Values are continuously propagated and as can be seen, the spinner has been updated to reflect the length of the current String value, as computed by the computation element.

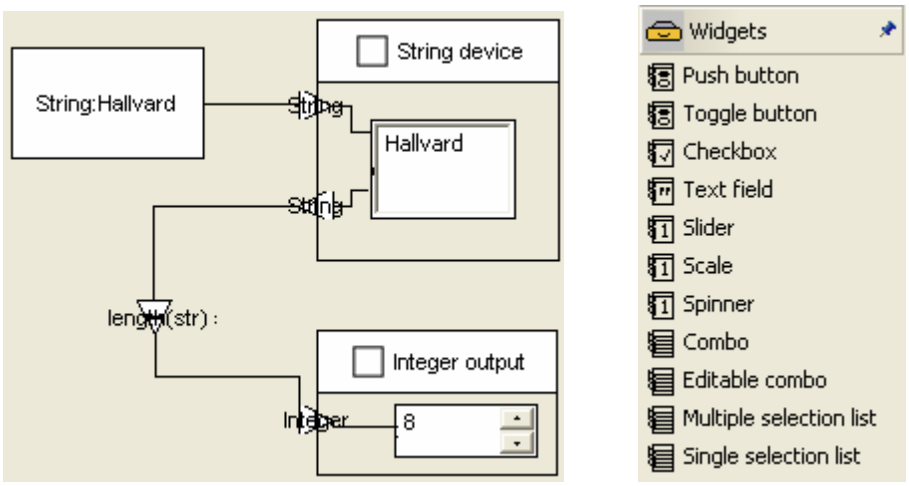

Figure #-5. Concrete widgets from a palette (right) may be inserted into the interactors

In general, propagation of values through the network of connections is triggered whenever a value changes, either when the user interacts with a widgets or a variable changes, e.g. when the objects, links and attribute values specified as sample data are edited. If the widgets are correctly attached to the interactor gates, the UI will be functional and testable. In Figure 6, a prototype for the larger example model from Figure 3 is shown. The top left variable contains the two mailboxes defined as sample data and these are propagated into the listbox inside the “view mailboxes” interactor. The user has selected the “Inbox” instance and its contained messages, as computed by the “messages” function, are shown in a listbox inside the “view messages” interactor. Finally, the user has selected the first of these messages and its “from”, “subject” and “content” attribute values are shown in respective text fields.

4.5 Top down vs. bottom up modelling

The process that has been outlined above starts with an informal CAP diagram, evolves it into a more formal domain and dialog model and adds live widgets to give a functional prototype that the end-user may interact

with. Although the prototype does not have the look and feel of a high-fidelity prototype, it may still be used to validate the abstract logic of the UI.

A more bottom-up approach is also supported. Since the "raw" widgets are available in a dedicated widgets palette, as shown right in Figure 5, they may initially be inserted without their interactor counterpart. With a decent layout, it can be made into a non-functioning horizontal GUI prototype, that may be validated as such. The widgets may even be operated, but to make such a prototype functional, the abstract logic must be added, by wrapping the widgets in the corresponding interactors, connecting the widgets to the gates, and adding other Diamodl language elements.

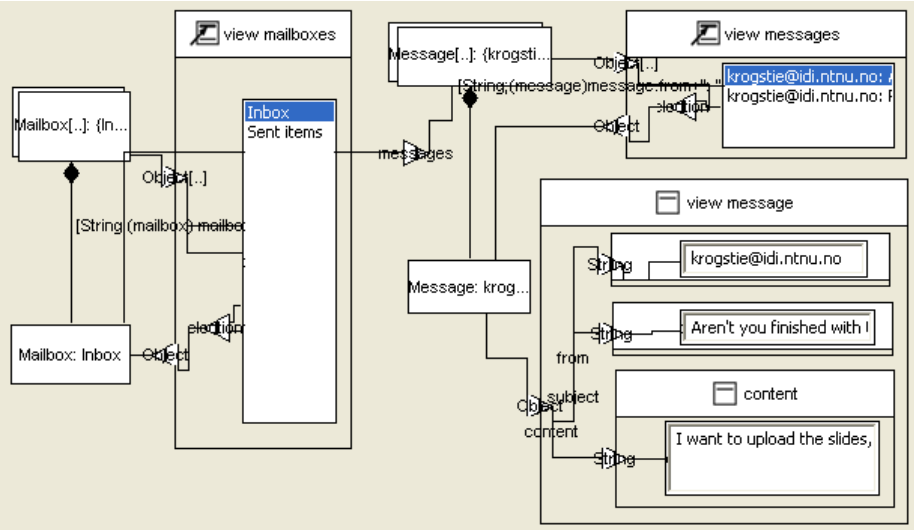

Figure #-6. A testable GUI prototype for the example model

We have tried out both top-down and bottom-up processes in design workshops and in our experience both processes have their advantages and it is important to be able to vary between abstract and concrete representations of the same design. To better support this we are considering ways of quickly hiding different elements of the design, to leave only those that are relevant for the goal of the current design activity. In addition, we are working on ways of generating standalone UIs that still are coupled to the executable model. Currently we are able to generate an XML representation of the widget hierarchy that may be previewed and operated using the XSWT plugin for Eclipse. The XML may be edited to provide a reasonable layout and hence a more realistic prototype, as shown at the bottom of Figure 7.

4.6 The editor implementation

The editor is heavily based on the open-source Eclipse project. First, it is implemented as an Eclipse editor plugin and integrated with the Eclipse Workbench framework, as shown in Figure 7. Second, the Eclipse Modelling Framework (EMF), is used as the basis for the language model. Third, the Graphical Editor Framework (GEF) provides the MVC-based architecture and basic building blocks upon which the editor is built. The core design is based on [9]. Finally, the Standard Widget Toolkit (SWT) is used for the "live" widgets inside the editor.[1] The project is itself open source, and can be found at http://opensource.idi.ntnu.no/projects/diamodl/.

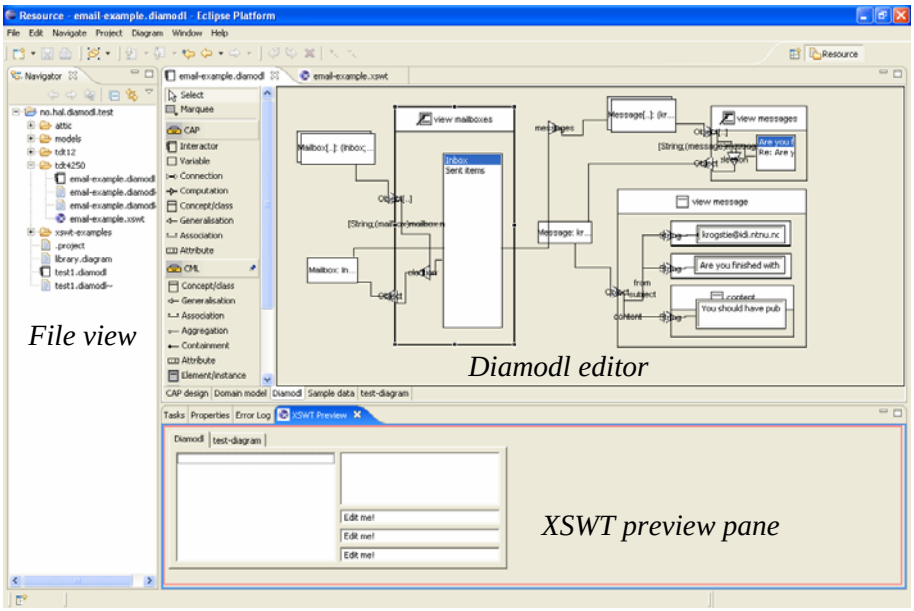

Figure #-7. Eclipse with Diamodl editor and XSWT preview pane

The use of EMF is of particular importance, as it allows implementing a complete language model (concrete diagram syntax + abstract syntax), including an efficient in-memory representation with event and editing services and XMI-based persistence. In addition, our UML subset is mapped to EMF's meta-objects, and the instances and links from the domain model that are used during execution of the GUI prototype are actually EMF instances. To support custom functions in computations and connections, the

[1] For details of EMF, GEF and SWT see http://www.eclipse.org/{emf,gef,swt}

pnuts[2] scripting engine has been extended to support EMF objects "natively". This language and engine was chosen because it had a clean an extensible design, but other scripting languages like BeanShell and JavaScript are easily added.

5. COMPARISON AND RELATED WORK

In order to provide a testable GUI prototype, our hybrid tool combines three elements: 1) user interface modelling, 2) domain modelling and 3) live user interface components and data. In this section, we compare our work with others' covering at least two of these elements.

UMLi/ARGOi [10] extends UML's activity diagram and the ArgoUML tool with support for interaction, hence covering both user interface and domain modelling. Their tool does not however support executing the model(s). The PetShop tool [11] uses an object-oriented variant of Petri Nets for specifying behaviour of an application's UI and supports execution and linking to a real GUI designed in JBuilder. They build on a stronger formal theory, which covers both object and GUI behaviour, so more of the total application logic may be specified. Compared to model-based tools like PetShop, the strength of our tool is twofold: 1) Our interactors are combined with live widgets, so the abstract behaviour in a sense becomes concrete, making it easier to validate specification based on the observed behaviour, and 2) since Diamodl is partly based on UML class, object and state diagrams, it should be easier to learn.

Sketching tools like SILK [12] support live user interface components, and partly cover UI modelling, since they support navigation between views. Such tools try to recognise the designer's intention, to provide additional semantic actions and behaviour. The strength of SILK is twofold: 1) It supports pen-based sketching, which is more flexible than mouse-based object-manipulation and 2) What you see is what you use, i.e. the model is the user interface. Our tool tries to build on the second of these, by making the model "live". We also to some extent support 1) by providing creation tools in the palette which understand mouse gestures, but our tool is not as fluid and flexible as SILK. On the other hand, our tool retains the benefit of model-based tools, by being able to specify more of the UI's behaviour.

In between tools based on formal models and sketching tools, we find CanonSketch [13], which combines quick sketching of CAP diagrams with HTML generation. The intention is much the same as ours, the ability to work on both abstract and concrete representations. Their choice of the CAP

[2] pnuts (http://www.pnuts.org/) is an elegant and flexible Java VM-based scripting language.

notation is based on its level of abstractness: less abstract than formal UI modelling languages, while more abstract than concrete prototypes. Recognising this, we chose to integrate CAP into the Diamodl notation as described in this paper, and CanonSketch's authors kindly provided implementations suggestions and icons for our tool. CanonSketch does not however, support modelling the domain and the deeper UI logic and behavior. The combined notation CAP and Diamodl lets us provide added value: both the ability to specify and execute deeper UI behaviour and support for populating the UI with sample data.

The most wide-spread tools covering two of the mentioned elements are data modelling and database tools with GUI support. E.g. Microsoft Access supports modelling and entering data and building form-based user interfaces. However, such tools have an implicit UI model that is not exposed to the designer, so it is impossible to edit the dialog structure or behaviour, e.g. to add functionality or integrate custom user interface components. One of the few tools that support some level of user interface modelling is Genera's Genova tool. However, their UI model focuses on choice of concrete user interface elements and layout and not on deaper structure or behavior.

6. CONCLUSION AND FURTHER WORK

Model-based user interface design and iterative design are rooted in different traditions. To take advantage of both, we need methods and tools that let the designer combine complementary representations in a flexible manner. Our hybrid tool for model-based UI design and prototyping supports diagram-based, executable user interface models integrated with live UI components, so functional GUI prototypes may be built. The feasibility of the approach has been tested in case studies, and its use in a master course has shown that the combination of CIOs and executable AIOs gives added value to both. We have yet to gain industrial experience with the tool, and further work is needed in two directions.

First, we need a better understanding of how model-based UI design and GUI prototyping may be integrated with current industrial practice. Although the model-based UI design notations are well aligned with IS models, it is not clear how system developers and user interface designers should align their processes. Similiarly, although the tool supports both abstract modelling and concrete prototyping, it is not clear how engineers and interaction designers should utilise the tool in an integrated method.

Second, the Eclipse frameworks, sub-projects and third-party plugins provide many opportunities for integration. We have started to integrate our

editor with the XSWT XML language for SWT UIs, to allow Diamodl to control complete SWT UIs. We need to provide access to corporate data by means of XML and database supports, and are looking at the IBM DB2 plug-ins for Eclipse, and the IBM Integration plug-in for the open-source Derby database. On longer term, the Graphical Modelling Framework (GMF) project recently initiated by IBM and Borland for building EMF+GEF-based editors, may be provide a better platform than our own.

REFERENCES

[1] Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. CAiSE 2005 keynote.

[2] Paternò, F.: Model-based Design and Evaluation of Interactive Applications. Series of Applied Computing, Springer-Verlag London, 2000.

[3] Szekely, P.: Retrospective and Challenges for Model-Based Interface Development. In Ch. Kolski & J. Vanderdonckt (eds.), Proceedings of CADUI'2002, Kluwer Academics Publisher, Dordrecht, 2002.

[4] Myers, B., Hudson, S.E., Pausch, R. Past, Present and Future of User Interface Software Tools. ACM Transactions on Computer-Human Interaction, Vol. 7, no. 1, March, 2000.

[5] Puerta, A.R., Cheng, E., Ou, T., Min, J. MOBILE: user-centered interface building. In Proceeding of the Conference on Human factors in computing systems, p. 426-433, 1999.

[6] van der Veer, G.C., van Welie, M.: Task Based Groupware Design: putting theory into practice, van der Veer, G.C., van Welie, M. Proceedings of DIS 2000, 17-19 August 2000, New York, United States.

[7] Constantine, L.: Canonical Abstract Prototypes for abstract visual and interaction design. In Jorge, J., Nunes, N. and Falcão e Cunha, J. (eds.): Proceedings of DSV-IS'2003. Lecture Notes in Computer Science, Vol. 2844. Springer-Verlag, Berlin Heidelberg New York, 2003.

[8] Trætteberg, H. Dialog modelling with interactors and UML Statecharts - A hybrid approach. In Markopoulos, P., Johnson, P. (eds.): Proceedings of DSV-IS'98, Springer-Verlag/Wien. 1998.

[9] Moore et.al.: Eclipse Development using the Graphical Editing Framework and the Eclipse Modelling Framework. IBM RedBooks, 2004.

[10] da Silva, P.P., Paton, N.W. UMLi: The Unified Modeling Language for Interactive Applications. In <<UML>>2000, York, United Kingdom, October, 2000. A. Evans, S. Kent and B. Selic (eds.). LNCS Vol 1939, p. 117-132, Springer, 2000.

[11] Navarre, D.; Palanque, P.; Bastide, R.; Su, O.: A model-based tool for interactive prototyping of highly interactive applications. In IEEE Conference Proceedings of 12th International Workshop on Rapid System Prototyping, 2001.

[12] Landay, J.A., Myers, B.: Sketching Interfaces: Towards More Human Interface Design. IEEE Computer, vol. 34, no. 3, March 2001, pp. 56-64.

[13] Campos, P.F., Nunes, N.J.: CanonSketch: a User-Centered Tool for Canonical Abstract Prototyping. In Proceedings of ECHI-DSV-IS 2004.