Domain Object Models and User-interfaces

Michael Thomsen

DEVISE, Centre for Experimental System Development
Department of Computer Science, University of Aarhus,
Aabogade 34, DK-8200 Aarhus N, Denmark.
miksen@daimi.au.dk

1. Introduction

The use of models is native to object-oriented development. During development, models of the most important concepts in the problem domain are continually created, modified and extended. Clearly a lot of knowledge of the problem domain lies behind the production of these models, so it seems natural to investigate how much this knowledge can be used in producing the user-interface.

This position paper discusses the use of object models in object-oriented development, and through a case study looks at the relationship between the domain model and the user-interface in this concrete project. From this investigation some preliminary conclusions are drawn.

2. Object-oriented modelling

Object-oriented languages originate from the Simula language developed in Norway in the sixties (Dahl et al., 1966). Simula was designed to be a language for making simulations. It is therefore not surprising that one major benefit of object-oriented languages is an underlying *conceptual framework* providing means for modelling. This conceptual framework provides object-oriented languages with the ability to model the concepts and phenomena in the "real world" that the envisioned system is concerned with. It is important though to remember that the world is not object-oriented; object-orientation is rather a perspective on the world, which of course only captures some of its aspects.

Object-oriented development then, is based on an understanding of the concepts used within the settings that the system will eventually support. The setting we refer to as the *referent system* and the process of translating referent system specific concepts to concepts within the computer system we refer to as *modelling*. The result of the modelling process we refer to as the *model system*, the object model or just the model. Using the model to refer back to the referent system context we denote *interpretation*. (Madsen et al., 1993, pp.289, Knudsen et al., 1994, pp.52).

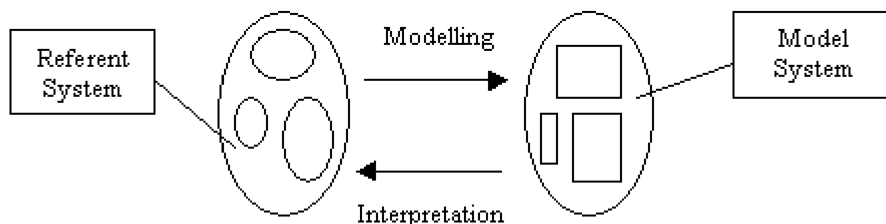

Figure 1. Modelling and Interpretation

Besides the direct benefits from modelling, object-orientation also provides a *unifying perspective* on the whole development process. The conceptual framework provides means for organising knowledge during the analysis phase, object-oriented design notations provide means for design and object-oriented languages facilitate implementation (Madsen, 1996).

2.1 Different model perspectives

When making or interpreting models it is important to remember that these are often made from different perspectives for different purposes. Not realising this can lead to misunderstandings.

Fowler (1997) mentions three perspectives (following Cook & Daniels): *Conceptual, Specification* and *Implementation*. In the conceptual perspective only the concepts found are described and they are described without regard to the actual programming language used. During specification the concepts are further specified. This leads to the interfaces or types of the concepts. Finally in the implementation perspective implementation details are added.

The *Unified Software Development Process* (Jacobson et al., 1999) seems to have a similar division where the first model found is the *Domain model*. This model later influences and is extended by the Analysis, Design and Implementation models.

In summary, there is not only one model system for a given referent system. There is not one thing that is *the object model* and the object model often evolves as development progresses. This is important to notice, as the object model found in later phases most likely will have less direct relation to the problem domain. This can e.g. be the result of non-functional requirements such as time- or space-requirements.

3. The relationship between object models and user-interfaces

Having discussed the use of modelling in object-oriented development we progress to examine the relationship between object models and user-interfaces. This is done via a case study of a concrete development project.

3.1 Introducing the Dragon project

The Dragon project involved the DEVISE research group and a global, distributed shipping company (which cannot be named for commercial reasons). It ran from January 1997 to August 1998 and participants included, from the university: one project coordinator, one participatory designer, one ethnographer, one usability expert, three full-time and three half-time object-oriented developers. Participants from the business included senior management, business representatives from the major continental regions, administrative staff, customer service personnel and members from a private consulting company.

The goal of the project was to create a prototype of a new customer service system, and in this process investigate and evaluate tools and techniques for rapid evolutionary prototyping and user involvement in system development. The project involved many iterations of prototyping and intensive work within a highly distributed company (offices in some 70 countries). An overview of the project is given in (Christensen et al., 1998).

3.2 Object model and user-interface in the Dragon project

The Dragon project provides an interesting case for investigating the relationship between object models and user-interfaces for a number of reasons:

- The object model found even in the running system is very close to a "clean" domain model, that is a model constructed from a conceptual perspective. This stems from the fact that the project was a prototyping project where collection of knowledge of the problem domain and ways of supporting its work processes were more important than performance issues
- The problem domain of shipping is quite complicated or at least not trivial
- Both the domain model and the user-interface where discussed and produced in cooperation with users from the problem domain

As mentioned above the object model found in the Dragon system has very few implementation specific details. We will therefore refer to the model -- also the actual implementation of it -- as a domain model. This domain model consists of 57 classes. Of these there are around 15 classes which are the most important ones.

The user-interface of the Dragon system is a traditional WIMP interface. For each major area of work the system is divided into 13 major, upper tabs (see figure 2. below). The user interacts with each of these tabs by filling in the data-fields and by using the process buttons at the top. The process buttons are used both to call functions and to indicate the current state of the data displayed. Some major tabs also have a number of minor tabs holding details, and furthermore a number of smaller windows besides the large, main window can be opened.

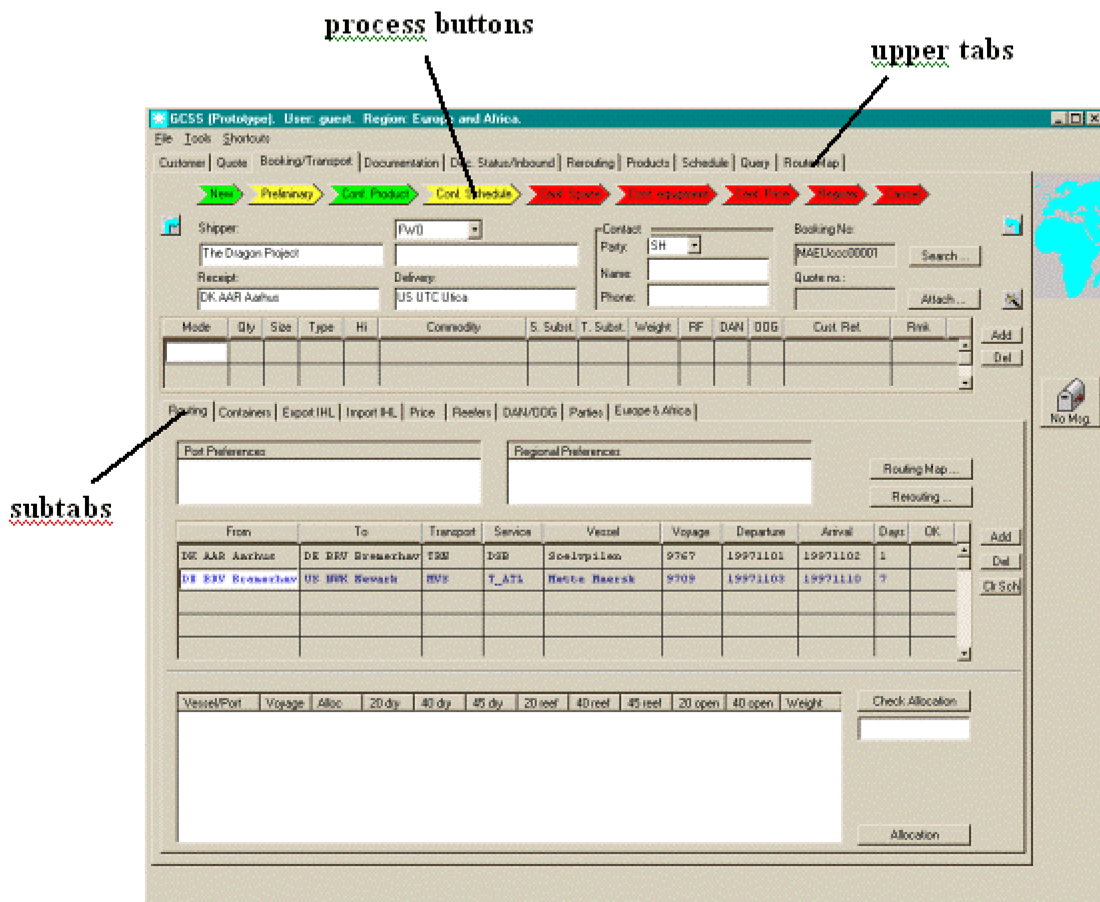

Figure 2. The Dragon system user-interface

3.3 The domain model and user-interface relationship

A comparison of the domain model and the user-interface has resulted in the following observations, which are discussed below:

1. Most of the major tabs each correspond to one important class in the domain model
2. There are some small "help" classes that are not shown in the UI at all.
3. Text, integer and date attributes are represented by a static text label or a editable text field
4. Type attributes, which can have one of a number of predefined values, are represented by an option-button or by a editable text-field that performs a syntax check
5. Singular references/pointers to other model classes are represented by a number of widgets showing important properties of the referenced object and a button leading to another tab showing more information
6. Collections of references/pointers to other model classes are represented by a list-view, having a column for each attribute

Re: observation 1.

Seven of the 13 major tabs (Customer, Quote, Booking/Transport, Documentation, Products, Schedule and Vessel) each corresponds to one important class in the domain model. For all seven tabs almost all data in the tab is found in this one corresponding class. Also each of these tabs/classes correspond to the seven probably most important concepts in the problem domain.

The other six tabs (Stuffing, Doc. status, Allocation, Rerouting, Report and Route Map) correspond to neither one class nor one concept. They rather correspond to one specific, important task that operates on a number of classes.

Re: observation 2.

A small amount of classes (around 10 %) are not shown anywhere in the user-interface at all. These classes hold e.g. configuration properties or they hold duplicates of data found elsewhere that has been computed to increase performance. In general one could question whether such classes should be in the domain model at all.

Re: observation 3.

Simple attributes such as text, integer and data are in all cases in the system represented by either a static text label or by a text field. It seems that a static label has been chosen in all those places where the user either has no interest in or is not allowed to edit the attribute. If this is the case the editable textfield widget has been chosen.

Re: observation 4.

Attributes that can only hold a number of predefined values are represented by either an editable text field (figure 3a) or by an option button/combo box (figure 3b). In the first case the text field performed a syntax check on the input to test whether the input was legal. Finally, in a few cases a combination of the two input options was used (figure 3c): An editable text field with a right-click menu from which the options can be chosen.

Figure 3 (a, b, c). Representing type attributes

Re: observation 5.

In many cases in the domain model, a class holds a reference/pointer to another class. As examples the class `Quote` holds a references to the class `Customer`, and a reference to the class `Standard product`.

These relationships between classes in the domain object are represented in the user-interface by a number of widgets that present a few of the most important attributes of the related concept. In the example above the Quote tab has two text fields holding customer name and id to refer to the related Customer, and two text fields holding receipt and delivery to refer to the related Standard product.

Importantly these widgets do not show all information about the related concept. If the user whishes more information he will have to go to another tab. In both examples above the user can via a button go to another tab that will show all details about the currently related Customer or Standard product.

Re: observation 6.

Besides holding single references to other classes domain classes also often hold collections/lists of references to other domain classes. An example is the `Sea Corridor` class holds a list of references to related `Sea links`.

In almost all cases in the prototype these many related elements where represented by using a list view/table view widget (se figure 4. below). Furthermore the columns in this list view closely corresponded to each of the attributes in the related class.

Base port to base port

From	To	Transport	Days	Service
HK HKG Hong Kon	JP YOK Yokohama	MVS	3	FEME
JP YOK Yokohama	NL ROT Rotterda	MVS	24	AE1

Figure 4. Representing many related elements

4. Conclusions and further work

The case study suggest that domain models are useful input in designing the user interface. The observations showed that important classes in the domain model were very visible in the user-interface; that a class' attributes often were a good indication of what data should be displayed and how it could be displayed and relationships between classes in the domain model were also represented in the user-interface. Further work is needed to see whether these observations also hold for other applications and other domains and to investigate whether some more concrete statements can be made on the relationship. It would also be interesting to see how much more information about the user-interface could be found if the combination of a domain model and a task/use-case model was used.

This author doubts however that we should pursue ways of fully specifying (or generating) user-interfaces from domain models or task/use-case models or both. Many important choises in user-interface development are based on other factors than those encaptured by these models, e.g. factors such as ergonomics, understandability, performance, social conventions, user expertice etc.

Rather we should consider the production of the domain model, the task/use-case models and the user-interface to be separate activities. This does not imply that they should be done in isolation though. Each activity can provide very useful input to each of the other activities and the development can in this way achieve synergy --- a synergy that we indeed often experienced in the above mentioned Dragon project.

5. Acknowledgements

The Dragon Project was carried out in Center for Object Technology (COT) which is a project that has been partially funded by the Danish National Centre for IT Research (CIT) and the Danish Ministry of Industry.

6. References

1. (Christensen et al., 1998). Christensen, M., Crabtree, A., Damm, C.H., Hansen, Klaus. M.H., Madsen, O.L., Marqvardsen, P., Mogensen, P., Sandvad, E., Sloth, L., Thomsen, M.:"The M.A.D. Experience: Multiperspective Application Development in evolutionary prototyping." *Proceedings of the 12th European Conference on Object-Oriented Programming (ECOOP '98)*, Brussels. Springer Verlag.
2. (Dahl et al., 1966). Dahl, O.-J., Nygaard, K. "SIMULA an ALGOL-Based Simulation Language." *Communications of the ACM*, 9 (9), pp.671-678, September 1966.
3. (Jacobson et al., 1999). Jacobson, I., Booch, G., Rumbaugh, J. *The Unified Software Development Process*. Addison-Wesley.
4. (Fowler, 1997). Fowler, M. *UML Distilled*. Addison-Wesley.
5. (Madsen et al., 1993). Madsen, O.L., Møller-Pedersen, B., Nygaard, K. *Object-Oriented Programming in the BETA Programming Language*, ACM Press, Addison Wesley.
6. (Madsen, 1996). Madsen, O.L. "Open Issues in Object-Oriented Programming -- a Scandinavian perspective". *Software Practice and Experience*, val. 25 (S4), december 1995.