Fraunhofer Institut
Experimentelles
Software Engineering

User Interface Paradigms for Multi-View Modeling

Authors:
Louise Scott
Jörg Zettel

IESE-Report No. 032.00/E
Version 1.0
May 2000

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Executive Summary

The multi-view modeling paradigm has been successfully developed and is used extensively in all kinds of modeling environments. A major advantage of the multi-view approach is that it simplifies modeling activities by reducing the complexity of the model in any particular view. On the other hand, the multi-view paradigm introduces an extra dimension of complexity - that of the "view". It is our experience that if the multi-view concepts are not conveyed well by the user interface of a modeling environment, then the benefits of the multi-view approach cannot be fully realized. In this short paper we survey different approaches to supporting views in modeling tools. We use this survey to identify potential problem areas where careful attention should be paid to user interface design.

Keywords: Multi-view modeling, CASE, user interface.

Table of Contents

1 Introduction

The multi-view modeling paradigm has proved a very successful approach for modeling environments. It is now used extensively in modeling tools for software design, process modeling and other areas. One of the great benefits of the multi-view approach is a reduction in complexity of information in any particular view. This allows the user at any time to work only with the subset of information relevant to their immediate task, thus reducing the complexity of the modeling task.

While the multi-view concept reduces the complexity of modeling it introduces another dimension of potential complexity - that of working with views on the model. This increases the complexity of editing models (which must now be done via views) and adds the extra tasks of managing the content of views and navigating around views. The user interface of tools utilizing the multi-view approach must be carefully designed so that these tasks are adequately supported, otherwise the potential advantages of the multi-view paradigm will not be fully realized due to an increase in the complexity of view management. One of the problems which is identified in this survey is that tools currently do not take a consistent approach to these tasks.

Since the multi-view paradigm was first proposed, most of the work in the area has been on technical mechanisms for managing the data and interactions *within* the tool [1]. Little attention has been payed to formalizing user interface mechanisms for supporting multi-view modeling.

This paper investigates paradigms for managing and navigating around information in multi-view modeling environments by considering different multi-view tools. Two of the tools are commercial tools (SELECT Enterprise 6.0[1] and Together/J 2.0[2]) and the third is a Fraunhofer IESE in-house tool (Spearmint 3.0[3]). We identify common approaches and differences in an attempt to find areas where users may have difficulty using multi-view tools.

1 SELECT Software Tools, www.selectst.com
2 TogetherSoft (formerly Object International), www.togethersoft.com
3 Fraunhofer Institute for Experimental Software Engineering, www.iese.fhg.de/Spearmint

2 Concepts of multi-view modeling

Figure 1 shows the basic concepts of multi-view modeling. In the multi-view framework a central or underlying model exists on which many (possibly partial and possibly overlapping) views can be taken. The user edits the underlying model through any of the views.

Figure 2 shows the fundamental operations that must be supported by multi-view environments. Firstly, the user must be able to create, delete and modify elements in the underlying model. These are common modeling operations that are supported in all modeling tools (even if they are not multi-view). Secondly, the user must be able to include and exclude elements from any view without affecting the underlying model. These operations exist only in multi-view environments and they allow the user to create partial, overlapping views on the model.

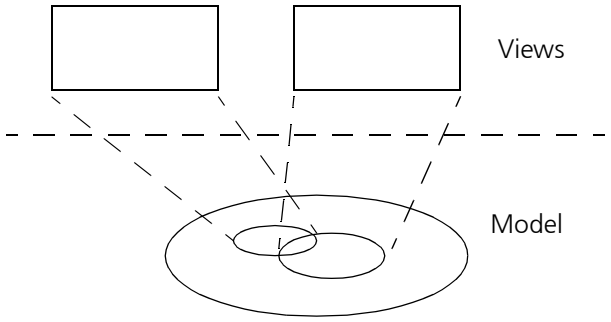

Figure 1: Basic concepts of multi-view modeling.

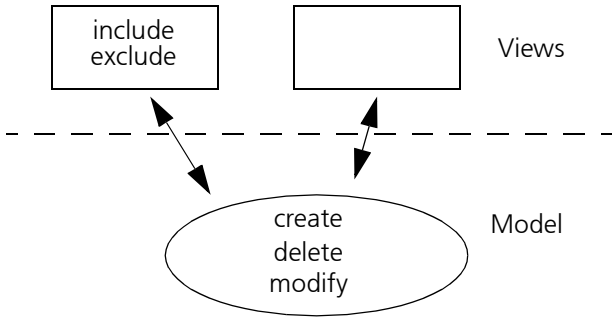

Figure 2: Fundamental multi-view operations.

In the next section we describe how these operations are supported in three multi-view modeling tools. In the section following that, we explore common user interface actions (specifically selection, deletion, clipboard and drag and drop actions) and see how they have been adapted to support the multi-view approach. We identify several problem areas where users may have difficulty understanding the multi-view approach.

3 Multi-view operations

Create and modify

Creating and modifying elements are basic modeling operations. Although the three tools we looked at all employed slightly different user interface actions (in the details) to invoke the operations, we found nothing to suggest that users would be confused by their approaches. Creating an element in a view always creates a new element in the model and is always followed immediately by an include in the view. Thus newly created elements are always visible in the view in which they were created. Modify actions always modified elements in the model and any modifications were immediately reflected in all views in which the element was visible.

That tools have adopted a consistent and intuitive approach to create and modify is encouraging. However, this is most likely because these operations already existed and their adaption to fit the multi-view framework was straightforward.

Delete from model

The delete operation on the model was far more problematic. The tools adopted different approaches to supporting this operation and it was not always clear exactly which operation they were supporting.

For example, SELECT Enterprise adopts a different approach depending on whether you are working in a view or directly on the model (possible through a model "Dictionary"). When in a view, a special popup menu item is provided reading "Delete from model". When viewing the model directly, the standard "Delete" action can be used. In both cases, confirmation from the user is required before the operation proceeds.

Together/J also supports two alternatives. When in a view the delete popup menu item provides a choice of invoking the exclude operation or the delete from model operation (however, when invoked from the delete key the operation is always to exclude from view). When viewing the model directly, the standard delete action always invokes the delete from model operation. Spearmint provides no direct view on the model and so makes no such distinction. When in a view, the standard delete and cut actions invoke the delete from model operation.

There is a lot of potential in these approaches for the user to confuse delete from model and exclude from view which we will explore further when we discuss the exclude from view operation.

Include/Exclude

Include/exclude are operations on views that allow the user to control which information is contained in any particular view and to share information across views. These functions are specific to multi-view environments, therefore we expected to find new user interface approaches to support these operations. What we found were many different and potentially confusing approaches, indicating that there is currently no accepted "standard" way to provide these functions.

For example, SELECT Enterprise provides four mechanisms (at least) to include existing elements from the model into a view.

1) a special popup menu item called "add" which invokes a model browser (a list of model elements). The user can create a new element or choose an element from the list to include in the view.

2) a special popup sub-menu called "Populate" with several items. From this you can include elements in the view based on special semantic relationships, for example, "Populate sub-classes".

3) the standard paste action

4) drag and drop

Together/J provides just one way to include elements in a view which is very similar to the first option supplied by SELECT Enterprise. Using a special popup menu item called "Import" a model browser is invoked that can be used to include (and exclude) elements to (and from) the view.

Spearmint adopts different approaches to include depending on which kind of view is being manipulated.

1) Spearmint has views in which the content is determined semantically and not by the user. On these views, no include functionality is provided to the user (the system automatically includes relevant elements).

2) Spearmint has some views in which the content is determined wholly by the user. In this case elements can be included using an include button which acts on the selection (this is possible because Spearmint selects on the model, it would not be possible if Spearmint adopted a view based selection). Also available on these kinds of views is the ability to include neighbors of a

selected item (this is a semantically based include operation) invoked via a button. Another possibility is to include all of the elements of a model in a view.

The examples above illustrate that no "standard" user interface paradigm for this operation has yet been developed. The approaches are different not only across tools but also sometimes within tools. This is an area which can lead to a lot of confusion for the user if the interface does not clearly convey the meaning of its actions.

We have found similar problems with the exclude operation. For example, SELECT Enterprise uses the standard delete action to invoke the exclude operation. Together/J uses three mechanisms

1) a special popup menu item called "import" that invokes a model browser which can be used to exclude elements (as well as include elements)

2) the delete key

3) the "delete" menu item followed by selection of the "unlink" option.

In Spearmint a special menu item called "exclude" is provided to exclude the selected items from the view. On views where the content is automatically determined, elements are excluded automatically when they no longer meet the semantic requirements for being included in the view.

There is a lot of potential with some of the above approaches for the user to confuse the exclude operation with the delete from model operation, especially when "delete" menu items or keys are used to invoke exclude from view operations. This is made worse by the fact that not all users (especially those unfamiliar with multi-view modeling environments) may understand the subtle differences between the two operations. Tool designers must be careful to explicitly convey the semantics of exclude from view in the user interfaces of their tools.

The above discussion (summarized in Table 1) identifies the multi-view specific operations of include/exclude in/from view as potential problem areas for users using multi-view tools. These new operations and the confusion that can arise between delete from model and exclude from view means tool designers should

pay careful attention to the user interface actions used to support these operations.

Table 1: Invocation of multi-view operations.

Operation	delete from model		include in view		exclude from view	
	on views	on the model	arbitrary	semantically	arbitrary	semantically
SELECT Enterprise 6.0	• "delete from model"[a]	• "delete"	• "add" • "paste" • drag&drop	• "populate"[a]	• "delete"	
Together/J 2.0	• ("delete")[b]	• "delete"	• "import"[a]		• ("delete")[b] • "import"[a]	
Spearmint 3.0	• "delete" • "cut"		• "include"[a]	• automatically • "include neighbors"[a]	• "exclude"[a]	• automatically

a. This is a new user interface action.

b. Together/J makes a distinction depending on how the delete action was invoked.

4 Common user-interface actions

In this section we look at the effect that supporting multi-view modeling can have on common user interface actions, namely delete, cut and paste. We discuss selection first, because although it does not directly invoke system operations, it influences greatly the behavior of other common actions.

Delete

The standard delete functionality is usually provided by a menu item (either in a popup menu or from an "edit" menu) with a keyboard shortcut using the delete key.

The tools we surveyed revealed two possible mappings for delete - to an exclude from view operation or to a delete from model operation. In SELECT Enterprise the delete action excludes the element from the view; in Spearmint the delete action deletes the element from the model; in Together/J the behaviour is not so consistent - the user is given a choice if the action is invoked from the menu (and has the ability to define a default behavior), but if the action is invoked from the keyboard the exclude from view operation is always performed. Implementing different behaviour for the menu option and the delete key is a bad design and has the potential to be very confusing. This and the range of different approaches in the other tools suggests that the distinction between the exclude and delete operations is not well understood and not well supported by multi-view tools.

Cut

The usual behavior for a cut action is a copy action followed by a delete action. Together/J does not provide clipboard functionality at all. SELECT Enterprise does not provide cut functionality. This is probably because this would have meant a copy action followed by an exclude from view operation, which does not make much sense.

Spearmint does a copy action followed by a delete action (that is, the copy followed by a delete from the model operation), which is consistent with the standard behavior of a cut action.

The fact that two of the three tools surveyed did not support cut suggests that the semantics of cut in the context of multi-view environments are not well enough understood by tool builders that they can make a confident implementation.

Paste

Paste can be an include operation or a create operation followed by an include operation. Both alternatives were present in the tools surveyed. SELECT Enterprise adopts the include approach (with the interesting variation that if the element is already included subsequent pastes make the element occur multiple times in the view). This means that the paste actions does not create a new element in the model. In Spearmint the paste action is a create operation followed by an include.

Once again, different approaches suggest possible confusion for the user.

Drag and Drop

Drag and drop is another common user interface mechanism. A drag and drop action is usually either a copy or a cut action followed by a paste action. The only tool in our set that utilized drag and drop was SELECT Enterprise. In this case the approach of copying and pasting was used (probably because SELECT Enterprise does not provide a cut action). This means, according to the semantics of paste in SELECT Enterprise, that the dragged element is included in the target view.

The biggest potential problem with drag and drop would be if it did not behave consistently with the cut and paste actions of the tool. In the case of SELECT Enterprise, this was not the case.

The results of this section are summarised in Table 2. The table clearly shows how different tools map delete and paste to different multi-view operations. There is also confusion within Together/J where the operation invoked depends on how the user interface delete is invoked. These are all potential areas of confusion for the user.

Table 2: Behavior of standard user interface actions invoked on views.

Action	delete[a]		cut		paste		drag and drop	
Operation	exclude	delete[b]	exclude	delete[b]	include	create[c]	include	create[c]
SELECT Enterprise 6.0	X				X		X	
Together/J 2.0	(X)[d]	(X)[d]						
Spearmint 3.0		X		X		X		

a. This is the standard user interface action “delete”.

b. This is the system operation “delete from model”.

c. Followed by an implicit include.

d. Together/J makes a distinction depending on how the delete action was invoked.

5 Discussion

It is our experience that users do not always have a clear understanding of the concepts underlying multi-view modeling environments. Therefore it is essential that the user interface of these environments conveys clearly the concepts and functionality related to multi-view modeling. From our survey we have identified many different user interface approaches to supporting multi-view modeling operations. Each time an inconsistent approach is adopted either between tools or (even worse) within tools, there is the potential to confuse the user about which operations are really being invoked.

A major area of concern is in providing support for new functionality associated specifically with multi-view modeling where no standard approach has yet been developed. This is evident with the many different names used to support include/exclude in/from view. While no standard approach exists, designers should be very careful that their user interface actions are identified with a name that clearly conveys the operation that the action will invoke.

Another major problem area concerns the three-way interaction between the delete from model and exclude from view operations and the standard user interface action delete. Between tools there was no standard mapping for the delete action and even within tools the mapping was sometimes inconsistent. It is not likely that all users will be aware of the subtle differences between delete from model and exclude from view, therefore designers should pay special attention to clearly distinguishing the operations on the user interface.

There is a danger when dealing with multi-view environments that standard user interface actions are mapped to different multi-view operations in different tools. This is compounded by the problem that users may already have expectations about these standard operations from using other (non-multi-view) modeling tools. If the actions do not match the user's understanding or behave non-intuitivly then there is sure to be confusion. The lack of support for standard functions such as cut and paste in Together/J suggests that even the developers have not reached a clear understanding of how these actions should behave. It is therefore unreasonable to expect users to have a clear understanding which implies that even more attention must be paid to the user interface support.

Apart from strict user interface issues, several other interesting issues came to light as a result of our survey. One is that because of multi-view modeling there are two possible alternatives for selection. One is that the selection acts on the underlying model providing a global selection mechanism that is synchronized with the views. The other is that the selection acts on the view only, meaning

that each view has its own selection independent from other views. In SELECT Enterprise and Together/J selection acts on views while Spearmint supports selection acting on the underlying model. One possible use for a global selection is as a navigation aid. Another possibility is to use the global selection as input to user interface actions that may act on other views. This could complement clipboard functions as another means of transfering information. These new possibilities should be further investigated in future research.

Another issue that arises when dealing with multi-view modeling is navigation around views and information in views. All of the tools we surveyed provided some way, for example, to navigate from an element to all of the views containing that element. It is possible that good navigation mechanisms on the user interface can help users to overcome the complexity introduced by having many views on a model, but we have not explored this issue in this paper.

6 Conclusion

It is important that multi-view modeling tools provide good user interfaces to support multi-view modeling so that the advantages of the multi-view approach can be fully realized. We have identified several different approaches to implementing user interface support for multi-view environments. While most tools surveyed adopted the same or similar approaches on many aspects, there were some significantly different and possibly confusing approaches.

Ideally in the future tools will adopt consistent and universal approaches to supporting multi-view modeling. However, before this can take place, additional reasearch on the topics we have identified here is necessary to determine which approaches provide the best support for users.

References

[1] Krasner G and Pope S, A Cookbook for Using the MVC User Interface Paradigm in Smalltalk-80, Journal of Object-Oriented Programming, August/ September, pp. 26-49, 1998.

Document Information

Title:	User Interface Paradigms for Multi-View Modeling
Date:	May 2000
Report:	IESE-032.00/E
Status:	Final
Distribution:	Public