COMPUTERISED SUPPORT FOR REQUIREMENTS MODELLING IN FOOM

Terence C. H. Tan
Paul A. Swatman
Centre for Information Systems Research
Swinburne University of Technology
P. O. Box 218 Hawthorn
Victoria 3122
Email: cisr@swin.edu.au

ABSTRACT

This paper briefly introduces an information systems development/acquisition method, FOOM Swatman & Swatman (1992a) and justifies the need for computerised support for the method prior to further field trials. The paper then describes the requirements of a workbench which may be used to support FOOM and reports the experience gained in using FOOM itself in the requirements analysis and modelling processes.

INTRODUCTION

The wide and continuing acceptance of the results of Boehm's IBM and TRW studies (Boehm, 1976) which indicated that approximately 60% of system faults can be traced to specification errors, ambiguities and omissions suggests that the most promising of the approaches to alleviating the 'software crisis' may be those which focus on the problem of specifying software.
Swatman & Swatman (1992a) describe a framework, illustrated in Figure 1, for an information systems development methodology, widely applicable within the conventional information systems domain, which draws upon a number of established areas of research:

- **socio-organisational contextual analysis** following the work of Checkland (1981, 1989) and Checkland & Scholes (1990) which, in the general case, denies the existence of a single, objective requirements specification waiting to be discovered by the systems analyst
- **the object oriented approach** in which situations are modelled as systems of interacting, encapsulated objects, each object belonging to some classes
- **mathematically formal specification languages** in particular, the object oriented specification language Object-Z (Duke *et al.*, 1991) by means of which the abstract characteristics of classes may be described precisely and unambiguously

All three areas make a contribution to the systems analysis and requirements modelling process. The socio-organisational approach is *relevance-centred.* It is concerned with increasing the likelihood that the impact of intervention within the organisational context would be understood—and, thus, that any intervention would be appropriate.
Although the major contributions promoted by adherents to the object oriented approach are concerned with engineering aspects of the systems development process (such as modularity, information hiding, robustness, reuse, traceability), the approach also offers a contribution to communication between specifier and client, in that object orientation is a natural modelling paradigm (see, for example, Henderson-Sellers & Edwards (1994)).
Formal Specifications were, initially, considered to be a *conformance-centred* approach to software development. They were presented as a basis for a formal 'concretisation' process[10] leading to provably correct code. More recently, formal specifications have come to be acknowledged for their contribution to problem understanding (Hall 1990a; Sommerville 1989; Swatman & Swatman 1992; Wing 1990), their precision offering potential for enhanced communication and evaluation of understanding.

[10] This process is normally referred to as refinement within the specialist literature (see, for example, Morgan (1994)). In this paper
we use refinement in its more common sense of clarification or increasing relevance.

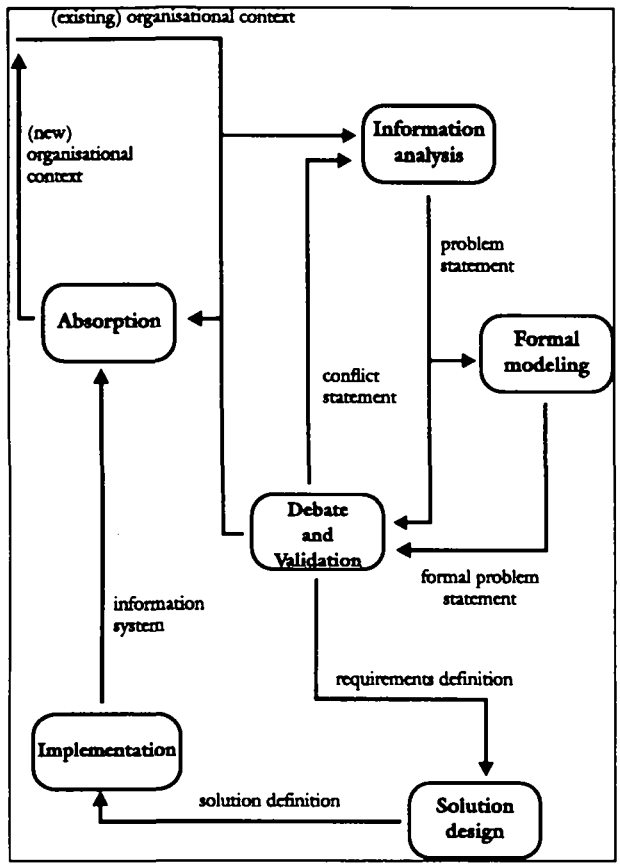


Figure 1: A Framework for Information Systems Development

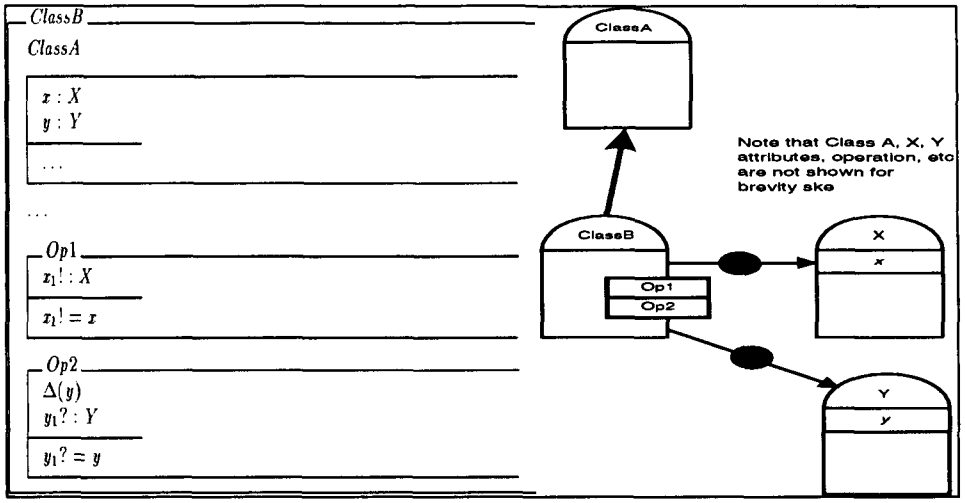
Our preliminary research (Fowler *et al.*, 1993; Swatman 1993; Swatman *et al.*, 1992; Swatman & Swatman, 1992a) suggested that a socio-organisational approach could be used beneficially in concert with object oriented formal specification techniques within the information systems domain.

The FOOM framework was developed by synthesis and logical argument, drawing on research in a number of largely independent areas across the breadth of the information systems and software engineering domains (Swatman & Swatman, 1992a, 1992b). Preliminary evaluation of feasibility and potential benefit were undertaken by means of simulated (Swatman *et al.*, 1991) and small commercial (Swatman *et al.*, 1992) system specifications; and by means of educational case studies and pseudo-laboratory experiments (Swatman, 1993)

THE NEED FOR CASE SUPPORT

The deliverables from the FOOM methodology contain textual, symbolic and graphical material. Although there are tools which support the preparation of conventional text, Object-Z specifications, extended MOSES (Henderson-Sellers & Edwards, 1994; Wafula & Swatman, 1995a, 1995b) and Event Chains diagrams (Fowler *et al.*, 1995), such tools are not, in any useful sense, integrated. Importantly, they fail to take advantage of the strong semantic connection between the different parts of the specification—although it is possible to generate a set of extended MOSES O/C icons relatively easily, from a given Object-Z specification (Wafula, 1995) (see Figure 2).

The preparation of a presentation quality Object-Z specification is currently a tedious process, involving a number of different tools, none of which is designed for ease of use. Preparing an Object-Z specification typically requires a conventional editor such as vi, a graphical editor (we have so far used Corel Draw), a typesetter such as LaTeX or troff, a viewer for the processed document (xdvi, ghostview), possibly a syntax or type checker (fuzz), and finally a program (dvips) to translate the 'device independent' output file to a postscript file suitable for printing.



(a) Class Specification (b) O/C Diagram

Figure 2: Object-Z Class Specification to MOSES O/C

Compounding the problem, the tools are usually command-line driven, each offering unique command switches and user-interface, making them hard to learn and unappealing to most IS professionals.
The project reported in this paper is dually focused:

- The elicitation and specification of the requirements of a FOOM workbench which is intended to alleviate the problems described above by providing an integrated environment, designed specifically for the task of preparing the required documents (a conventional Research and Development project)
- The use and evaluation of FOOM in this context (an action research project)

THE FOOM WORKBENCH

The workbench is made up of the following components (the high level structure of the workbench is illustrated in Figure 3):

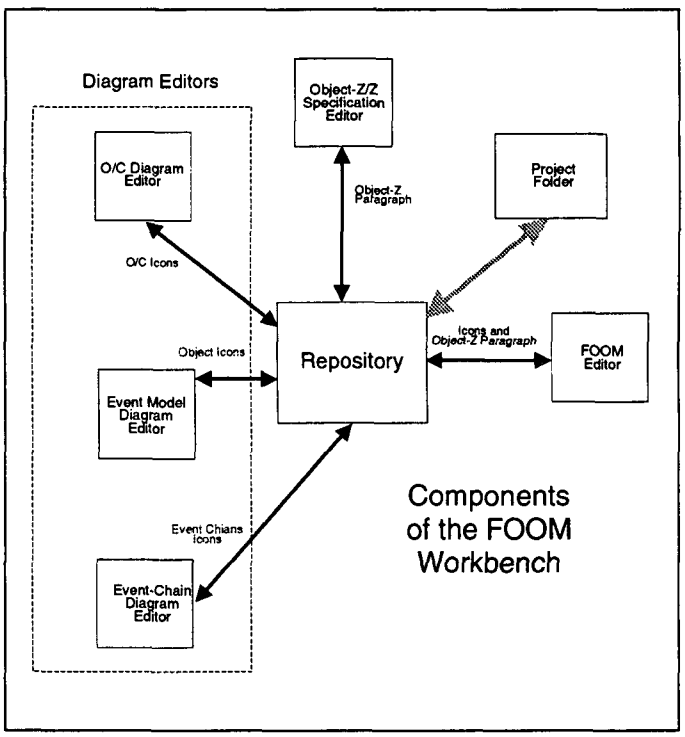


Figure 3: Components of the FOOM Workbench

- **The Object-Z Specification Editor** is responsible for preparing the Object-Z specification of the system under investigation. Besides providing all the required support for editing the Object-Z specification, it handles the decomposition of the Object-Z specification into its subcomponents (Object-Z paragraph).
- **The O/C Diagram Editor** is used for the preparation of the extended O/C Model[11] diagrams, providing all the necessary editing support, such as the ability to connect one class icon to another. The key for retrieval of this information is based on the class's name associated with each icon.
- **The Event Model Diagram Editor** is used to prepare the diagrams which show the dynamic interaction (the sequence of the communicated messages) of objects[12]. This editor is conceptually similar to the O/C Diagram Editor, some of the notable differences being that an Event Model Diagram does not show class inheritance relationships; and an object icon contains additional attributes to represent names and aliases of objects.
- **The Event Chain Diagram Editor** provides support for the generation of diagrams which illustrate system dynamics. The basic shape of the icons used in an event chain diagram is based on the diagrammatic notation of MOSES.
- **The Repository** has an important role in keeping track of the various information provided by the tools. The Object-Z specification and MOSES Icons of each system are kept together in a project folder.
- **The FOOM Editor** itself, which is the tool which combines the different types of information used in the preparation of a FOOM document. Its basic functions are similar to those of a sophisticated word processor or typesetter description

A schematic representation of the class structure of the workbench is set out in Figure 4 and a complete formal specification in Object-Z may be found in Appendix A of (Tan, 1994). In this formal specification, an essentially arbitrary and pragmatic approach was taken to determining the level of granularity. For instance, although the class schema was specified, the classes which formed its subcomponents were not. Further decomposition leading to a specification of finer granularity is straightforward but would, of course, result in a much longer specification. Granularity decisions were made to maximise the communication of concepts while retaining simplicity—characteristics which are sometimes in conflict.

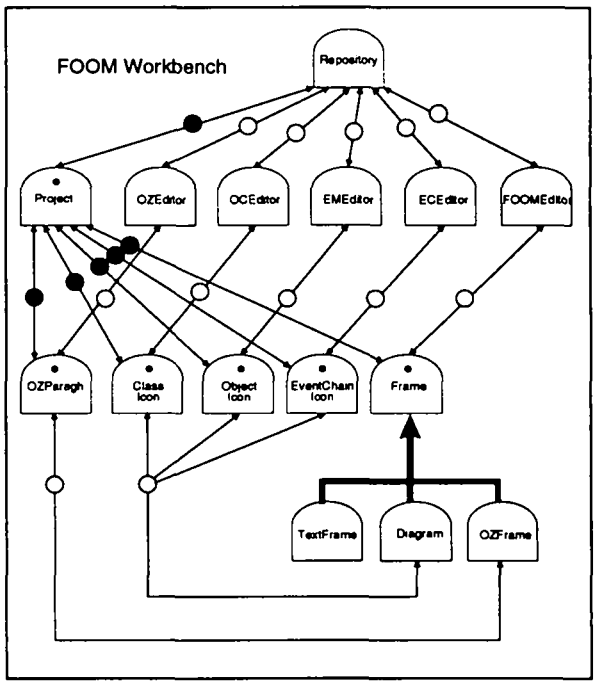


Figure 4: The Workbench Class Composition

[11] The O/C model is primarily used to show class relationships such as the inheritance hierarchy.

[12] In MOSES, the class icon can also be used to show dynamic interaction (see Henderson-Sellers & Edwards, 1994). By contrast, in the Object Diagram notation of Booch (1994), only object icons are used to illustrate the sequencing of messages.

There is an additional commonality between the various tools which is not apparent from Figure 4: the use of a set of standard user-interfaces so that, for instance, the tool-bar and menu-bar look-and-feel would be similar across the system. The inheritance hierarchy relating to the tools is shown in Figure 5.

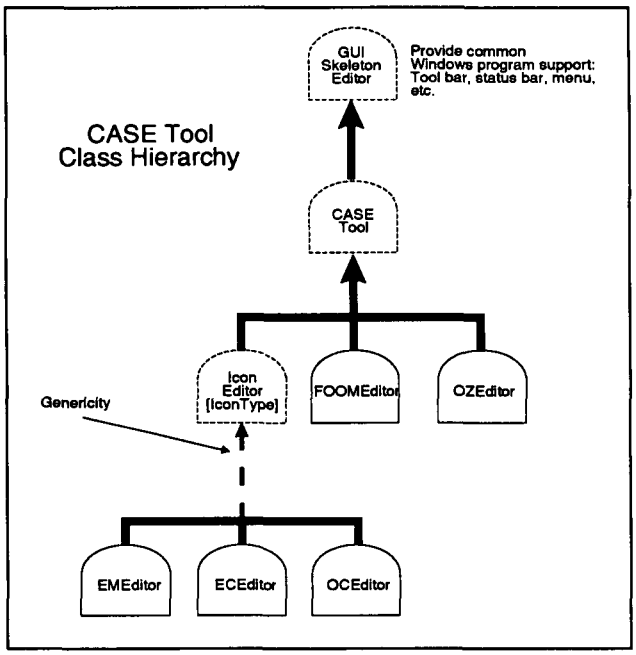


Figure 5: Tools Hierarchy tools

While it is inappropriate to specify the requirements of the workbench in detail here, it is useful to discuss the structure of a general FOOM specification and the way in which the FOOM Editor manages to assemble the resulting document from the fragments stored in the repository.
The functioning of the FOOM Editor depends heavily on the functions provided by its *Frame* objects. Most of the functions provided by FOOM Editor are not native operations, but promoted operations of the *Frame* objects which in turn rely on the services of their component objects. The *Frame* class encapsulates the complexity of different types of information from the viewpoint of the FOOM Editor and, to the FOOM Editor, its operations have been reduced to simply keeping a list of frames—all the hard work specific to a particular type of information is handled by the correct type of *Frame*.

Five types of frame are required in the preparation of the FOOM documents:

- *TextFrame* which allows text to be typed into the document
- *OZFrame* which is used to contain a series Object-Z Paragraphs
- *ClassDiag* which holds a group of O/C icons.
- *ObjectDiag* which holds a group of Object icons.
- *EventChainDiag* which holds a group of Event Chain icons.

The inheritance hierarchy rooted at the *Frame* class is shown in Figure 6 and an illustration of a FOOM document labelled to indicate the classes of objects it contains is shown in Figure 7.

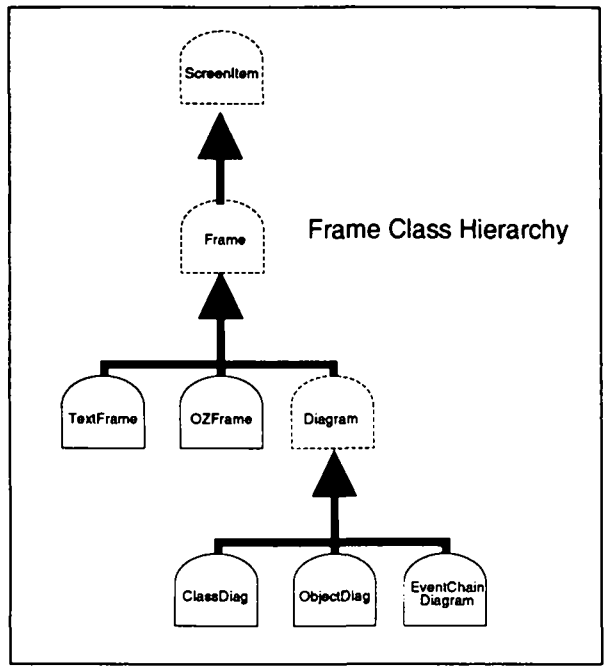


Figure 6: *Frame* Class Hierarchy frame

DISCUSSION AND CONCLUSIONS

The Problem Domain

The initial research plan was:

- To investigate the CASE support required to facilitate further field trials of FOOM, then
- Prepare a detailed specification of the FOOM Editor and the Repository for prototyping.

The first phase progressed smoothly, but as work on the second phase progressed, it quickly became apparent that many of the components of the FOOM Editor also form components of other editors within the system (Object-Z Editor, O/C Editor, etc). The clear potential for reuse, while obviously beneficial from a systems developmental point of view, requires the specifier (and the formal specification) to confront three levels of abstraction simultaneously:
description.

- **System level** The interactions between the editors and the repository.
- **Subsystem level** The composition of the components which gives the editors their functionality.
- **Component level** The organisation of the components (i.e. class hierarchy and composition)

Understanding, then specification of, the components of all other editors within the systems was required earlier within the process than was expected.

Using Soft Systems Methodology

The use of SSM was rather limited due to the nature of the project. In a user-developer situation like this, where time and the project scope did not permit significant consultation with users, conflicts simply did not arise or could usually be solved in a very simple way[13]. An approach consistent with SSM was used however, in the validation of the specification which, was achieved through discussion with members of the Centre for Information Systems Research at Swinburne.

[13] It is easy to establish a consensus with oneself.

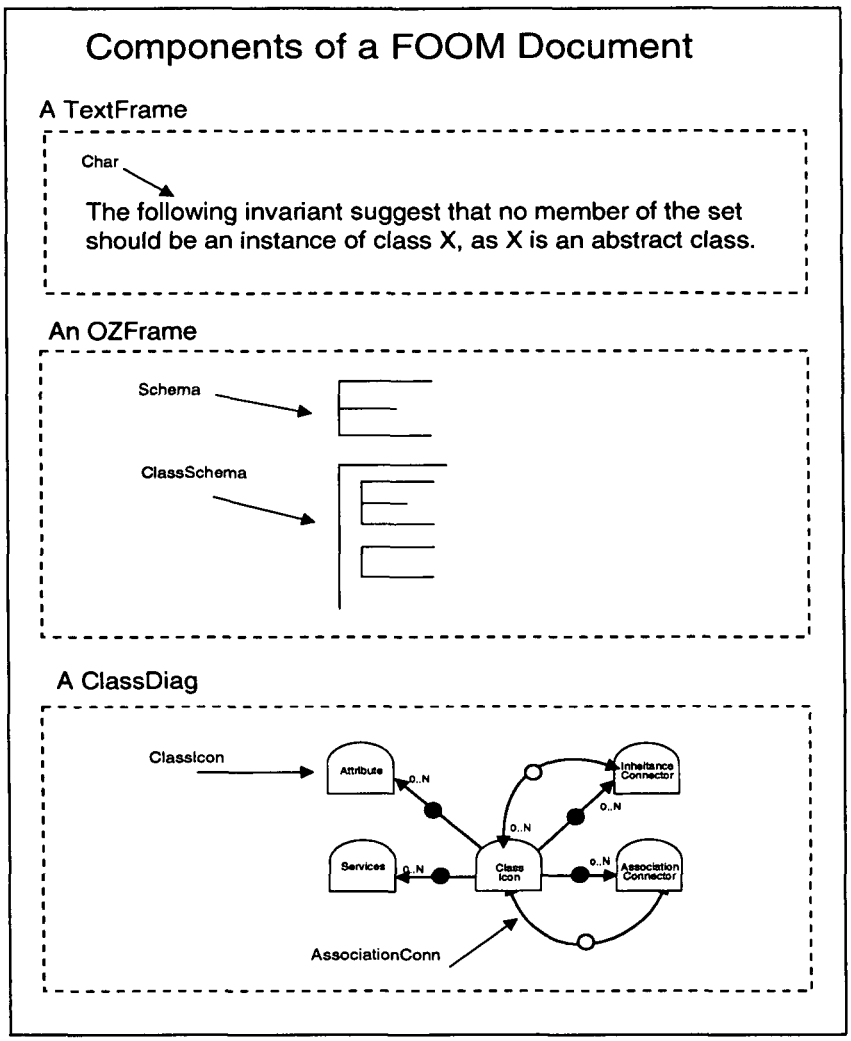


Figure 7: The Components (Objects) of a FOOM's specification

The requirements specification (architectural design) of the repository is essentially a *hard* problem. The term 'hard problem' is used to referred to problems which are essentially *how* questions (Wilson, 1990). The design of the repository is such a *how* question. Although the use of the Soft System aspect of FOOM was limited, it was helpful. We constructed a rich picture[14] and generated a number of root definitions (Tan, 1994, Appendix B). This is an unusually liberal approach to looking at the problem situation. SSM does not dictate what information should be represented, or how the picture will be used[15]. The rich picture focuses on the interactions between users and the human activity system (not necessarily a computer system) and between users and users, which may be difficult to express in a more formal notation (MOSES for instance). Its less constricting and non-prescriptive approach to looking at the problem situation seems to offer the potential of uncovering the needs of the users more completely.

Using Formal methods

The properties exhibited by each of the tools is complex, and deserves to be analysed in great detail. In this project we focussed on two tools in particular: the Repository which forms the basis of the workbench implementation; and the FOOM Editor which is used to assemble the information from the repository together with new information (text), entered directly into the FOOM specification. We considered that the detailed analysis of these two tools would be likely to result in a better design of the system and consequently, a formal specification detailing their relationships was prepared.

[14] This technique is often used to generate a set of views of the system under consideration.
[15] In fact, a rich picture does not even need to be a ``picture' (Kreher, 1993).

Although, the focus of the specification was on the two tools, the specification also included the rest of the tools, as they share common components. For instance, the Object-Z Editor and the FOOM Editor both use the services provided by the *OZParagh* class.
Nonetheless, to constrain the amount of specification to a more manageable level (given the time constraints of the project), the following measures were taken:

- the specification focuses mainly on the exploration aspect of the system as a whole, investigating primarily the hierarchy of classes and the structure of objects in order to determine a good architectural design for the system.
- concurrency issues are not considered in this specification (although it is accepted that support for multiple users is an important issue for an ``industrial-strength'' tool).
- operations and attributes are defined in the classes on an ``as needed'' basis[16]. This measure eliminated many operations in the class specification. The eliminated operations, however, did not affect our insight into the system as similar operations are defined elsewhere.
- some classes were not defined but their basic properties can be observed from the definition of similar classes.

The specification of the repository not only expresses the requirements of the system, it also conveys a substantial amount of design information. For instance, the classes *Frame* and *OZParagh* would certainly be used throughout the development life cycle, and the operations specified would have their counterparts in the implementation phase.
The specification of the workbench was not verified formally (by the proof of theorems) but was validated through a series of walkthroughs. The walkthroughs focused on the structure of the workbench and, in a minor way, on the mathematical correctness of the expressions. As expected, it was found to be difficult to rely just on the formal specification to explain the requirements and the design of the system. The class (O/C Model) diagrams were most valuable in communicating the overall design rationale. One clear advantage of the presence of a formal specification was that it allowed us to simplify the diagrams as the details could be found readily in the specification when required.
We did not find the Event Model of MOSES to be particularly useful in the modelling process because the system did not exhibit the type of complex object interactions which are shown on object interaction diagrams. The reason may be that the current design in Object-Z is expressing primarily the *what* aspect of the system, whereas the Event Model is intended to show the sequencing of events.
Although our experiences using Object-Z in this context were positive overall, there are some areas where we experienced difficulty:

- **Abstract class** The idea of a virtual class (no instance of this class should be allowed) should be made more explicit using a keyword. The advantage of this is that the virtualness of the class can be observed directly from the declaration rather from the invariants of the class.
- **The equality of two objects** The new reference (which has superseded value-based) semantics of Object- makes the definition of equality (as opposed to identity) difficult. By equality we mean that two objects from the same class have the same values in their attributes and constants. The difficulty arises if we test the equality of two objects (say x and y) which have an attribute (say a) which is of class type. For x to be said to be equal to y, do we require $x.a=y.a$ (i.e. that the references are equal and thus $x.a$ is the same object as $y.a$—or merely that the object referred to by $x.a$ is equal to the object referred to $y.a$. Of course, $x.a$ may also have one or more attributes of class type An operator could (we think) be formally defined for value comparison rather than identity comparison—such an operator would have been useful (a informally defined symbol was used to stand in for such an operator within the specification).

USING FORMAL METHODS AND OBJECT-ORIENTATION

The architecture of the workbench was designed using a combination of the top-down and bottom-up approaches. Identification of the subsystems was relatively straight forward as the subsystems are the tools themselves. The identification of the components of tools was, however, considerably more difficult as component behaviour was dictated by more than one subsystem (Editor). For instance, *OZParagh* is used by the

[16] Usually, we tried not to apply this rule unless the addition of the attribute or operation made the system seem unduly complex.

repository, the Object-Z Editor and the FOOM Editor indirectly; and its design has a significant effect on the structure of the system. As another example, the idea of *Frame* was not obvious initially, and the original design of the FOOM Editor manipulated the objects created by other Editors directly. This approach was not very successful as the FOOM Editor was handling too much detail[17]. The use of *Frame* objects makes the management of objects from other Editors more straightforward.

FUTURE WORK

Although the current specification has established an architecture for the workbench—the partition of subsystems, the components of the editors, the class hierarchy of the components, the communications between objects, the delegation (promote operation) of responsibility, etc—more detail will need to be specified before the implementation.
The following list is a sample of critical issues which could benefit from further formal specification:

- **Concurrency issues** Policy of multiple access and update of records.
- **Validation rules** Rules which will validate the consistency of the data should be specified.
- **The Library** folder should be generalised so that each project can have its own library folder.

CONCLUSIONS

Of the three components of FOOM the first component, the SSM approach, was involved least because the project was essentially a hard problem. The other two components—formal methods and object-orientation—played a much larger part, and were found to have complementary roles. For instance, the diagrams provided by the MOSES notation helped to explain and illustrate the behaviour of the system. In general, FOOM was found to be a useful approach to modelling the requirements of this system.
The specification of the workbench reported here is currently being extended and refined into a prototype implementation as part of an ARC funded project at Swinburne University of Technology.

REFERENCES

Boehm, B. W. (1976). **Software Engineering**. IEEE Transactions on Computers, C–25 (12), 1226–1241.

Booch, G. (1994). **Object–Oriented Analysis and Design with Applications**. Benjamin Cummings, Redwood City, California, 2nd edition.

Checkland, P.B. (1981) **Systems Thinking, Systems Practice**. Wiley, Chickester

Checkland, P. B. (1989) **Soft Systems Methodology**. Human Systems Management, 8(4), 237–289.

Duke, R., King, P., Rose, G., & Smith, G. (1991). **The Object–Z Specification Language: Version 1. Technical Report 91–1**, Software Verification Research Centre, Dept. of Computer Science, University of Qld, Australia.

Fowler, D.C., Swatman, P.A. & Swatman, P.M.C. (1993). Implementing EDI in the Public Sector: Including Formality for enhanced control. In J.Gricar & J.Novak, editors, **Strategic Systems in the Global Economy of the 90's, The 6th International Conference on Electronic Data Interchange and Interorganisational Systems**, pages 244–256, Kranj, Slovenia. Moderna Organizacija

Fowler, D.C., Swatman, P.A. & Wafula, E.N. (1995). **Formal Methods in the IS Domain: Introducing a Notation for presenting Object–Z Specifications**. Object Oriented Systems, 2(2)

Hall, A. (1990). **Seven myths of Formal Methods**. IEEE Software, 7 (5), 11–19.

Henderson–Sellers, B. & Edwards, J.M. (1994) **BOOKTWO of Object–Oriented Knowledge: The Working Object**. Prentice Hall.

Kreher, H. (1993) **Critique of Two Contributions to Soft Systems Methodology**. Eur. J. Information Systems., 2 (4), 304–308.

Morgan, C. C. (1994) **Programming for Specifications**. Prentice Hall International Series in Computer Science, 2nd edition.

Sommerville, I. (1989) **Software Engineering**. Addison Wesley, Wokingham, 3rd edition

[17] The specification was, in particular, much less elegant and more difficult to understand.

Swatman, P. A. (1993) Using Formal Specifications in The Acquisition of Information Systems: Educating Information Systems Professionals. In J.P. Bowen & J.E. Nicholls, editors, **Z User Workshop: London 1992**, Workshops in Computing, pages 205–239. Springer Verlag, London.

Swatman, P. A. & Swatman, P. M. C. (1992a). **Formal Specification: An analytical tool for (Management) Information Systems**, Journal for Information Systems 2 (2), 121–160.

Swatman, P. A. & Swatman, P. M. C. (1992b) **Managing the Formal Specification of Information Systems**. In Proceeding of the International Conference on Organisation and Information Systems, Bled, Slovenia

Swatman, P. A., Swatman, P. M. C & Duke, R. (1991) **Electronic Data Interchange: A high–level Formal Specification in Object–Z**. In P.A. Bailes, editor, 6th Australian Software Engineering Conference (ASWEC'91): Engineering Safe Software, Sydney, NSW.

Swatman, P. A., Fowler, D. C. & Gan, C. Y. M. (1992). Extending the useful Application Domain for Formal Methods. In J.E. Nicholls, editor, *Z* **User Workshop: York 1991**, Workshops in Computing. Springer Verlag, London.

Wafula, E. N. & Swatman, P. A. (1995a). **The FOOM Specification Structure: Two Complete Examples**. Centre for Information Systems research, Working paper, Swinburne University of Technology, PO Box 218, Hawthorn, Vic, 3122.

Wafula, E. N. & Swatman, P. A. (1995b) **Merging FOOM and MOSES: A Semantic mapping from Object–Z to Object Communication Diagrams**. Centre for Information Systems Research, Working Paper, Swinburne University of Technology.

Wing, J. M. (1990) **A Specifier's Introduction to Formal Methods**. IEEE Computer, 23 (9), 8–24.