CHAPTER 6

A MATRIX EDITOR FOR A METACASE ENVIRONMENT

This paper is published in *Information and Software Technology,* Vol. 36, No. 6, 1994, pp. 361–371, and is reprinted here by kind permission of Elsevier Science–NL, Sara Burgerhartstraat 25, 1055 KV Amsterdam, The Netherlands.

Preface

This was the first published article of those in this thesis, and describes the requirements for the matrix editor. The description of the matrix editor thus departs slightly from what was actually implemented: see the postscript for details. More importantly, the information presented here about the wider MetaEdit+ environment is superseded by that in the other articles on the environment. There are also some minor terminology differences, in particular the use of ‘complex object’ instead of ‘graph’ or ‘object with a decomposition graph’.

A MATRIX EDITOR FOR A METACASE ENVIRONMENT

Steven Kelly
University of Jyväskylä
Department of Computer Science and Information Systems
P.O. Box 35
FIN–40351 Jyväskylä
Finland
email: kelly@cs.jyu.fi

Research in metaCASE or CASE shells has largely focused on supporting methods by allowing the definition of the concepts and representational symbols used in their diagrams. Little interest has been shown in environments supporting representational paradigms other than diagrams, such as matrices or (hyper-)text. The matrix in particular is often a better format for business information systems, metamodelling, and automatic algorithms for decomposing a system. This paper presents a set of functional requirements for a matrix editor for a metaCASE environment, and suggests a user interface for such an editor, using experience from the design of the Matrix Editor for MetaEdit+, under development in the MetaPHOR project at the University of Jyväskylä.

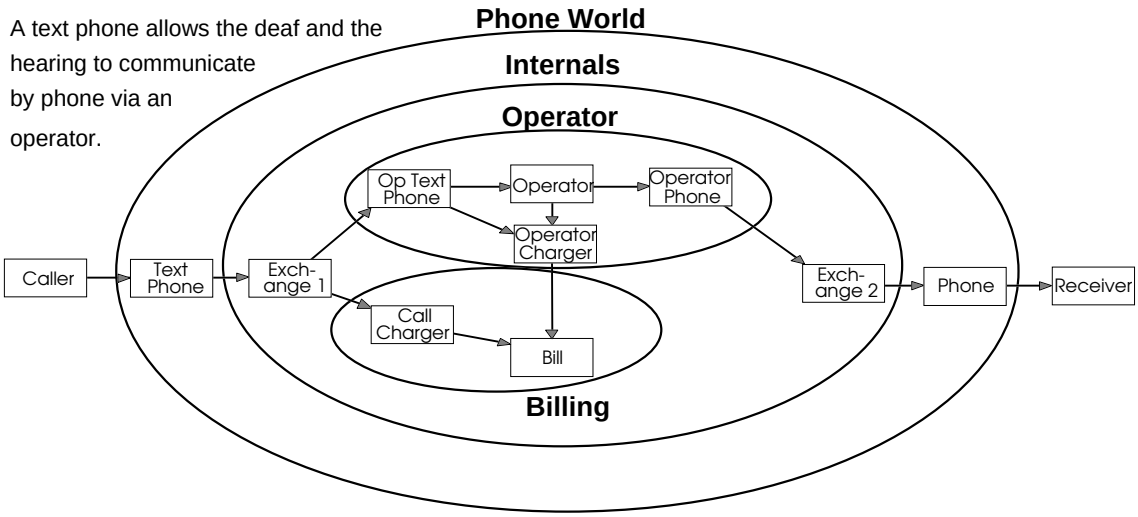
Keywords: metaCASE, CASE, matrix, representation independence, user interface.

1 Introduction

Computer Aided Systems/Software Engineering or CASE tools, much-heralded as the perfect answer to every software designer's problem, have largely failed to have the expected effect on the field. Recent studies[1,2] suggest that the main reason for this observed lack of success has been the narrowness of their approach, which forces the designer to use methodologies supplied as part of the package, rather than allowing him to use and evolve those with which he is already familiar. The response of the industry has been the so-called CASE shells, which allow the user to specify his own methodology or *metamodel*, and then to design with that[3]. However, still more work is needed to support the full spectrum of methodologies, especially in the area of methods that use matrices as a representational paradigm.

The efforts to build method independence into CASE shells are proving successful, insofar as they allow independence from a particular method's graphical symbols, conceptual data, and rules for creating models. This should be extended still further to include full representational independence, so that the user can choose among several basic representational paradigms. Such an approach seems best served by having a multi-tool CASE shell, with each tool able to display the conceptual information in a given paradigm — including graphical diagram, matrix, free-form text and hypertext — and where new tools

can be added. Figure 1 shows the same conceptual graph represented as both a diagram and a matrix. The matrix can be read by scanning across from a row name, via a cell, and up to a column name, e.g. 'Caller' 'O' (outputs to) 'TextP(hone)'. This example graph will also be used at various other points in the paper to illustrate the concepts discussed.



		Phone World										
			Internals									
				Operator				Billing				
	Caller	TextP	Exc1	OpTP	Oper	OpPh	OpCh	CallC	Bill	Exc2	Phon	Recv
Caller		O										
TextP	I		O									
Exc1		I		O				O				
OpTP			I		O		O					
Oper				I		O	O					
OpPh					I					O		
OpCh				I	I				O			
CallC			I						O			
Bill							I	I				
Exc2						I					O	
Phon										I		O
Recv											I	

FIGURE 1 A Text Phone System as a Graphical Diagram and a Matrix

A notable problem in solving the 'software crisis' has been the assumption by developers that they knew what people needed, and their consequent directing of work along lines that are more theoretically interesting than practically useful. It is therefore essential before building a matrix editor to ask whether people need it, and for what purposes. In this respect, we should examine case studies of researchers who have experienced the difficulties of using matrix methods without a dedicated matrix editor, and can reveal the areas where such a tool could be most useful.

Two such researchers, Bidgood and Jelley[4], give compelling evidence that a matrix editor is needed. In their case study, they used a spreadsheet

application as a makeshift matrix editor, putting business processes on one axis and business data on the other, and entering in the relevant cells values indicating whether the process created, read or updated the data. They then performed various re-orderings of the axes by hand, and made the following comments:

1. "With a matrix of any size, clustering the cells is a complex, iterative task";
2. "names were replaced with code numbers to avoid [sorting based on] preconceptions"
3. of differentiating creates, reads and updates "recording all four… produces a cluttered-looking matrix"
4. the definitions of and divisions between individual processes had to be worked on simultaneously with, but separately from, the matrix: any updates to one had to be propagated by hand to the other.
5. "A matrix manipulation tool would have been extremely useful"

A matrix editor which automates the clustering and sorting provides an effective answer to the first two problems. The ability to record all information and choose a subset for any given display prevents the cluttering effect mentioned in the third comment, without sacrificing useful data. The fourth problem is solved by having a matrix editor as one tool among many accessing the same data, so that work is consistently and automatically updated between tools and users. The fifth comment confirms the impression given by the preceding four, namely that a matrix editor as part of a CASE shell is sorely needed right now for practical applications.

Similarly, previous CASE applications with matrix representations have lacked functionality. Meta System's PSL/PSA[5] (see also the paper by Teichroew and Hershey[6]) could perform matrix manipulation algorithms, e.g. to divide a system into subsystems, and produce output in matrix form, but changes made in the matrix could not be propagated back to the database. Cadre's Paradigm Plus[7] has a matrix editor providing a view on the database, and allows the user to add and delete relationships. However, the matrix always covers all instances of two user-selected types in the repository: there can be no mixing of, say, Data Flow Diagram Processes and Stores on the same axis, still less Processes and Flows (as required by e.g. Steward's use of Two Entity Data Flow Diagrams as matrices[8]). Further, the display is restricted to greying the cell to indicate the presence of a relationship or number of relationships: no other information can be shown. Similarly on the axis only the name of the object can be displayed, and there is no facility for building groups, or algorithms for decomposing the system into subsystems. Arthur Andersen's Plan/1[9,10] has such functions, but is limited to their fixed set of matrix types and single methodology.

This paper discusses the requirements for a tool to manipulate matrices as part of a CASE shell, and looks at how ideas for such a matrix editor are progressing in a real CASE shell, MetaEdit+. In the next section, we will look at the MetaEdit+ CASE shell, and at the methodologies which make use of

matrices. After that, the functionality of a matrix editor as a tool in a CASE shell will be described, and in the next section we describe a user interface for a matrix editor as part of MetaEdit+. Finally conclusions are presented, along with ideas for further research.

2 Background

2.1 Overview of MetaEdit+

2.1.1 Data model

MetaEdit+ is a multi-user, multi-tool, meta-CASE environment under development in the MetaPHOR project[11]. It is based on the earlier MetaEdit, which used the four concepts of the OPRR model[12,13](described below) to model design methods, and also as the underlying data model for the design diagrams themselves.

- **Properties**, which appear as textual labels in diagrams, contain single data entries such as a name, text field or number.
- **Objects**, which appear as shapes in diagrams, contain properties and model concepts such as an Entity in an Entity Relationship Diagram or a Process in a Data Flow Diagram.
- **Relationships**, which appear as lines between shapes in diagrams, contain properties, and model concepts such as a Data Flow in a Data Flow Diagram.
- **Roles**, which appear as the end points of Relationships (e.g. an arrowhead), contain properties, and model the part an Object plays in a Relationship, such as which end of a relationship is 'to' and which 'from'.

For MetaEdit+, the OPRR model has been extended to GOPRR[14], for instance by the concept of Graph, corresponding to a single diagram in other CASE tools, and by allowing objects to have a recursive structure.

This fixed conceptual meta-metamodel forms the basis on which varied representations of data, including not only the usual graphical diagrams but also hypertext, text and matrices, can be built. This allows the application to support a wider range of existing methodologies, and also allows customisation closer to the in-house methodology of any given organisation.

2.1.2 Architecture

The basic architecture of MetaEdit+ is illustrated in Figure 2. The heart of the environment is the MetaEngine, which handles all operations on the underlying or conceptual data. This allows the addition of extra tools, each only responsible for its own paradigmatically different view on the same underlying

data, thus introducing *representation independence*. A tool, as the term is used within the MetaPHOR project, is a type of window with associated functionality, with which a user can view and possibly alter a design in a particular way.

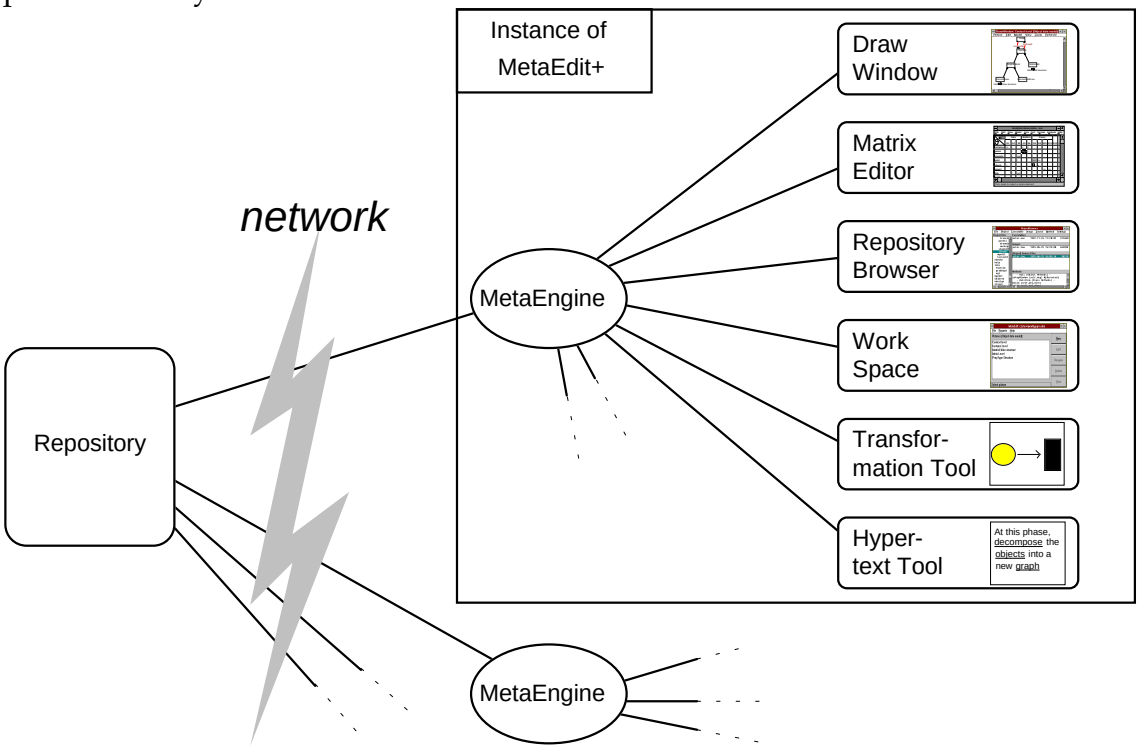


FIGURE 2 MetaEdit+ Architecture

MetaEdit+ can run either as a single-user workstation environment, or simultaneously on many workstation clients connected by a network to a server. At each client is a running instance of MetaEdit+, including all its tools and the MetaEngine, which takes care of all issues involved in communicating with the server. At the server is a repository holding all the data contained in models, and also metamodel, user and locking information.

The various tools communicate with each other through the MetaEngine, and through the shared data in the repository. These tools are:

- the Work Space, which controls access to graphs and metamodels;
- the Draw Window, where graphical diagrams are edited;
- the Transformation tool, which produces textual descriptions of the entities stored in the current model;
- the Query Editor, which allows the user to formulate graphical queries on the database;
- the Repository Browser, which allows hierarchical access to any data stored in the repository;
- the Hypertext subsystem, which gives the ability to add notes and links to any design element in any other tool;
- the Matrix Editor, which is described further in the sections that follow.

2.2 Matrix methods

Within the field of information systems development, the use of the term 'matrix' is somewhat different from that used in mathematics. Both share the concept of a grid of values, whose location can be specified by their co-ordinates, an enumeration along the horizontal and vertical axes. In ISD, this is then extended by using objects, rather than whole numbers, as each item on an axis. These objects possess properties, and there may be some hierarchical structure above the objects. Each element of the matrix can be any (result of a function on some) item(s) present in the model, and is determined by a function involving the items that exist on the axes for its row and column.

Many early software design methods had or have evolved matrix representations. These largely fell into disuse, the graphical approach being preferred and fashionable at that time. The development of graphical user interfaces further favoured the use of the diagrams in CASE tools, and hence they mainly supported the graphical side of these methods. The matrix side is still, however, as useful as it was originally. Technology has now advanced, especially in the area of databases, and can support a single conceptual graph represented as both a diagram and a matrix, so the user can benefit from the matrix representation in just those areas where it is an easier paradigm than graphical diagrams.

Further, in fields closely related to software engineering, such as systems and business process (re-) engineering, methods have developed that rely heavily on matrices. Support for these is limited to proprietary, fixed-method CASE tools, often produced by the organisation that developed the method. This approach reduces the flexibility of the program and the ability of the user to do things his way to best suit the current contingency.

The uses of matrices in methodologies can be divided into three classes: those methods that use matrices as an integral part, such as IBM's Business Systems Planning[15], those that mainly use diagrams but where the use of matrices as an alternate representation is recognised, such as Two-Entity Data Flow Diagrams[8], and methods for which there is no explicit use of matrices, but for which applications of matrices can profitably be found.

In this section, we will look at these methods, and the uses of matrices within them, in the above order.

2.2.1 Mainly matrices

Within the field of business engineering there are many methods which use matrices, for instance IBM's Business Systems Planning[15], Andersen Consulting's Method/1 which is supported by their Plan/1 suite of software[9,10], information engineering[16], business information characterization study[17], and relational modelling[18]. Also, Bidgood and Jelley[4] used matrices exclusively for their Strategic Information Systems Planning exercise.

2.2.2 Diagrams and matrices

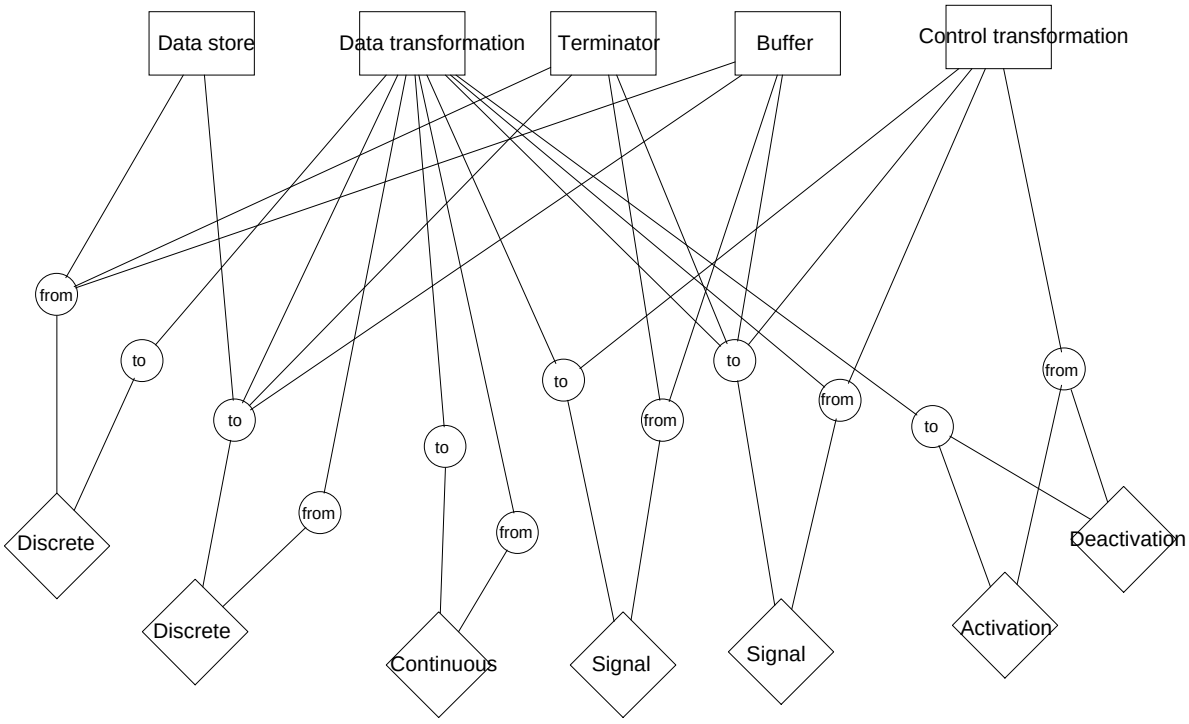
Many early software design methods used matrices alongside graphical diagrams, as an alternate representation useful for certain phases of analysis and design. For instance, Data Flow Diagrams can be represented as N-squared or Lano matrices[19], and there is a similar mapping from Two-Entity Data Flow Diagrams to matrices[8,20]. MacDonald and Palmer[21] present a methodology which includes two different matrix types, along with several complementary diagram types. There also exist standard algorithms for representing and manipulating Petri nets[22] as matrices, including reachability, deadlock and liveliness analyses.

2.2.3 Other uses of matrices

Whilst in graphical methods the main focus of attention is on the objects, in a matrix we have a relationship-oriented view: most of the display is given over to the relationships; to create a relationship requires only a single selection (as opposed to selecting both objects in a graphical editor); any number of relationships can be added without crowding or expanding the picture; and relationships can be added without concern for how they must be routed around objects.

There exist standard algorithms for manipulating a matrix to show the interrelationship of its members, for instance transitive closure, diagonalisation and affinity analysis. A hierarchical axis allows grouping of items, and the user can decompose groups to reveal detail. More decompositions are possible than in a diagram, where the overall picture quickly becomes unclear. As the representations are in general simpler than in graphical notations, screen operations should be faster. The matrix format can be adopted for other formats of similar appearance, for instance displaying text in tabular format, e.g. business information analysis and integration technique[23], and ISAC[24].

Finally, the matrix format is well-suited for metamodelling: the main concern is the names of the items and the connections between them, rather than any particular layout. Also, the number of relationships in a metamodel is often high compared with standard models, making graphical representations clumsy and confusing, when a matrix would still be clear (e.g. Figure 3 shows the legal bindings of relationship (diamond), role (circle) and object (rectangle) types for the Real Time Structured Analysis metamodel, represented as both a diagram and a matrix).



	Control Transformation	Data Transformation	Terminator	Data Store	Buffer
Control Transformation	signal	signal, activation, deactivation	signal		signal
Data Transformation	signal	signal, discrete, continuous	signal, discrete	discrete	signal, discrete
Terminator	signal	signal, discrete			
Data Store		discrete			
Buffer		signal, discrete			

FIGURE 3 Real Time Structured Analysis OPRR Metamodel, as a graph (properties hidden) and a matrix (showing relationship names)

In a metamodel with several graph types, object types would be on the axes, grouped into graph types, relationship bindings would be shown within groups, and links between graph types (e.g. 'Process explodes to Data Flow Diagram' or 'Process shares properties with Entity') could be inserted between items and groups, or between items in different groups (respectively for the examples). At this method integration level, those object types which did not have any links outside their graph type could be hidden for compactness and clarity.

3 Functionality

In research into metamodelling, one must constantly be aware of the distinction between the underlying concepts contained within a model or metamodel, and any surface level representation of those concepts. This distinction becomes felt all the more keenly when expanding from a single-user, single-tool architecture to a situation where many users require access to the data, and the data can be viewed in many representations in different tools[25]. A similar distinction also appears at the level of application or tool design, where the design task is broken down into those parts concerning the underlying concepts and functions of the tool, and those concerning the representation of that tool to the user. In this section we will deal with the Matrix Editor with respect to the former, the functionality, and in the next section the latter, the user interface.

3.1 Groups

One of the benefits of a matrix representation in design is that it allows a large amount of information to be clearly arranged and manipulated at once. This allows the designer to make a better decomposition of the problem area into subsystems. Consequently, the matrix editor supports the operations of that phase of design: grouping together related entities, hiding and revealing detail at will, ordering the entities or groups by hand or automatically by algorithms, etc.

On the instance level, in keeping with the GOPRR paradigm, these groups are nothing more than complex objects: objects which in turn are composed of objects and/or further complex objects. This means that the groups can freely be used in other tools, for example in the Draw Window. On the representation level, however, only the matrix editor need know the extra information for the display and functionality of a hierarchical axis.

3.1.1 Decompose and condense

The *decompose* operation takes an object which is (or will be) made up of other objects and shows those component objects, along with any associated relationships. The reverse operation, *condense*, shows the parent object of the selected object(s), or, if none yet exists, collects the selected object(s) together into a new object.

Both decompose and condense take place without opening any new windows, and therefore other surrounding objects must allow enough space for the operation, or be moved. Whilst this presents problems in the Draw Window, in the Matrix Editor it is nothing more than an insertion or deletion of rows or columns: a common operation on spreadsheets, and simple to perform.

3.1.2 Explode

The explode operation, as its name suggests, is a somewhat more forceful version of the closely-related decompose. When a complex object is exploded, a

new tool window opens, within which further detail of the object is revealed. This window is not a descendant of that from which the explosion was requested: its only connection to its invoker is that of the shared data of the exploded object, as stored in the repository.

This definition leads unfortunately to an asymmetry: although for decompose there is a reverse operation, namely condense, for explode there can be no such operation. The action has been a creation rather than a change, and thus the semantically correct reverse is simply to close the new window. This, of course, can be accomplished by the normal Windows and tool closing actions: as the invoking window has no control link to the new window, there cannot and should not be the possibility to close the new window from there.

From the point of view of matrices, the explode operation is the poor relative of the decompose and condense operations. It has no matrix-specific functionality, merely opening a window for whatever tool the exploded object's detail was designed in.

3.2 Representational graph type conversions

Within MetaEdit+, a representational graph is any display of the conceptual data, as shown in a particular tool. There are several representational graph types, including diagram (displayed in the Draw Window), and matrix (in the Matrix Editor). As the representations of concepts in the tools are different, so the representational data stored by each tool for its graph and components is different. It would appear useful for parallel versions of the same conceptual graph to be maintained automatically as different representational graph types: a diagram, say, which could also be viewed as a matrix. However, because the data requirements are different, any change to the data within one graph type would normally require additional information within the other graph type, which could be difficult to calculate automatically. For instance, adding a new object in a matrix should cause that object to appear in the diagram, but the position information for the object in the diagram would be missing. Sayani[26] provides a good general overview of such problems.

The basic approach taken within MetaEdit+ is to allow the user to take a representational graph of one type and create a graph of another type from it. This ensures that should a user have a diagram-type graph, and decide he wants to view it as a matrix, say, he can do so. Clearly he could make updates in the matrix, and then convert this matrix to a diagram, with the same net result as if the two had been completely linked. This process will be referred to as a graph type conversion, to distinguish it from the transformations performed by the Transformation tool.

Each tool is responsible for adding the representation information it needs: in other words, the receiving tool in a conversion performs the conversion. Thus it is also possible for particular tools to implement automatic updates: for instance, the Draw Window could reflect the addition of a relationship and its two roles in the Matrix Editor by simply drawing it as a straight line. In this

case the user could either move the line to a satisfactory place later, or invoke the graph type conversion to rebuild all the diagram.

3.2.1 Diagram to matrix

In the most general case, creating a matrix begins with selection of the objects to be used as axis items. For a given matrix, there may be more than two axes, although each matrix can only display two of its axes at once, taking a slice or projection along the others. The items along an axis are not necessarily of the same type, and hence the possible relationships for each element in the matrix will vary according to that element's row and column item types. Once all the axes' object items have been chosen, the two display axes are selected, and slice or projection operations specified for any remaining axes. The matrix can then be drawn in its own window: a new window is opened for each new matrix.

When creating a matrix from a diagram, the choice of objects can be partially automated. In the simplest case, all the objects in the diagram are placed along both the x and y axes, and the roles for relationships contained in that diagram are shown as the elements (note that because there are two cells for each pair of objects, A→B and B→A, the elements are roles rather than relationships). The order of objects along the axis defaults to alphabetical order of object type then object name: other sort orders can be applied once the matrix has been created.

It is also possible to select which types of objects or individual objects should be included, and to build up each axis separately in this way. Multiple diagrams can be used to form a single matrix, as indeed can any collection of objects.

3.2.2 Matrix to diagram

When converting from a diagram to a matrix, there was little information to be added, and for that which was required, viz. the sort order, a simple hierarchical alphabetical order formed a useful starting point. When going from a matrix to a diagram, however, all the position information is missing, and the algorithms required to automatically position objects and route relationships are long, complex, and not always effective. Despite considerable interest in this area — di Battista et al.[27] list 265 references — the problem has remained largely unsolved, and perhaps the best approach from a metaCASE point of view is to use a combination of one of the many minimum crossing grid-based algorithms, and placement and direction heuristics with arguments provided both by the metamodel and by the user. Such an approach, based on energy functions and simulated annealing, is presented by Davidson and Harel[28].

3.2.3 Text ↔ matrix

As yet, no common exchange format exists for CASE matrix representations. The CDIF standard[29] deals only with conceptual data and graphical representational, not matrix representational: however, it is possible to build a matrix with reasonable ease from the conceptual data alone.

As a matrix has much in common with both spreadsheet and database formats, it seems reasonable to adopt the common exchange format of those, at least as a basic starting point for information portability. Thus, it should be possible to both import and export matrices as flat format ASCII files, with columns separated by some delimiting character (normally a TAB), and rows by a new line. As neither the databases nor the spreadsheets support hierarchical axes, there is no need to complicate the interface format by adding support for these: only the lowest level axis items will be exported.

3.3 Axis algorithms

The operations which can be performed on matrices split into two classes: those that only affect the order of items on the axis, and those that both affect the order and can also form new groups of items. In the former class are sorting the axis items based on the contents of one of their properties, and diagonalising the matrix to bring the largest possible number of a type of relationship as near as possible to the diagonal. In the latter class are the operations of affinity analysis and transitive closure, which are explained in more detail in the following sections.

Clearly, the user could perform the same operations as the algorithms, and could even split a system into its subsystems in a graphical diagram. As discussed in the introduction with reference to Bidgood and Jelley's work[4], the benefit of these automatic algorithms to the software or systems engineer is that they are fast, accurate, and have no preconceptions about what the subsystems should be — often such preconceptions will lead an engineer to a fair solution, but blind him to the optimal one. However, no algorithm is perfect, and so the user can of course fine tune the results of the orderings or groupings produced automatically.

3.3.1 Sorting

Sorting the axis items based on their characteristics, rather than those of relationships or roles between them, is a useful initial step in dividing up the items into subsystems, as well as for presenting information in a logical format. A simple alphabetical ordering is insufficient in many cases, and should be augmented with orderings based on values of properties of the items, or the items' types. The sort order function on these could be alphabetical, numerical, chronological, or, for those properties which take one of a fixed set of values, by some user-defined order on those values: for instance, in Booch's Class Diagrams[30], the items could be ordered on the common property of Visibility, with values of Imported, Exported and Private.

Sorting can be applied either recursively inside all groups, or only on those groups which are selected.

3.3.2 Diagonalisation

Diagonalisation is a simple and fast algorithm that rearranges either rows or columns to bring cells containing a selected relationship type as close as possible to the diagonal which runs from top left to bottom right.

Diagonalisation can be applied only on a single level of a single group: this can of course be the whole matrix, if no groups have been defined yet. This restriction maintains groups that the user may already have created. The result of diagonalisation is a matrix where all elements of likely subsystems are close together, making forming groups a simple task, whilst still allowing the user the freedom to choose exactly where the boundaries of groups should fall. This procedure is widely used in methods such as IBM's Business Systems Planning[15].

3.3.3 Affinity analysis

Affinity analysis is similar to diagonalisation, in that the rows or columns are rearranged according to the relationships in the cells. However, the process is refined one stage further by associating values with each relationship (or role) type: the higher the value, the more the corresponding row and column items are considered to be related. These values can be defined by the metamodel or the user, and allow a measure of customisability to the algorithm.

Affinity analysis is normally performed on a single group at a single level, often this would be the whole matrix. Optionally, when the operation is performed, as well as ordering the items so the closely related elements are adjacent, it can also form groups above the items, one for each related set. This will remove any existing group structure above the items.

Again, the most common use would be to decompose a system into subsystems with high cohesion and low coupling.

3.3.4 Transitive closure

Mathematically, the transitive closure of a node in a graph is that node and all nodes which can be reached from it by following edges, hence its alternative name, reachability analysis. For our purposes, we can use a given relationship or relationships as valid edges, and can choose whether those relationships are to be viewed as directed or not. Further, the transitive closure of a matrix will be the set of the transitive closures of its axis items, using only those relationships of chosen type(s) present in the matrix.

Like affinity analysis, transitive closure can form new groups, and is normally performed on a single group at a single level. Transitive closure is useful for finding what parts of a design are connected to a given part, and hence may be affected if that part changes. It can also be used to check that information can flow between two parts of a system, or for various analyses in Petri net structures.

4 User interface

This section presents a more detailed description of a suitable user interface for the general matrix editor described above. Thus it is important to remember that we are dealing with one tool among the many in MetaEdit+, and consistency of approach across all tools is desirable. For instance, the visual cues to show that an item is selected or can explode should be similar to those used in the Draw Window, as far as the different representational paradigms allow.

A user interface can be thought of as consisting of two main components: a visual interface, which is the view of the data presented to the user, and a functional interface, which is the way the user accesses the functionality of the operations on the data. In this section, we will look in turn at each of the areas of the Matrix Editor, the tool window itself, the axes, the elements, and the overlay. For each area, the user interface will be discussed, starting with the purely visual components such as layout, then the components which are more functional such as menus and dialogs.

4.1 Matrix Editor tool window

4.1.1 Layout

The basic display format for the matrix editor was chosen to be similar to that of spreadsheets, thus using conventions with which most users would already be familiar. The main difference is that the user-created axes (as opposed to the fixed row and column names) in a spreadsheet are simply normal cells with textual content, whereas for a matrix they are an entirely different structure, being hierarchical rather than grid-based.

In Figure 4, a mock-up screen shot shows a matrix illustrating the types of data access different Processes have on data Stores in a Data Flow Diagram. The Processes are in the vertical axis on the left, with labels from their descriptive text field, and the symbol shown above the axis. The Stores are shown with their numbers as labels, and grouped according to the storage medium, Files, Buffers or Paper, with the group names shown above the contained items. The Stores numbered 7 and 8 have been hidden (indicated by a dotted line), and 12 and 13 have yet to be assigned to any group. The element containing the relationships between parse (a Process) and 4 (a Buffer Store) has been selected (shown by inverse video and handles at the corners), and the user is prompted in the bottom context-sensitive help bar to click again to allow selection of either the read or update relationships (shown by r and u in the element). The relationships between print and 6 have been locked by another user, (shown by the grey background) so this user cannot alter them. The read relationship between check and 6 can be exploded to another diagram (shown by the rectangular outline). The scroll bars, title bar and menu bar reflect the Windows standards.

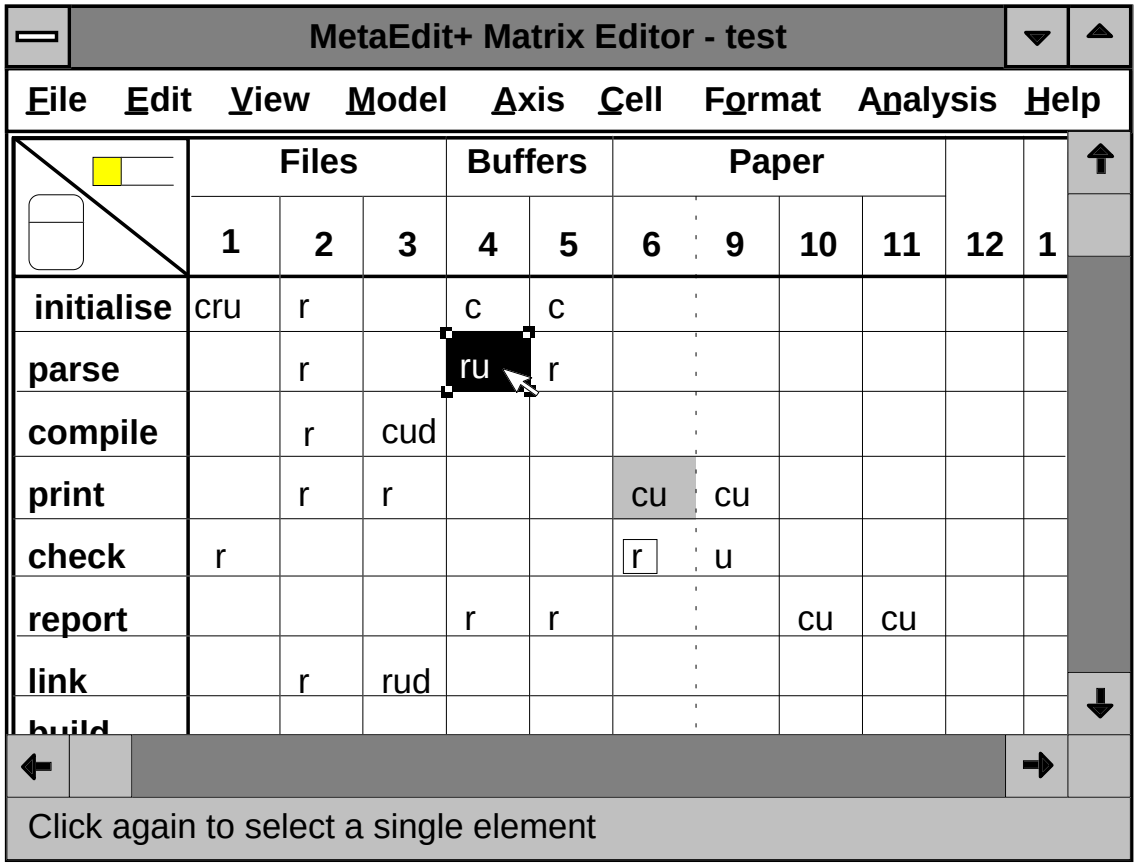


FIGURE 4 Matrix Editor Window

4.1.2 Multi-dimensional aspects

Apart from the normal two dimensional square matrix, the Matrix Editor supports matrices formed from an n-dimensional set of data, and thus having n axes. The number of dimensions will depend on the cardinality of relationships, i.e. the number of roles each relationship has. Whilst most ISD methods use only binary relationships, some, such as NIAM[31], allow multi-part relationships.

Mathematically, and hence computationally, the extension from 2 to n dimensions presents no real difficulties; the main problem is to present the information to the user in a manageable way. Three-dimensional space rendered into two dimensions is a familiar concept, and the additional information displayed outweighs the problems of interpretation in most cases. However, manipulating a three-dimensional space rendered into two dimensions is far more difficult, as are any actions involving higher dimensions. Hence there seems little point in providing some kind of perspective (isometric or otherwise) display.

The other possibility for working in two dimensions with multi-dimensional data is to use some projection of the other dimensions into the two dimensional plane matrix. An example would be in the three dimensional case, where relationships are ternary (i.e. each relationship has three objects participating in it), and we choose objects for the three axes and ask for a

display of relationships between those objects. In this case, a normal two-dimensional matrix is extracted by showing only those ternary relationships whose third member is selected on the third, undisplayed, axis, condensing the information from all such relationships into the appropriate individual elements of the resultant matrix. This allows precise tailoring of the information to be displayed, as any number and distribution of objects along the third axis can be selected.

4.1.3 Menus

The structure of the menus is intended to reflect the user's way of working, and as such can be customised by definition in the metamodel for that method, and later altered by each individual user. The structure shown in Table 1 is the default, containing all possible commands. As all operations involving selection of objects and actions can be performed by either keyboard or mouse, accelerator keys can be defined, and each menu item has a one letter abbreviation (not shown in the table). Similarly, some of the menu items open a dialog, and these are followed by an ellipsis mark in the application.

TABLE 1 Menus and Menu Items

File	**Edit**	**View**	**Model**	**Axis**	**Cell**	**Format**	**Analysis**	**Help**
New	Undo	Hide	Properties	Before	Choose	Text	Sort	Matrix
Rename	Cut	Show	Explode	After	Display	Lines	Diagonalise	Method
Print	Copy	Show All	Objects	Parent	Function	Labels	Affinity	About
Import	Paste	Types	Relations	Child	Format	Cell	Analysis	
Export	Delete	Zoom	Roles	Format		Element	Transitive	
	Find	Normal					Closure	
	Select	Diagonal						
	All	Triangular						
	Extend	Choose						
	selection	Axes						
		Matrix						
		Info						

4.2 Axes

4.2.1 Layout

The most important feature of the axis layout, which distinguishes it from the otherwise similar format used in traditional spreadsheets, is the hierarchy. This is displayed as a tree structure of groups, with the area of each group extending over that of all its members (see Figure 5). The dimensions of each row or column and hierarchical level are user-definable, with a setting for automatic sizing, and another for a default set of values. Similarly, the font, type style and point size can be selected, or assigned a default value according to the level in the hierarchy.

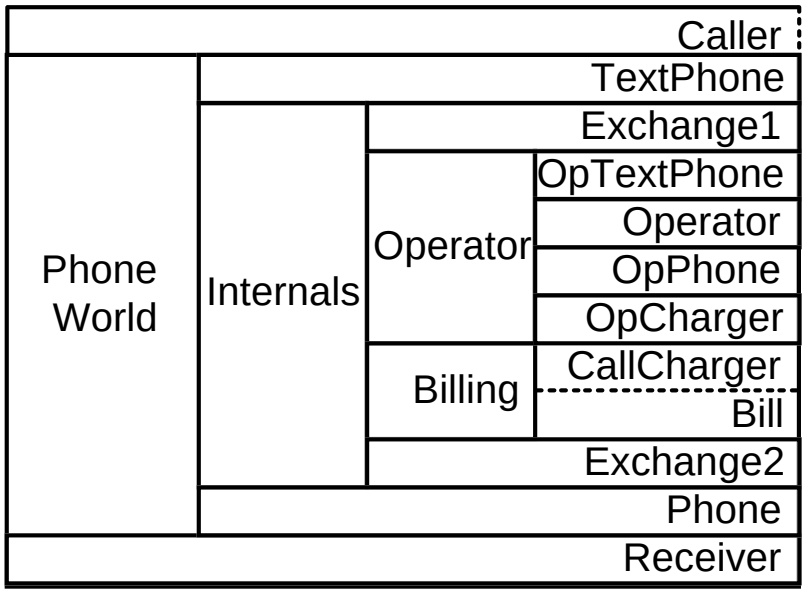


FIGURE 5 Hierarchical Axis Structure

If an item or group is hidden, then the dividing line where it would appear is dotted: for instance, an item or set of items has been hidden between CallCharger and Bill in the example. If there exists a hidden extension of the hierarchy above or below a given item, then that line too is dotted: for instance, there is a further decomposition of Caller that is hidden in the example.

4.2.2 Dialogs

Besides popup confirmation requests, there are only two dialogs for axis operations. The first concerns showing and hiding individual items, groups, types of items, or levels in the hierarchy. In the second dialog, the user can select formatting for all the various elements of the axis display: lines, fonts, box sizes, text direction (only for the vertical axis), etc.

4.3 Elements

The intersection of a given row and column pair forms a box, which we will call a cell. Within each cell there are potentially several relationships or roles, which we will refer to as elements: the pair of objects that define the contents of the cell may have more than one relationship between them.

If the axes are identical, then there will be two entries for a given pair of objects, denoted by the two ordered pairs (A,B) and (B,A): in this case, the cell at (A,B) will contain the roles of B in any relationships between A and B, or the relationships flowing from A to B. Where the axes are different, roles and relationships of any direction may be contained in the cell.

4.3.1 Layout

As there are possibly several elements in the cell, the area of the cell is divided into equal parts, and each element is assigned to one of these sub-cells. The metamodel or user can set an upper limit on the number of sub-cells that can be

displayed in a cell; over this limit, the last sub-cell will show an ellipsis (…) symbol.

The information for each element can be a property or the type of any of the following: the row item, the column item, the relationship, or either of its two roles (in the case where the axes are identical, only a single role is available).

The display of an element can be selected to be a symbol, data, a character, or an 'x' (see Table 2). The symbols are defined in the metamodel, and should give some indication of the type of the relationship or role. Data is some text from a property or type, and can be limited to a specific number of characters. The single character display is dependent on the relationship or role type: the particular character for each type is defined in the metamodel or by the user. Finally, the display in each element can be restricted to just an 'x' to show that it is non-empty.

TABLE 2 Element Display Options

Symbol	Data	Character	X
⇨	voice	I	X
⇔	19.2	I,O	X
	text	O	X

In addition, each cell can show further information: whether it is locked by another user (indicated by painting the background grey), whether it is selected (indicated by marking the corners with small squares), and whether it is the current cell (indicated by inverting the colours). If an element of the cell can be exploded, this is indicated by a border around its sub-cell.

4.3.2 Dialogs

Some operations in the body of the matrix, for instance copy or move, operate on all the elements of each selected cell; others, such as displaying properties, only operate on one of the elements of the cell. As the basic selection unit is the cell, the user must have a way of refining that selection still further, to a single element. This is provided by means of a dialog which displays a list of all the elements, with that element selected which was closest to the point in the cell where the user clicked. In the case where there is only one element in the cell, the dialog is not required, and the choice of element or cell is determined by the context of the operation selected.

5 Conclusions

The need for a matrix editor in a metaCASE environment is shown by the number of widely-used methods that use matrices, the narrowness of existing method-specific matrix tools, and the insufficiency of e.g. spreadsheets as a substitute. For instance, Andersen Consulting's Plan/1[9,10] supports matrices,

but only for their Method/1, and the ability to form a hierarchy on an axis is limited to a single, obligatory level: each item initially belongs to a default null group. Similarly Bidgood and Jelley[4] found it painful to work with a spreadsheet as makeshift matrix editor: apart from the obvious problems caused by the spreadsheet not having been designed for that kind of usage, the data in the spreadsheet could not easily be shared with that in a CASE repository.

This paper has attempted to examine and explain some of the properties that make matrices a useful representation in certain areas of information systems design, to show that a metaCASE matrix editor is feasible, to present a set of requirements for such a tool, and to suggest a user interface for it as one tool among many in an actual application, MetaEdit+. The matrix editor described is:

- representation independent, allowing the underlying conceptual data to be shared with other tools and displayed in their paradigm — an advantage over earlier tools such as PSL/PSA[5] in which matrices were read-only output formats;
- based on the GOPRR meta-metamodel[14], rather than a single metamodel, and so not tied to any one method — an advantage over fixed method matrix editors such as in Plan/1;
- customisable both from the metamodel and by the user;
- structurally rather than computationally oriented, concerned more with the relation of items to each other than producing a single value from an item or set of items;
- similar in its visual and functional user interface to both spreadsheets (a similar format familiar to users) and other tools in MetaEdit+ (to sharpen the learning curve for the whole application);
- designed to answer previously recognised needs from real practical work.

The matrix editor described in this paper is currently being implemented in Visual Works Smalltalk within the MetaPHOR project. The prototype application will be tested within the University of Jyväskylä, and the final product is intended to be tested in real commercial environments. Future research with the tool will include comparison of the relative efficiencies of matrix and diagram formats for various operations, design of new matrix operations and methods that take advantage of representation independence, and testing of algorithms to transform or add new information from a matrix to a diagram.

As the concept of matrix editors in metaCASE environments is so far in its infancy, it is to be hoped that these suggestions will serve as a starting point from which the ideas can be further refined and developed.

Acknowledgements

This paper was produced as part of the MetaPHOR research project, funded by the Academy of Finland and University of Jyväskylä. I would like to express my thanks to the other members of the project, particularly Juha-Pekka Tolvanen, for their help and encouragement in the research and development of the matrix editor, and to Prof. Kalle Lyytinen and the reviewers for their helpful comments on this paper.

References

1 **Smolander, Kari, Tahvanainen, Veli-Pekka and Lyytinen, Kalle** 'How to combine tools and methods in practice — a field study' in B Steinholz, A Sölvberg, L Bergman (Eds), Advanced Information Systems Engineering, proceedings of the Second Nordic Conference CAiSE '90, Stockholm, Sweden, May 8–10, 1990, Springer-Verlag, Berlin (1990) pp 195–214

2 **Le Quesne, P N** Individual and Organisational Factors in the Design of Integrated Project Support Environments, Ph.D. Thesis, London Business School (1990) p 260

3 **Bubenko, J A** Selecting a Strategy for Computer-Aided Software Engineering (CASE), SYSLAB University of Stockholm, Stockholm, Sweden (1988)

4 **Bidgood, T and Jelley, B** 'Modelling Corporate Information Needs: fresh approaches to the information architecture' Journal of Strategic Information Systems Vol 1 No 1 (Dec. 1991) pp 38–42

5 **Meta Systems Ltd.** PSL/PSA 6.0 User Manuals, Ann Arbor, MI (1987)

6 **Teichroew, Daniel and Hershey, Ernest A III** 'PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems' IEEE Transactions on Software Engineering (Jan. 1977)

7 **Cadre Technologies Inc.** Paradigm Plus, Cadre Edition User Manuals, Providence RI (1993)

8 **Steward, Donald V** Software Engineering with Systems Analysis and Design, Brooks/Cole (1987)

9 **Arthur Andersen Consulting** Foundation-Method/1: Information Planning, Version 8.0, Chicago (1987)

10 **Arthur Andersen Consulting** Foundation-Method/1: Documentation, Version 8.0, Chicago (1987)

11 **Kelly, S, Lyytinen, K, Marttiin, P, Oinas-Kukkonen, H, Rossi, M, and Tahvanainen, V-P** MetaPHOR: Second intermediate report, Universities of Jyväskylä and Oulu, Finland (internal report) (1994)

12 **Welke, R J** 'The CASE Repository: More than another database application' in W W Cotterman and J A Senn (Eds) Challenges and Strategies for Research in Systems Development, Wiley, Chichester UK (1992)

13 **Smolander, K** OPRR: A Model for Modelling Systems Development Methods, in Licentiate Thesis WP-20, University of Jyväskylä, Finland (1991)

14 **Smolander, K** GOPRR: A proposal for a meta level model (unpublished working paper), Jyväskylä (1993)

15 **IBM Corporation** Business Systems Planning — Information Systems Planning Guide, Publication #GE20-0527-4, (1975)

16 **Martin, J** Strategic Information Planning Methodologies, Prentice-Hall, Englewood Cliffs, NJ (1989)

17 **Kerner, D V** 'Business Information Characterization Study' Data Base (Spring 1979) pp 10–17

18 **Vepsäläinen, A P J** 'A Relational View of Activities for Systems Analysis and Design' Decision Support Systems 4, North-Holland (1988) pp 209–224

19 **Lano, R J** The N-Squared Chart, a TRW internal report, Redondo Beach, CA (1977)

20 **Langefors, B and Sundgren, B** Information Systems Architecture, Petrocelli / Charter, New York (1976)

21 **MacDonald, I G and Palmer, I R** 'System Development in a Shared Data Environment: the D2S2 Methodology' in TW Olle, HG Sol, AA Verrijn-Stuart (Eds), Information Systems Design Methodologies: a comparative review, North-Holland (1982)

22 **Peterson, J L** 'Petri Nets' ACM Comp. Surveys Vol 9 No 4 (1977) pp 223–252

23 **Carlson, W M** 'Business Information Analysis And Integration Technique (BIAIT): a new horizon' Data Base (Spring 1979) pp 10–17

24 **Lundeberg, M** 'The ISAC Approach to Specification of Information Systems', in TW Olle, HG Sol, AA Verrijn-Stuart (Eds), Information Systems Design Methodologies: a comparative review, North-Holland (1982)

25 **Venable, J R and Truex, D P, III** 'An Approach for Tool Integration in a CASE Environment' in Proceedings of CASE Studies 1988, Meta Systems Ltd., Ann Arbor, MI (1988) Ref. C8812

26 **Sayani, Hasan H** 'Using Graphic Front-End Tools for PSL/PSA' in Proceedings of CASE Studies 1988, Meta Systems Ltd., Ann Arbor, MI (1988) Ref. C8816

27 **Di Battista, G, Eades, P and Tamassia, R** Algorithms for Drawing Graphs: an Annotated Bibliography (draft), Brown Univ., RI (March 1993)

28 **Davidson, R and Harel, D** 'Drawing Graphs Nicely Using Simulated Annealing' Technical Report, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot (1989)

29 **Electronic Industries Association** CDIF CASE Data Interchange Format Interim Standards, EIA Engineering Dept., Washington (1991)

30 **Booch, G** Object-Oriented Design With Applications, Benjamin / Cummings, Redwood City, CA (1991)

31 **Nijssen, G M and Halpin, T A** Conceptual Schema and Relational Database Design: A fact oriented approach, Prentice Hall, Englewood Cliffs, NJ (1989)

Postscript

The actual implementation of the matrix editor followed this article closely, with the following exceptions:

- The hierarchical axis was not implemented, because it would have implied having a higher level graph open in the same editor as the main graph: this gives rise to problems when considering locking in the multi-user situation.
- Following from this, viewing of decompositions does not occur within the same window, but opens a new window.
- Multi-dimensional possibilities were omitted: there is little use for them, and they are overly complex in practice. The rationale for multi-dimensional matrices stemmed mainly from the need to represent ternary and higher order relationships: these are now represented simply by showing the relationship in all cells whose axis items are a pair of the objects involved in the relationship.
- Extra possibilities for manipulating the display of cell contents were added: cells can display relationships, the roles related to the horizontal axis item, or the roles related to the vertical axis items. The initial choice is automatic, based on whether roles or relationships in the metamodel store more information. It is also possible to treat relationships as directed, thus a relationship between A and B will only be shown once (at cell A-B, but not at B-A as would normally be the case).
- A list-based tool was added for quick manipulation of large numbers of items on the axes.
- The algorithm for calculating the cell contents from the relationships in the graph has undergone significant evolution over the course of its life. The two main areas of difficulty are n-ary relationships and the need for high performance: in a 100 by 100 matrix, there are 10,000 cells to calculate. The current algorithm is sufficiently generic that it is also used on the metamodel level, to identify which are the legal bindings of relationships and roles that can be created between a given set of objects.

The matrix editor has been used as a metamodelling tool, as suggested here; the results are discussed in the article ‘Evaluating Method Engineer Performance: An Error Classification and Preliminary Empirical Study’.