Designing a user interface for relational documents

David Carrington, Tim Jones[*], Anthony MacDonald, Mark Toleman[†] and Jim Welsh

Software Verification Research Centre
Department of Computer Science and Electrical Engineering
The University of Queensland
Queensland 4072, Australia

Abstract

Relations of all kinds play a vital role in the user's comprehension of, and navigation within and between, software documents. User-created relations have the additional role of enabling the user to create and maintain relational documentation that cannot be generated by tools or derived from other relations. In this paper we consider the design of UQ⋆'s user interface for both these purposes, i.e. the presentation, query and navigation of relations in general, and the creation and editing of user-created relations in particular.

1 Introduction

Computer-aided software engineering (CASE) tools have a vital role to play in software development and maintenance. In practice, however, the uptake of innovative CASE tools by software engineers is typically slower than their designers expect, and the benefits are consequently limited. This slow uptake of CASE tools is often attributed to usability concerns, and improved usability of CASE tools is therefore of vital importance.

A software product is captured as a set of software "documents" which represent the software at its various stages of development, typically the requirements specification, architectural design, detailed module designs and compilable code modules. Early CASE tools focussed on support for production of these individual representations. It is now widely recognised, however, that manipulation of relational structures or links between (components of) this set of documents is a vital part of effective software engineering.

Work at The University of Queensland has produced an experimental generic, language-based software development environment, UQ⋆ [Welsh et al. 1994], which is distinguished by its facilities for capturing and manipulating relational structures within and between software documents, and for displaying these structures in textual or diagrammatic form. A 'generic language-based environment', when provided with appropriate descriptions of the languages and representations involved in a development method, provides its users with language-specific support for the method.

The overall architecture of UQ⋆ is shown in Figure 1. The central document server manages the syntactic and relational structures and ensures their persistent storage as appropriate. Syntactic structures are represented as abstract syntax trees while relational structures are arbitrary non-hierarchical connections within and between software documents. The document server makes these structures accessible to a collection of tools. These tools can be categorised depending on whether they interact with the user of the UQ⋆ system or not, and whether they construct syntactic structures or not. A text editor is an example of an interactive and constructive tool while a static semantic analyser is an analytic tool that generates relational information linking syntactic constructs of its input document with error messages.

There are three sources of relational information[1]:

User-determined: Users can define a relation between document segments. An example is a relation between a section of code and its description in a corresponding specification document.

Tool-determined: Analytic tools can calculate relations. An example is a *declaration-use* relation between declarations of identifiers in a software document and all uses of that identifier, as calculated by a semantic analyser for the language.

Derived: A relation can be defined in terms of other relations. Derived relations are calculated dynamically by a Prolog engine in the document server.

[*]Ubilab, UBS AG, Postfach 8098 Zurich. Switzerland
[†]University of Southern Queensland, Toowoomba, Queensland 4350, Australia

[1]Relations are also used in UQ⋆ to represent the edges to be drawn between nodes in diagrammatic documents [Jones et al. 1997] but user interaction with these relations is beyond the scope of this paper.

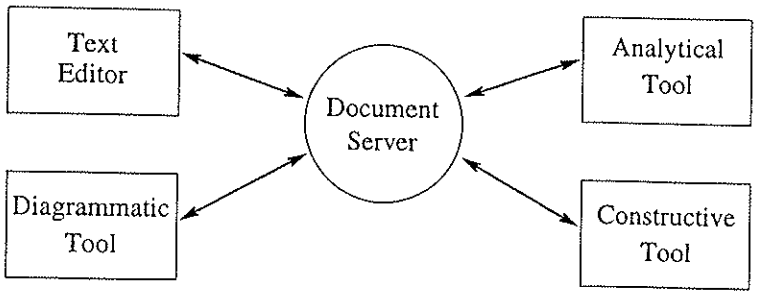

Figure 1: UQ⋆ architecture

2 The design challenge

In their simplest form, UQ⋆ relations often act as hypertext links between document components. Using the presentation and navigation facilities commonly used in web browsers is therefore a natural user interface option. Hypertext links are a particularly simple form of binary relations, which is basically a one-to-one or many-to-one one-way mapping. While UQ⋆'s binary relations between document components are more general than this, achieving the simplicity and efficiency of the click-to-follow paradigm wherever possible is a logical user interface design goal.

However, UQ⋆ relations are not necessarily binary, and do not necessarily involve only viewable document locations. N-ary relations can be created, and the elements in relation tuples can be strings or other attributes rather than references to document components. The underlying model of UQ⋆ relations is thus more akin to that used in relational databases. The presentation of relations in a tabular format, with labels or 'handles' representing references to document components, is a logical candidate for consideration. Tabular presentation is a powerful user interface option, in that it readily accommodates the full range of user interaction requirements on relations: inspection, query, navigation, creation and editing. The Doors document management system [Stevens et al. 1996], which supports the construction and maintenance of relations over document sets for applications such as requirements traceability, uses tabular presentation as the primary means of user interaction with these relations. With this tabular approach, however, simple commonly occurring operations such as hypertext navigation can appear clumsy since they involve the introduction of an intermediate tabular display.

Achieving a seamless integration of the efficiency of hypertext behaviour with the generality of tabular manipulation when required is the challenge in designing an effective interface for relational document manipulation.

3 Requirements and design

In this section we review the requirements and design choices for an effective interface for relational document manipulation in UQ⋆, under the headings:

- static presentation of relations,
- dynamic presentation of relations,
- navigation of relations, and
- editing of relations.

In this extended abstract we focus on the problems posed. The corresponding presentation will include illustration of the solution chosen for implementation in UQ⋆.

In our discussion, we use the following code segment to highlight several presentation issues.

```
1   BEGIN
2       x := y ;
3       z := y * 10 ;
4   END
```

Static presentation of relations. Static presentation of relations is the physical display of relations within documents in UQ⋆. There are two aspects to static presentation of relations in UQ⋆:

- which relations are viewable, and
- how these viewable relations are made visible.

The aspects of a document type and any associated relations that are viewable by the editor are determined by the view description for that document type in the overall EDL (Environment Description Language) [Allison et al. 1998]. Whether and how the relations are made visible at any moment is determined by a user-controllable presentation description, whose default settings are again determined in the EDL. The user can, however, change these presentation settings to suit their purpose at any time.

The visual presentation of relations needs to minimise sensory overload. Visual presentation of

relations focuses on the presentation of the attachment point(s) of a relation in a document. Relations can be marked with highlights (or underlining) of various colours or symbols. However, individually marking every relation attachment point can lead to overloading in several ways:

- too many different colours/symbols for the user to remember, and
- physical crowding of the frame.

However, UQ⋆ relations can be attached to more than arbitrary pieces of text and an attachment point can be related to several lines in a document. This attachment problem is further complicated by the possibility of nested attachment points. Using our example to illustrate this point, consider a situation where a relation exists that has a statement as one attachment point. In the example, line 2, line 3, and lines 1-4 are all statements. While attaching to either line 2 or 3 is straight forward, attaching to lines 1-4 is not so easy as a highlight (or continuous underline) would hide the attachments to lines 2 and 3. We require that not only does the visual presentation minimise sensory overload, but that the presentation should be able to present relations attached to document structures (such as a statement) rather than arbitrary pieces of text and be able to present nested relations. The overloading can be minimised by:

- controlling the number of visible relations via the visibility mechanism, and
- using a single marker for all relation attachments, rather than a different colour or symbol for each relation instance and type.

Neither of these solutions solves the nested relations/large attachment point problem and this will have to be solved by our choice of marker. This marker could have the form of a delimiting outline as shown in Fig. 2.

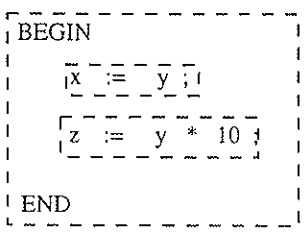

Figure 2:

Dynamic presentation of relations. Navigation of relations can be separated into queries that identify possible navigations and the actual decision to navigate the relation. Dynamic presentation (or querying) of relations prepares for navigation by presenting the relations that are available for navigation from a point within a document.

Recognising what the user has selected is the largest problem associated with the dynamic presentation of relations. The UQ⋆ user views the document as a formatted textual document and the document is internally represented as (and the relations are attached to) an abstract syntax tree (AST). While the user has a general awareness of the document's syntactic structure, they do not necessarily appreciate the precise syntax adopted by the system. This dichotomy between the user's perception of the document and that of the system may lead to selection problems. There are several manifestations of the selection problem, but they all come under the covering of the *fuzzy* selection problem, i.e., the user has selected a section of the document that does not exactly match an AST node or is not the AST node the user requires. For instance, in our example if the user has highlighted half a line, as shown in Fig. 3, does the selection apply to the y or the complete line? The solution

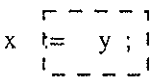

Figure 3:

must both provide a sensible method of choosing which node the user has selected and yet provide the user with a method of viewing the relations attached to nearby nodes (note: by nearby nodes, we mean as the user sees the document, and is not necessarily based on closeness within a tree).

Navigation of relations. An important goal for the navigation of relations is to balance the goals of simplicity and usefulness. In trivial cases, i.e., when there is only one possible destination, the user should have trivial interaction to navigate, i.e., the separation of query and navigation should not exist. The more general case where there are multiple possible navigation destinations, the user should be presented with a useful method of selecting the required destination.

When a navigation operation is performed, the impact on the user must be considered. The effect of the navigation is dependent on the context and the relation being navigated. For example, navigations within the same document may change the contents of the current window, but if the navigation is to a different document, a new window may be opened.

Once the selection problems associated with dynamic presentation of relations have been solved, navigation becomes a straight-forward problem and does not differ significantly from other hypertext-based environments. This will include the provision of a *back* method. While UQ⋆ relations allow navigation in either direction it is not sufficient to allow the user to navigate a relation and expect

them to be able to navigate back using relational navigation.

Editing of relations. Editing of relations is to follow the same paradigm as the editing of documents, i.e., the user should be able to treat relations as first class objects and be provided similar operations on relations as provided for documents. Relation editing can be viewed at two levels:

- the creation/removal of a relation instance, and
- the creation/removal/modification of tuples within a relation instance,

The view defined in the EDL restricts not only the relation types that can be viewed but also the viewable types that can be edited.

The creation of documents requires both the type of document (Modula2, EBNF, etc.) and the name of the document. Similarly, the creation of relation instances requires the type of the relation and a name for that instance. Creation of a relation creates an empty instance. Removal of a relation instance removes all tuples in that relation instance. Removal of a relation instance can have significant and possibly catastrophic consequences for the user and should be handled carefully.

Creation, removal and modification of a tuple each require the selection of a tuple as part of their operation. In the paragraph on dynamic presentation of relations, we discussed the *fuzzy* selection problem. That problem also has to be solved for tuple editing, e.g., we don't want to delete every tuple attached to a node when it is selected for tuple removal, but rather allow the user to select the tuple of interest. This multi-step selection for editing will be similar to navigation.

4 Conclusions

In this extended abstract we have identified some of the problems arising in designing a user interface for relation manipulation in UQ⋆. Solution of these problems is now in progress, and will also be covered in the workshop presentation of the paper.

References

Allison, W., Carrington, D., Jones, T., MacDonald, A., Toleman, M. and Welsh, J. (1998), Environment description language for UQ⋆, Working Document, Software Verification Research Centre, University of Queensland.

Jones, T. S. and Welsh, J. (1997), Requirements for a generic, language-based diagram editor, *Australian Computer Science Communications* 19(1), 316–325.

Stevens, R. and McCaskill, G. A. (1996), Methods and tools for the interactions between systems and software, Technical report, QSS Ltd, Magdalen centre, Oxford, UK.

Welsh, J. and Han, J. (1994), Software documents: Concepts and tools, *Software-Concepts and Tools* **15**, 12–25.