CHAPTER 3

METAEDIT+:
A FULLY CONFIGURABLE MULTI-USER AND MULTI-TOOL CASE AND CAME ENVIRONMENT

This paper was published in *Proceedings of the 8th International Conference on Advanced Information Systems, CAiSE'96*, Springer, pp. 1–21, 1996.

METAEDIT+
A FULLY CONFIGURABLE MULTI-USER AND MULTI-TOOL CASE AND CAME ENVIRONMENT

Steven Kelly, Kalle Lyytinen
Matti Rossi
Department of Computer Science and Information Systems
University of Jyväskylä
PL 35
FIN-40351 Jyväskylä
Finland
email: kelly@cs.jyu.fi
fax: +358 14 603011

Abstract: Computer Aided Software Engineering (CASE) environments have spread at a lower pace than expected. One reason for this is the immaturity of existing environments in supporting development in-the-large and by-many and their inability to address the varying needs of the software developers. In this paper we report on the development of a next generation CASE environment called MetaEdit+. The environment seeks to overcome all the above deficiencies, but in particular pays attention to catering for the varying needs of the software developers. MetaEdit+ is a multi-method, multi-tool platform for both CASE and Computer Aided Method Engineering (CAME). As a CASE tool it establishes a versatile and powerful multi-tool environment which enables flexible creation, maintenance, manipulation, retrieval and representation of design information among multiple developers. As a CAME environment it offers an easy-to-use yet powerful environment for method specification, integration, management and re-use. The paper explains the motivation for developing MetaEdit+, its design goals and philosophy and discusses the functionality of the CAME tools.

Keywords: CASE, CAME, method, software engineering environments, repository, metamodeling, conceptual modeling, object oriented modeling, tool interoperability, tool integration

1 Introduction

CASE (Computer Aided Systems Engineering) environments have been one of the major technological innovations in systems development during the last decade. Many have claimed that CASE technology will solve the information systems (IS) development problems (Cha86, McC89) that have plagued the community for so long. These include, among others, mediocre productivity (e.g. unrealistic time schedules and cost overruns), and insufficient quality (e.g. low product validity and lack of verifiability) (Bro75, Cha86, Ost87). CASE technologies are expected to provide task related support for software developers in analyzing, designing and implementing a set of information systems (IS) or their components according to a *method.* A method can be defined as a language (vocabulary and grammatical composition rules) which

can be used to represent features of the information system to a number of actors (including technical actors such as specific abstract machines like a Smalltalk machine) and a set of rules which define by whom, when, and how such representations are derived and/or used.

The origins of CASE date back to the mid 70's when such well-known software tools as PSL/PSA (Tei77) and SREM (Alf77) were launched. Despite this early start, the breakthrough of these technologies has only occurred during the 90's. One reason for this is the declining cost of computing technologies and its increasing functionality — especially graphical user interfaces. Another is the increased need for disciplining the art of software development and maintenance through standardized process and product models. Finally there is a continuing need to improve the quality and productivity of software production through investments in capital intensive technologies.

In spite of these developments the rate of adopting CASE has been laggard, and the success of adoptions doubtful (Wij90, Aae91). One reason for this is software organizations' lack of the necessary maturity to adopt highly sophisticated technologies such as CASE. Another is the cost of adopting, using and maintaining the technological infrastructure and associated know-how. The third reason is the inadequate technological sophistication of CASE. Most tools in use are stand alone tools that support creation and maintenance of graphical models and can generate code to limited problem domains. Accordingly technologies have not matured for software development in the large and by many. The major deficiencies are thus: insufficient support in integrating methods, inadequate support for alternative representation paradigms, lack of mechanisms to cater for multiple users, rigid method and process support, and focus on task automation (Hen90).

In this paper we report on the development and use experiences of a prototype next generation CASE tool, MetaEdit+. The environment seeks to overcome all the above deficiencies, but pays particular attention to the requirements concerning flexibility, method integration and representational richness. In line with this MetaEdit+ is a multi-method, multi-user, multi-tool platform for both computer aided software engineering (CASE) and computer aided method engineering (CAME). As a CASE tool it establishes a versatile environment for flexible creation, maintenance, manipulation, retrieval and representation of design objects (information) structured and created according to a method. As a CAME tool it provides a flexible and easy-to-use environment for method specification, management, integration and re-use. This paper will explain the motivations for developing MetaEdit+, its design goals and philosophy, its design architecture, its current tool set, and its future development.

2 Related research

Weaknesses in current CASE tool support can be divided into the following aspects:

1. lack of mechanisms for integrating sets of methods while maintaining consistency between various models (Kel96, Mar95, Kel94a)
2. lack of support for multiple users to create, modify and delete sets of partly overlapping model instances,
3. inadequate catering for and mapping between multiple representational requirements such as diagrams, matrix, table and hypertext representations (Kel96, Mar95),
4. lack of flexibility and evolvability in method support ranging from syntactic variation in methods to crafting totally new method components (Lyy89), and
5. insufficient catering for different information-related needs of a diverse set of stakeholders (Mar95).

2.1 Lack of method integration mechanisms

Several mechanisms are available for method integration or interaction. At the most rudimentary level these deal with mechanisms that enable translations from one representation format to another. Attempts to develop such CASE "EDI" solutions abound, e.g. CDIF (CDI91). Their weakness is that they do not support any inter-model consistency checking, semantic validation and tool interoperability. Accordingly, they can only be used in static model transfer from one environment to another. A more advanced approach has been to develop generic and universal method specification schemata. This "super-schema" would provide a common and universal semantic model onto which all methods used in the environments could be mapped. This can be done directly as in the A/D Cycle information model (Mer90), or through method reference models (Hey92) where the mapping takes place through a reference model. An early solution of this kind was the mapping of system development methods into generic modeling constructs of PSL/PSA (Tei77). The limitation of this approach is its closed nature of method integration which cannot tolerate any flexibility in the mappings. Moreover, it cannot cater for future evolution in the method arena. Finally, it can only support a limited number of method integration solutions which deal solely with object sharing and associated consistency checks.

2.2 Insufficient multi-user support

A large body of literature exists on concurrency control and alternative strategies to deal with multiple user operations in software engineering

repositories (for a review see Bro91). A number of strategies have emerged recently for achieving varying levels of optimistic concurrency control (Kat84). Despite these advances it is still not known which granularity levels are appropriate for effective concurrency, what are suitable transaction notions, and how much locking and what types of locks are needed. Moreover, it is not clear how much transaction management should be left to users' awareness of others' operations. In this respect, most commercial CASE environments provide solutions that are too crude or inefficient, while advanced mechanisms suggested by researchers can be computationally too demanding (e.g. use of work spaces and merge strategies) or cannot be adapted to the existing CASE architectures. Moreover, a big unsettled issue is how well semantics-driven and dedicated locking strategies operate in such environments and whether we should cater for differences between conceptual and representational objects, or between the different tools that operate on the design data (Kel94a).

2.3 Insufficient support for multiple representation paradigms

Whilst today's methods contain various representation paradigms — graphical diagrams, matrices, tables, etc. — most existing CASE tools operate on only one: graphical diagrams. If other representation forms are needed they are generated by some user triggered operations such as generating a report. Because of this, CASE tools do not offer the *representation independence* that could make them fully adaptable to differing representation demands. Thus most CASE tools offer only limited syntactic and graphical modifiability in supported methods. Another weakness is the lack of hypertext support for semi-structured and non-structured linking of design objects in different representation formats or model parts. Either the available functionality provides hypertext features *as* the CASE environment (Cyb92), or the support functionality is limited to some model areas (Poh94) or to user interface and user support (Oin93).

2.4 Lack of method modifiability and evolution

The importance of CAME in CASE has been noticed in several studies (Kum92, Che88, Bri90, Wij91, Hey93, Ste93). To this end CASE shells — metaCASE tools, or fully customizable CASE environments — have been developed. Such environments are expected to overcome the inflexibility of method support. According to Bubenko (Bub88) "a CASE shell includes mechanisms to define a CASE tool for an arbitrary technique or a chain of techniques". Yet, metaCASE technology has not yet matured sufficiently to provide adequate method modifiability though the number of CASE products leveraging method modification facilities is increasing. Commercial products offering such features include *Customizer™* (Ind87), *VSF* (Poc91), *MetaEdit™* (Smo91a) and *Paradigm+* (Pro94). Research versions of CASE shells include *RAMATIC* (Ber89), *ConceptBase* (Hah91) and *MetaView* (Sor88). Integration of CASE shells and their CASE environments comes in various kinds. A CASE shell can be a separate

tool that produces a methodology specification which the CASE environment uses (e.g. *Customizer*), or it can be an integral part of the CASE environment (e.g. *RAMATIC*). *MetaEdit* ™ was one of the first that offered CAME and CASE functionality within the same tool. In MetaEdit methods are specified graphically (Smo91b) and these specifications are converted into a textual form, before compiling and loading the complete method specification into a CASE environment. All these have been steps in the right direction. However, environments that can offer powerful and easy to use modification facilities, method component libraries, method re-use and run-time adaptability are still largely non-existent.

2.5 Lack of information retrieval and computational facilities

One problem in current CASE tools is their limited information retrieval and reporting capability. Some general and computationally powerful solutions exist in environments that apply a logic programming paradigm (such as ConceptBASE (Hah91)). Though sufficient in their expressive power and generality the use of such query functions is limited by their computational complexity and insufficient user-friendliness. This is due to the lack of data base schema representations and user friendly query formulation. Another problem is that all existing query systems center around retrieving and representing textual information while most of the design information is input and viewed in a graphical format. Finally, few environments provide a means to browse through the repository via hypertext links or various browsing mechanisms.

2.6 Summary

The record of CASE research in each area demonstrates that most concerns have been addressed during the last decade and considerable progress has been made in rendering CASE environments useful. Yet, what seems to be lacking is a comprehensive approach that seeks to tackle most, if not all of, these weaknesses *simultaneously.* Though this may require some compromises and difficult trade-offs in achieving all these goals (like improving multi-user facilities and method flexibility) our contention is that the real impact of future CASE — in the large and by many — will depend on our capability to offer more comprehensive solutions that address most of these concerns within the same environment. Unless such environments emerge the adoption of CASE will in all likelihood continue to be a frustrating experience.

3 The MetaEdit+ environment

As a meta-CASE environment MetaEdit+ seeks to address most of the above concerns (2.1–2.5) in a comprehensive manner by offering an environment which is:

- **multi-user**, i.e. several users can operate concurrently on the repository (2.2),
- **multi-tool**, i.e. each user can operate several tools simultaneously where each tool provides a different view to the same object (2.3, 2.5),
- **multi-method**, i.e. the environment offers several mechanisms for method integration and consistency checking (2.1),
- **multi-form**, i.e. the environment provides several representation formats for the same design object (2.3), and
- **multi-level**, i.e. the environment is a true metaCASE environment in that both an IS and its design methods can be engineered within the same environment (2.4).

The environment seeks to improve the usability (by multiple users, forms, methods and tools), flexibility (by offering a multi-tool, multi-method approach), and open nature of CASE (i.e. by enabling evolution and plugging of new tools through well defined service protocols). The design goal of the environment has been to base its architecture in principles of *conceptual modeling*, *layered data base architectures*, and *object orientation.* In this respect, the approach differs to some extent from other metaCASE approaches which focus more on the representation of methods as first order logical theories (Hah91), or on the graphical behavior of design objects (Ber89). From the viewpoint of conceptual modeling the design of a method specification is akin to the development of a conceptual schema for a software repository, and the design of a software tool resembles a design of an external view to a conceptual schema (ANS75). Hence, the method specification language is at the same time the conceptual modeling language for the repository schema, or forms the meta-metamodel level in the IRDS standard (ISO89). The adoption of full object orientation enables flexible organization and re-use of software components in the environment and a high level of interoperability between tools. This is achieved through both data integration (via shared conceptual schemata) and control integration (via object organization) thus making the environment fairly open.

Our motivation in using conceptual modeling and object orientation in the design of MetaEdit+ has suggested three principles for the design: *data independence, representation independence,* and *level independence. Data independence* is defined in a similar way as in traditional data base theory i.e. tools operate on design information without "knowledge" of its physical organization, or logical access structure. *Representation independence* forms a continuum with data independence and it allows conceptual design objects to exist independently of their alternative representations as text, matrix or

graphical representations. This principle allows flexible addition of new tools, each one only responsible for its own paradigmatically different view on the same underlying data. *Level independence* means that the environment follows a symmetrical approach in its treatment of data and metadata. Accordingly, the specifications of methods and their behaviors can be managed and manipulated in a similar way to any other object in the environment (therefore the name metaCASE). Moreover, the specifications can be concurrently operated through the same or somewhat specialized tools in the environment.

3.1 General architecture

The functional architecture of MetaEdit+ is illustrated in Figure 1. The heart of the environment is the MetaEngine, which handles all operations on the underlying conceptual data through a well-defined service protocol (Smo93a). In other words, the MetaEngine embodies the implementation of the underlying conceptual data model and its operation signature. Accordingly, software tools request services of the MetaEngine in accessing and manipulating repository data. Thereby they avoid the need to duplicate the manipulation code. This design choice allows flexible integration of new tools, each only responsible for its own paradigmatically different view (including operations) on the same underlying repository data. A tool, as the term is used within the MetaEdit+ environment, is a window type with its associated functionality, through which a user can view and possibly alter a design objects in a particular way.

The architecture has similarities with that of the ECMA-PCTE (ECM91) — e.g. common services, separation of components at different levels of integration — but differs from it, most noticeably in the enforcement of no direct communication between components at the same level, or over a common bus between components separated by more than one level: tools communicate only via the MetaEngine.

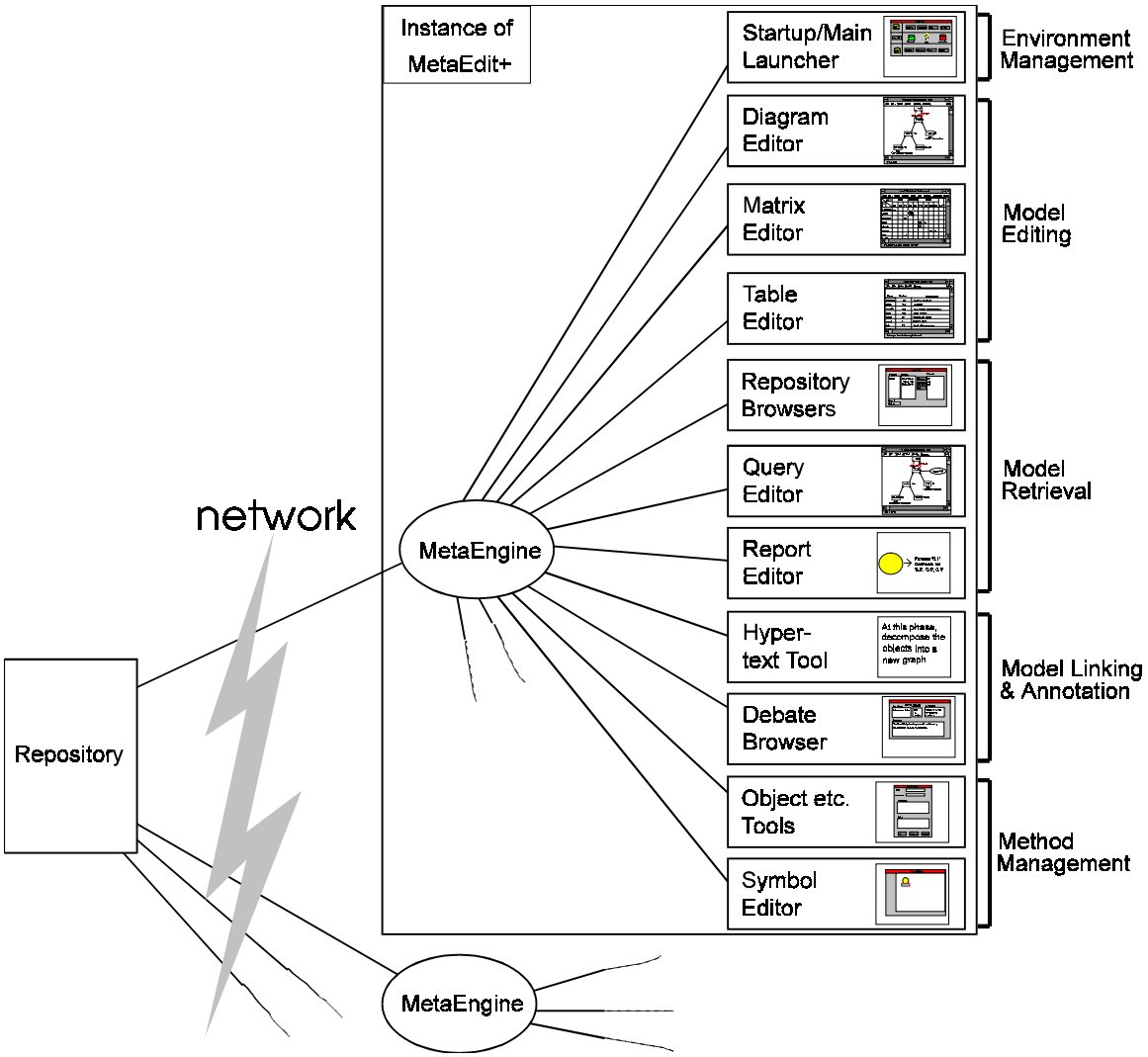

FIGURE 1 MetaEdit+ Architecture

MetaEdit+ can run either as a single-user workstation environment, or simultaneously on many workstation clients connected by a network to a server. Each client has a running instance of MetaEdit+, including all its tools and the MetaEngine. The MetaEngine takes care of all issues involved in communicating with the server. Tools communicate with each other only through the MetaEngine, and thereby through the shared data in the repository. Thus the major integration mechanism applied is data integration.

The server forms the software repository holding all the data contained in models, and also in the metamodel(s), in addition to user and locking information. In particular the MetaEdit+ repository includes: *object specification base* containing all the method specifications represented as GOPRR concepts; *symbol specification base* containing all symbols needed to represent Objects, Relationships and Roles; *tool related information base* containing all information needed to represent conceptual objects in different tools (such as spatial coordinates, or size), *user information base* containing all information related to various users such as their passwords, access rights, or current locks held; r*eport specification base* containing all report and other output specifications.

MetaEdit+ applies pessimistic concurrency control in dealing with user and multi-tool interactions with the repository. We have found locks useful despite some of their disadvantages such as stricter user control, interference with users' work, and poorer overall performance. The gains are greater as locks prevent conflicts from occurring between different copies of the repository data, help users to be warned about possible interference, and prevent gaining access to design objects already used in another's transaction. All these are of utmost importance in software repositories. Transactions are understood as long transactions. Their length is defined by a user triggered *commit* operation (automatically requested by the end of the session). The burden of deciding what to lock and when is removed from user's responsibilities and decided by the system. Another feature of the locking strategy is that MetaEdit+ follows more than one level of granularity in locking repository objects. It distinguishes locking granularities between metamodels, graphs, conceptual objects, and representation data. It can thus achieve the following desired features in locking: locks are acquired only when needed, they are well-placed, and are not too small to overburden the system. During their work users can gain information about locked objects and are thus aware of who has locks on which design objects. Accordingly, they can coordinate their actions through negotiating about how locks are freed and transferred. Although no formal testing has been carried out as yet, initial experiences suggest that with this strategy lock conflicts are surprisingly rare in normal CASE work.

3.2 Tool architecture

In the design of the environment we have classified tools into five distinct families according to their purpose and underlying common functionality. From the viewpoint of conceptual data in the repository each family portrays similar demands in terms of manipulation, locking and retrieval of conceptual design objects, though the different representational paradigms underlying the tools may pose additional demands on retrieval and locking. This has to be dealt with individually in each tool. Each tool family contains one or more tools (Figure 1). The five tool families are the following:

- **Environment management tools**: these tools are used to manage features of the environment, its main components and to launch it.
- **Model editing tools**: these tools are used to create, modify and delete model instances or their parts. In addition, these tools can be used to view the model instances from different representational viewpoints, and/or to derive new information from existing design information.
- **Model retrieval tools**: these tools are used for retrieving design objects and their instances from the repository for reuse and review. These tools can operate on both models and metamodels.

- **Model linking and annotation tools**: these tools are used for linking design objects for traceability and memorization, annotating model instances, finding specific "locations" in the design space, or maintaining conversations about design issues.
- **Method management tools**: these tools are for method specification, management and retrieval.

4 Conceptual data model

Because all method specifications in MetaEdit+ are interpreted as high level conceptual models of method (or methodology) the kernel of the MetaEdit+ functionality and architecture is determined by the underlying conceptual data model called GOPRR. MetaEdit+ uses the GOPRR conceptual data model as a universal and generic meta-metamodel i.e. as a sole language to specify methods. Very little if any method "knowledge" is buried into the code in tools. In addition, GOPRR is primarily intended to model observed, interpreted and recorded development reality *as seen through the methods* (including the world of thought and abstract ideas). In this respect it differs from the ontological IS models (see e.g. Wan93), which attempt to model what actually *is*, rather than just what is perceived and recorded.

4.1 The OPRR model

Basically, GOPRR (Smo93b) forms an evolutionary extension of the OPRR model which has been successfully used in specifying methods for MetaEdit (Wel92, Smo91b). Whereas the original ER model (Che76) provided only sketchy concepts of attribute: features the object can possess; and of role: the part an object plays in a relationship; the OPRR model has defined these notions in full.

The basic OPRR modeling constructs are:

- **O**bjects, which consist of independent and identifiable design objects. These typically appear as shapes in diagrams, and can have properties such as names. Examples of objects are an Entity in an Entity Relationship Diagram or a Process in a Data Flow Diagram.
- **P**roperties are attributes of objects and can only be accessed as parts of objects or relationships. Properties typically appear as textual labels in diagrams, and they can contain single data entries such as a name, text field or number. An example of a property is the number of a Process in a Data Flow Diagram (Gan79).

- **R**elationships are associations between objects, and can also have properties. Relationships typically appear as lines between shapes in diagrams, or verbs in texts. An example of a relationship is a Data Flow in a Data Flow Diagram.
- **R**oles define the ways in which objects participate in specific relationships. In diagrams roles typically appear as the end points of Relationships (e.g. an arrowhead). Roles too can have properties. An example of a role is the specification by directed arrow which end of a data flow relationship is 'to' and which 'from' part of the flow.

In addition OPRR provides constructs for defining cardinality constraints for relationships (i.e. as properties of relationship meta-objects), and means to determine properties which uniquely identify each object instance. The OPRR model is founded on fixed mapping rules between modeling constructs and their graphical behaviors (Ros92).

OPRR is further designed to be applicable to both the instance (model) and the type (metamodel) levels. Thus an instance object, say a Process '3.1' in a Data Flow Diagram model, has an object type of 'Process' on the metamodel level, while a flow relationship instance 'order info' on the model level is an instance of a relationship type 'Data Flow' on the metamodel level.

4.2 Extensions in the GOPRR model

GOPRR extends OPRR as a conceptual meta-metamodel in several ways. First, unlike OPRR the GOPRR model allows multiple representations of the same underlying conceptual object (e.g. graphical, matrix, text), and even different graphical representations of the same object in the same representation paradigm. This is achieved by making available mechanisms that can override the default representation. In this sense GOPRR forms a true conceptual "kernel" on which varied representations of data, including not only graphical diagrams but also hypertext, text and matrices, can be built. This allows GOPRR to support a wide range of methodologies including matrix, table or text oriented ones, and gives users the ability to see and manipulate design information in a variety of representations.

Second, the conceptual modeling constructs offered by OPRR have been extended in the GOPRR in several ways which yields a powerful but yet ease-to-use modeling language. These new Graph, object orientation, method integration and rule constructs are described below.

4.2.1 Concept of Graph

The GOPRR model adds the concept of Graph into the modeling constructs. A graph denotes an aggregate concept which contains a certain set of objects and their relationships (with specific roles). Graphs typically appear as windows on whole diagrams which contain objects and their bindings of roles and relationships; a graph also has its own properties. An example of a graph is a whole Data Flow Diagram (as a whole or just one level of it). In use, the Graph concept is fundamentally a generalized decomposition graph: it can be included

in a parent graph, attached to an object, role or relationship therein. For instance, in Data Flow Diagrams a top level graph may contain a Process '3', which has a decomposition graph called 'Decomposition of 3', containing Processes '3.1', 3.2' etc. Relationships from '1' and '2' to '3' in the top level graph are actually interface relationships, as we can specify that in the lower-level graph they link to e.g. '3.2' and '3.1' respectively. The interface to the object, and hence to the elements in its decomposition graph, can be shown in the child graph to any degree between 'not at all' and 'show copies of external objects'. The interface is maintained distinct from the elements of the decomposition graph itself, allowing reuse of the decomposition graph in different parent graphs. The interface 'specification' remains the same in all decompositions, but the elements attached to the interface at the higher level can be different in different parent graphs, thus allowing reuse of the graph as a white or black box.

The design of Graph is such that many "representational" graphs can be made for one "conceptual" graph. For instance, a matrix and diagram representation can be made of the same conceptual Data Flow Diagram. In this situation changes in conceptual graphs are propagated between different tools and their "representational" graphs according to their usefulness to the user. Currently, objects added in one graph are immediately available to other graphs, but not automatically added. Changes to properties are made instantly (on transaction commit, if different users are working on different graphs), and additions to or changes of relationships or roles are made instantly in the relationship-oriented Matrix Editor, but buffered in the Diagram Editor, so the user can control their layout when they are added.

The addition of the concept of Graph allows GOPRR to represent multiple methods, and multiple models, whilst still maintaining the contents of each as a coherent distinguishable whole. In this way graph enables modeling and representation of recursive structures such as decompositions, or complex objects as often found in development methods. The graph notion has also been specialized into a modeling unit called *Project,* which can contain other Graphs, and sub-projects. A Project type thus helps manage the allowed linkages between methods used in a particular project.

It is noteworthy that all concepts included in GOPRR are designed for reuse: both types and instances of object, relationship, role, property and graph can be reused within other graph or project types (or instances).

4.2.2 Object orientation

Another extension, in line with object orientation is the inclusion of *generalization* and *specialization* constructs into the GOPRR language. This extension helps to organize complex method libraries, enhances reuse, and together with the graph notion enables to model in economical way most method components.

In line with object orientation objects a third extension is *polymorphism* of modeling constructs: objects, relationships, roles and properties are polymorphic in the sense that an object seen in one method as an object can be seen in another method as a relationship, or a property. This enables method

component re-use and provides a powerful and flexible method integration mechanism. In this way the method specifications can include specifications of a set of interconnections between different IS models.

4.2.3 Method integration

In addition to decomposition and polymorphism, GOPRR also adds other powerful method integration constructs. Objects, relationships and roles can be reused in many different graphs: a change to the object via one graph is also visible in the other graphs. Similarly, properties can be shared between several objects, with changes affecting all objects referring to that property. These two constructs allow different degrees of saying that two objects in different places are 'the same': an important factor in representing the same 'real world' fact in two different methods. Explosion works similarly to decomposition, but with freer semantics. For instance, each object may have only one decomposition, wherever it occurs, but can have multiple explosion links for every graph in which it takes part.

4.2.4 Integrity checking rules

Finally GOPRR provides enhanced rules for checking the model integrity. It is possible to attach rules to properties, in addition to the normal type restrictions. For example, in modeling Data Flow Diagrams, a rule has been added to the property 'DFD Number' which constrains the contents of the string property to be a dot separated sequence of numbers, disallowing combinations like 'Fred', '2.', '3..2.1', '.'. It is also possible to add constraints on the collection of properties for a given object, role, relationship, graph or project type. For example, a rule could be added to specify that a 'start date' must come before an 'end date' in an activity modeling diagram. These rules, too, are inherited by descendant types, but may be overridden.

4.3 Example

Although the improvements in GOPRR are best seen with complicated methods, for ease of understanding we take a simple Data Flow Diagram metamodel as our example. One way to model Data Flow Diagrams with GOPRR is to note the similarities between the various object types (i.e. processes, externals and stores), and how they may be connected. For instance, instances of all three object types must have a name and a description, and they can connect via a Flow relationship with a Process. These similarities motivate the creation of a generalized 'DFDObject' type, which is specialized into 'Process', 'External' and 'Store' types. DFDObject itself is marked so that it can never be instantiated: it is purely an abstract type.

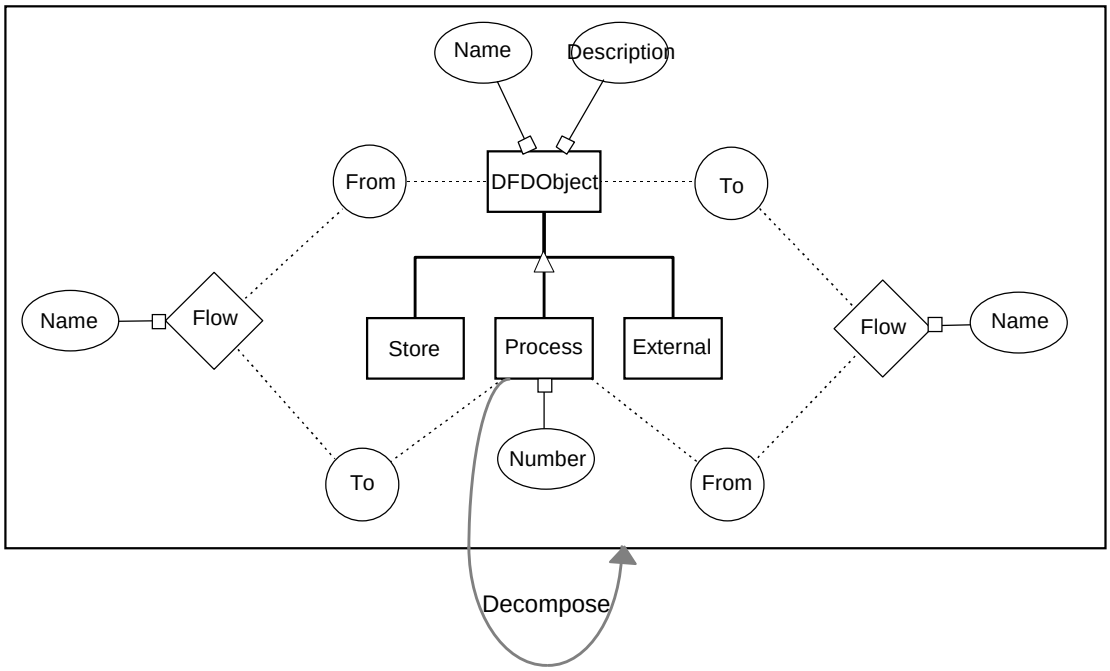

FIGURE 2 A GOPRR metamodel of Data Flow Diagrams

This inheritance hierarchy can be seen in the center of Figure 2, where the rectangles are object types, diamonds are relationship types, circles role types, and ovals property types. DFDObject thus has two properties, Name and Description, and Process inherits these two and adds a third, Number. Objects can be connected by a Flow relationship, with the proviso that one of the objects must always be a Process: on the left, the Process is in the To role, and on the right, in the From role.

The whole figure within the rectangle represents the Graph type of Data Flow Diagrams. The fact that a Process can decompose to a lower-level Data Flow Diagram is represented by the curved gray 'Decompose' relationship between Process and the DFD graph type's rectangle.

5 Method management tools

5.1 Motivation and purpose of the method management tools

In MetaEdit+ the method management tool family has been developed to ease the creation and testing of methods, their management and evaluation support. The primary goal of the tool family is to allow flexibility in method creation and management and ease method construction. Therefore the environment supports alternative ways of method engineering: 1) creation from scratch, where all the parts of the method being defined contain new types, 2) component oriented, where methods are constructed through using prefabricated parts, and 3) reuse oriented, where method engineering seeks to

achieve maximal generality of the repository types, and then by specializing these components derive new methods.

5.2 Design principles of method management tool family

The development of the MetaEdit+ method management tool family has been influenced by earlier method engineering frameworks (Har93, Hey93, Ros95b, Wel92). These frameworks have sought to consider those aspects that are necessary in a completely functional method engineering environment. Functionally such an environment consists of the following parts:

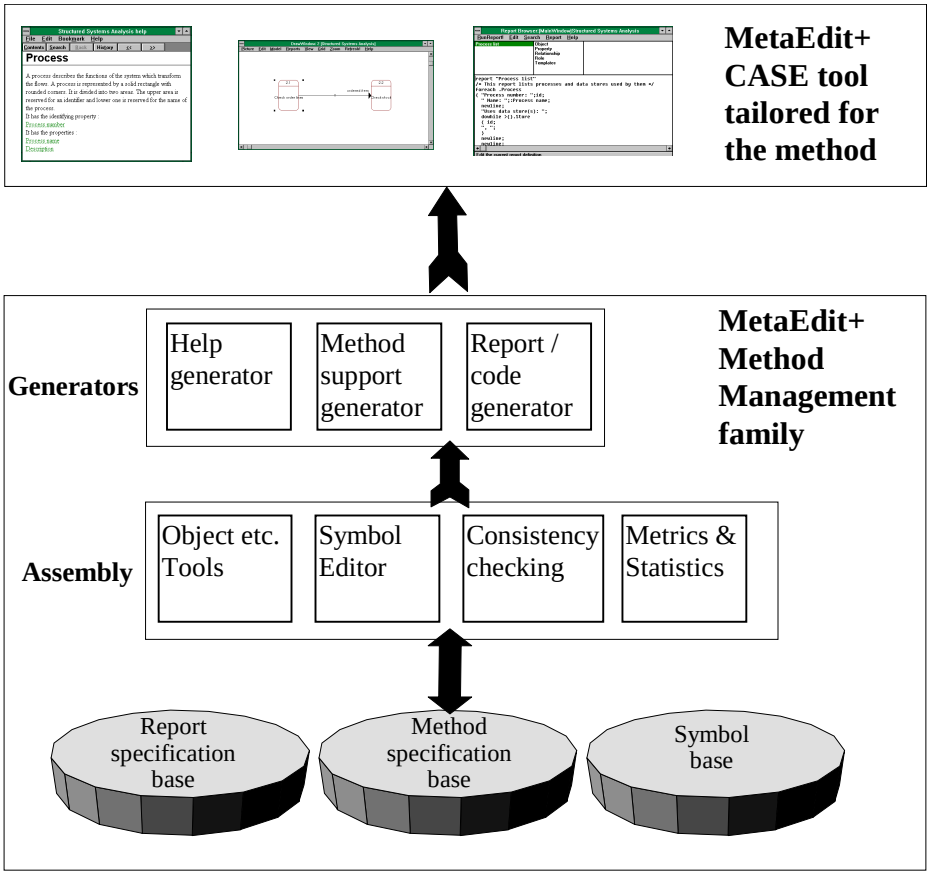

FIGURE 3 Method Management Tools in MetaEdit+

Below each subsystem component will be discussed in more detail.

5.2.1 Method Assembly System

This system part consists of several specialized editors and model retrieval and analysis tools that are needed in method assembly. These tools together allow one to specify a method's objects and relationships and their representations, so that they can be immediately tested within the environment. The most important components are the *metamodel editors* (including object, relationship, role, property and graph editors) by which every method's components and their connections are specified. Due to their different semantics and graphical behaviors Objects, Properties, Relationships and Roles all have their own

specification tools. This helps to define their specialized semantics separately, but in particular allows re-use of existing object, relationship and role specifications. These concepts are then collected into complete method specifications using the Graph tool. This also allows the re-use of existing graph "patterns". Each tool also has a dialog definition subsystem, which allows custom definition of dialogs associated with each object type.

The *Symbol Editor* helps specify symbols that are used to distinguish each object type from other object types. Symbols are defined by a specialized drawing tool and are thereafter connected to the appropriate metamodel type. The Symbol Editor also improves re-use as new symbols can be derived by combining or modifying existing symbol patterns and templates retrieved from the repository. The *Consistency checking system* in MetaEdit+ incorporates several rules that ensure the syntactical completeness and consistency of the resulting method specification. Completeness checking covers checking for missing values and missing links between different method components. Consistency checks verify the internal integrity of the method specification by analyzing that the method specification does not include contradictory specifications. The *Metric & statistics system* of the environment offers a number of reports developed using the report generator tool, that analyze the method specification (Ros94, Ros95a). The metrics reports provide a set of computed values, which can be used to review and analyze the properties of the specifications. Examples of metrics are the number of Object, Relationship and Property types in the method (Tei80), and the average number of Properties or Relationships per Object type.

5.2.2 Environment Generation System

This subsystem features several generators that help to deliver a usable and user-friendly CASE tool by using the information contained in the method specifications. The *Method support environment generation system* compiles the method's object specifications into parts of the metamodel repository as soon as they have been defined. As noted above such specifications define the structure of MetaEdit+'s repository data and the symbols to represent and forms to view the object instances. The *Method help generation system* generates an on-line help component associated with each method. This help can then be accessed through a model editing tool interface from the repository. The generation is based on the defined properties of the metamodel types such as a definition what is an External and how it is used in different situations. *Report and transformation generation system* is used for delivering various reports and conducting checking on the models. These reports can be defined using the generic report generator discussed above.

Parts not available in the current MetaEdit+ method management tool family but recognized in the earlier frameworks are: a selection assistant for selecting the right method or its parts for a specific project (Kum92, Har94), and process description and support (Wij91). These needs are not currently addressed in MetaEdit+, but there are ongoing activities in the project that aim at adding these features.

5.3 An example of a method specification

Here we show how to develop part of the Data Flow Diagram metamodel. The example depicts how the defined components of the DFD are connected together to form the actual method. The tools used to manipulate the GOPRR concepts in the concept specification database are form-based.

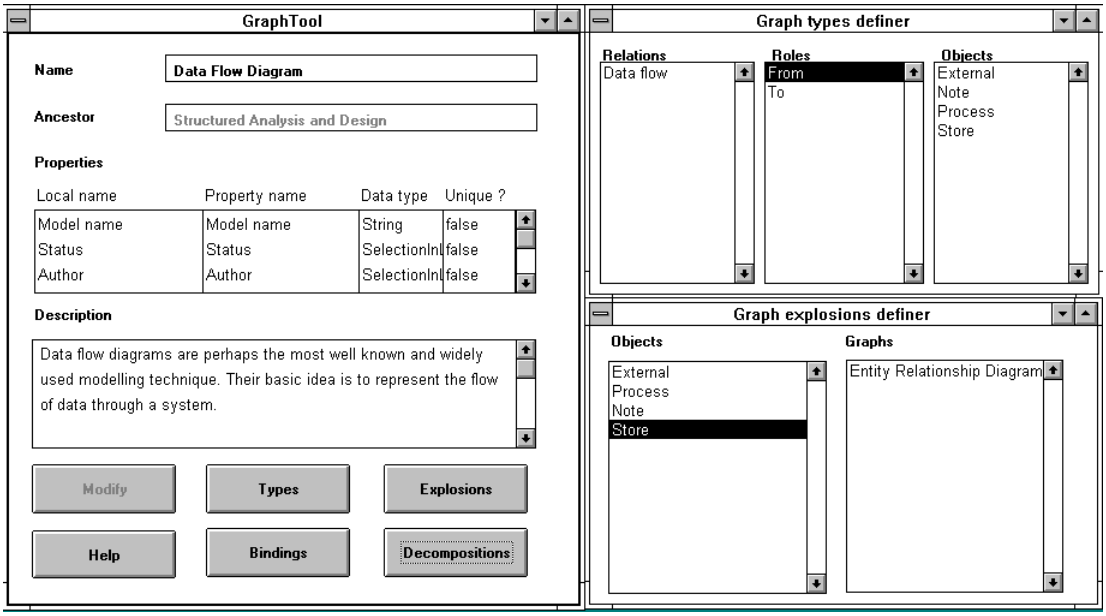

FIGURE 4 A Graph Tool

Figure 4 shows the resulting graph specification of the DFD method. The Graph Tool allows the user to add, remove and edit components of the method (the components, i.e. Objects, Relationships and Roles, are modified with similar tools) and to add and delete method connections. The window on the left shows the definition of the DFD, its properties (i.e. model name and documentation) and related documentation text for method help. The window in the upper right corner of the figure shows the components of the method. The window in the lower right corner shows the possible explosion connections between objects in the DFD and other Graph types: Processes can be exploded into lower level DFD's.

6 Discussion and conclusions

The limited functionality and rigidity of the current information systems development environments continues to pose a considerable challenge to both academia and practice. In this paper our goal has been to demonstrate how the prototype metaCASE environment called MetaEdit+ deals with these concerns. Overall, we have sought to develop MetaEdit+ as a platform for trying out different tools and tool construction principles, and also to try out the use of object oriented architecture in designing and implementing a metaCASE tool.

This is well reflected in its current implementation. MetaEdit+ has been implemented using VisualWorks Smalltalk environment using the ArtBase object repository system and NEDT graphical programming environment, with ENVY as code management system. By doing this, we have been able to re-use about 70% of all code needed to implement the current functionality.

Our goal in developing MetaEdit+ environment has been to develop an environment which:

- Supports high level specification of methods with a powerful yet easy to use method specification language
- Has an open architecture which separates the conceptual specification of the repository and the view (or representation) adopted in different tools and thus conveys a high-level object-oriented API for the tool-repository interactions
- Offers mechanisms for concurrent access of repository data through different tools and users
- Features a comprehensive and well-organized tool set for diverse and complex information handling tasks with some new functionality such as matrices, hypertext tools and the query tool
- Includes flexible, and varying mechanisms for tool integration and both vertical and horizontal method integration support
- Provides symmetrical treatment of IS models and metamodels, and thus enables re-use, metamodel management and utilization within the same environment
- Provides novel support for alternative representational paradigms including matrices, and tables.

We believe that with these features MetaEdit+ addresses many flaws found in current CASE tools. First, through its novel method integration mechanisms it provides innovative ways to organize methods and method families into methodologies, and also to organize methodologies with alternative levels of connectedness and inter-method integrity constraints. Second, through its concurrency management mechanisms MetaEdit+ is able to cater for varying needs and demands for concurrency management for different repository objects. Third, through its open architecture and tool interoperability MetaEdit+ can support the highly diverse representational paradigms and information processing needs which are demanded from software engineering environments. Fourth, through its meta-metamodel MetaEdit+ provides flexibility and evolvability in the method specification and use which is unmatched by any other existing metaCASE tool. Fifth, through the availability of a varied yet uniform (in terms of user accessibility and user interface) tool set the MetaEdit+ environment is able to cater for diverse needs of different system development stakeholders. In this sense MetaEdit+ achieves the design goals of better usability, improved flexibility and a open architecture.

Despite these advances MetaEdit+ is not currently a fully complete environment, suitable for all types of development tasks. First, it does not address the need for multiple distributed repositories which is typical for large scale software development. Second, its concurrency management strategies can be too demanding for large scale software repositories. Third, it does not provide flexible integration mechanisms with other tools (such as electronic publishing or CSCW tools).

Future work in MetaEdit+ will take several directions. First, we want to expand the flexibility and evolvability to cater not only for method representation specifications, but also process and actor models for ISD (Mar94). Second, we will finish the ongoing implementation of the concurrency management system and expand it with the possibility to try out alternative concurrency management strategies which may be applicable in different environments. The third direction is to increase the capabilities to describe integrity constraints within and between method specifications.

On the tool and MetaEngine level the following expansions are currently underway. The applicability of the concept of reusable graphs with 'interface ports', analogous to principles encountered in chip design, will be examined on the model and metamodel levels. The three constructs to represent different levels of 'two things being the same' in a model (multiple representations of the same concept, property sharing, hypertext links) will be examined in the light of current practice in methods. The possibilities of polymorphism based on bindings and metatypes will be examined further in particular as a solution to the problems of metatype polymorphism in existing methods (e.g. objectified associations in NIAM (Nij89), which can be viewed as both objects and relationships). Similarly, the possibilities of the matrix paradigm will be investigated.

To conclude, MetaEdit+ forms a bold attempt to build a versatile platform for implementing flexible design information systems that will form the necessary organizational memory and design resource for knowledge intensive systems and software engineering required in the next millennium. If any improvement has been made in realizing this vision we have achieved our goals.

Acknowledgments

This research was in part funded by the Ministry of Education, University of Jyväskylä, and the Academy of Finland, as the MetaPHOR project (Lyy94). We are also grateful to our colleagues in the MetaPHOR project who have been involved in designing and implementing some parts of the system.

References

Aae91 Aaen, Ivan, Carsten Sørensen, "*A CASE of Great Expectations,*" Scandinavian Journal of Information Systems 3(**1**) (1991) pp.3–23.

Alf77 Alford, M., "*A Requirements Engineering Methodology for Real Time Processing Requirements,*" IEEE Transactions on Software Engineering 3(**1**) (1977) pp.60–69.

ANS75 ANSI, "*Study Group on Data Base Management Systems: Interim Report 75-02-08,*" ACM SIGMOD Newsletter 7(**2**) (1975).

Ber89 Bergsten, Per, Janis Bubenko jr., Roland Dahl, Mats Gustafsson and Lars-Åke Johansson, "*RAMATIC - A CASE Shell for Implementation of Specific CASE Tools,*" Tempora T6.1 Report, first draft, SISU, Gothenburg (1989).

Bri90 Brinkkemper, Sjaak, "*Formalisation of Information Systems Modelling,*" Ph.D. Thesis, Univ. of Nijmegen, Thesis Publishers, Amsterdam (1990).

Bro75 Brooks, F., "*The Mythical Man Month: Essays on Software Engineering,*" Addison-Wesley, Reading, Mass, USA (1975).

Bro91 Brown, Alan W., "*Object-oriented Databases: their applications to software engineering,*" McGraw-Hill, Maidenhead UK (1991).

Bub88 Bubenko, J. A., "*Selecting a Strategy for Computer-Aided Software Engineering (CASE),*" Report 59, SYSLAB, University of Stockholm, Sweden (1988).

CDI91 CDIF, "*CASE Data Interchange Format Interim Standards vol. 1-3,*" Electronic Industries Association Engineering Department (1991).

Cha86 Charette, R., "*Software Engineering Environments, Concepts and Technology,*" McGraw-Hill, New York, USA (1986).

Che76 Chen, P. P., "*The Entity-Relationship Model: Toward a Unified View of Data,*" ACM Transactions on Database Systems 1(**1**) (1976) pp.9–36.

Che88 Chen, Minder, "*The Integration of Organization and Information Systems Modeling: A Metasystem Approach to the Generation of Group Decision Support Systems and Compute-aided Software Engineering,*" PhD Thesis, University of Arizona, Tuscon, USA (1988).

Cyb92 Cybulski, Jacob L., Karl Reed, "*A Hypertext-Based Software Engineering Environment,*" IEEE Software (March 1992) pp.62–68.

ECM91 ECMA, "*Reference Model for Frameworks of Software Engineering Environments,*" Technical Report ECMA TR/55, 2nd Edition (1991).

Gan79 Gane, C., T. Sarson, "*Structured Systems Analysis: Tools and Techniques,*" Prentice Hall, Englewood Cliffs, NJ (1979).

Hah91 Hahn, U., M. Jarke and T. Rose, "*Teamwork Support in a Knowledge-Based Information Systems Environment,*" IEEE Transactions on Software Engineering 17(**5**) (1991) pp.467–481.

Har93 Harmsen, F., S. Brinkkemper, "Computer Aided Method Engineering based on existing Meta-CASE technology," pp. 125-140 in *Proceedings of the Fourth Workshop on The Next Generation of CASE Tools*, Sjaak Brinkkemper, Frank Harmsen (Ed.)No. 93-32, Univ. of Twente, Enschede, the Netherlands (1993).

Har94 Harmsen, Frank, Sjaak Brinkkemper and Han Oei, "Situational Method Engineering for Information System Project Approaches," pp. 169–194 in *Methods and Associated Tools for the Information Systems Life Cycle (A-55)*, A. A. Verrijn-Stuart and T. W. Olle (Ed.), Elsevier Science B.V. (North-Holland) (1994).

Hen90 Henderson, J., J. Cooprider, "*Dimensions of IS Planning and Design Aids: a functional model of CASE technology*," Information Systems Research 1(**3**) (1990) pp.227–254.

Hey92 Heym, M., H. Österle, "A Reference Model of Information Systems Development," pp. 215–240 in *The Impact of Computer Supported Technologies on Information Systems Development*, K. E. Kendall, K. Lyytinen, J. L. DeGross (Ed.), North-Holland, Amsterdam (1992).

Hey93 Heym, M., H. Österle, "*Computer-aided methodology engineering*," Information & Software Technology 35(**6/7**) (1993) pp.345–354.

Ind87 Index Technology Corporation, "*Excelerator Reference Guide*," Index Technology Corporation, Cambridge, USA (1987).

ISO89 ISO, "*Information processing systems: Information Resource Dictionary System (IRDS) Framework*," Draft International Standard ISO/IEC DIS 10027 (1989).

Kat84 Katz, Randy H., "Transaction Management in the Design Environment," in *New Applications of Databases*, Georges Garderin and E Ge (Ed.), Academic Press, London UK (1984).

Kel94a Kelly, Steven, Veli-Pekka Tahvanainen, "Support for Incremental Method Engineering and MetaCASE," in *Proceedings of the 5th Workshop on the Next Generation of CASE Tools*, B. Theodoulidis (Ed.) Memoranda Informatica 94-25, Universiteit Twente, Enschede, the Netherlands (1994).

Kel94b Kelly, S., "*A Matrix Editor for a MetaCASE Environment*," Information and Software Technology 36(**6**) (1994) pp.361–371.

Kel96 Kelly, Steven, Kari Smolander, "*Evolution and Issues in MetaCASE*," Information and Software Technology 38(**4**) (1996).

Kum92 Kumar, Kuldeep, Richard J. Welke, "Methodology Engineering: A Proposal for Situation Specific Methodology Construction," pp. 257–269 in *Challenges and Strategies for Research in Systems Development*, Kottermann W. W. and Senn J. A. (Ed.), John Wiley & Sons, Washington (1992).

Liu95 Liu, H., "A Visual Interface for Querying a CASE Repository," in *Proc. of the Eleventh IEEE Symposium on Visual Languages (VL'95)*, Darmstadt Germany (1995).

Lyy89 Lyytinen, Kalle, Kari Smolander and Veli-Pekka Tahvanainen, "Modelling CASE Environments in Systems Development," in *Proceedings of the first Nordic Conference on Advanced Systems*, SISU, Stockholm (1989).

Lyy94 Lyytinen, K., P. Kerola, J. Kaipala, S. Kelly, J. Lehto, H. Liu, P. Marttiin, H. Oinas-Kukkonen, J. Pirhonen, M. Rossi, K. Smolander, V.-P. Tahvanainen and J.-P. Tolvanen, "*MetaPHOR: Final report*," University of Jyväskylä, Finland (1994).

Mar94 Marttiin, P., "Towards Flexible Process Support with a CASE shell," pp. 14–27 in *Advanced Information Systems Engineering, Proceedings of the Third International Conference CAiSE'94, Utrecht, The Netherlands, June 1994*, G. Wijers, S. Brinkkemper and T. Wasserman (Ed.), Springer-Verlag, Berlin (1994).

Mar95 Marttiin, Pentti, Kalle Lyytinen, Matti Rossi, Veli-Pekka Tahvanainen and Juha-Pekka Tolvanen, "*Modeling requirements for future CASE: issues and implementation considerations,*" Information Resources Management Journal 8(**1**) (1995) pp.15–25.

McC89 McClure, C., "*CASE is Software Automation,*" Prentice Hall, Englewood Cliffs, NJ (1989).

Mer90 Mercurio, V. F., B. F. Meyers, A. M. Nisbet and G. Radin, "*AD/Cycle strategy and architecture,*" IBM Systems Journal 29(**2**) (1990) pp.170-188.

Nij89 Nijssen, G. M., T. A. Halpin, "*Conceptual Schema and Relational Database Design: A fact oriented approach,*" Prentice-Hall, Englewood Cliffs, NJ (1989).

Oin93 Oinas-Kukkonen, H., "*Hypertext Functionality in CASE Environments: Preliminary Findings,*" Conference on Computers and Hypermedia in Engineering Education, Vaasa, Finland (May 24–26 1993).

Ost87 Osterweil, L. J., "Software processes are software too," pp. 180–188 in *Proceedings of the 9th International Conference on Software Engineering* (1987).

Poc91 Pocock, John N., "VSF and its Relationship to Open Systems and Standard Repositories," pp. 53-68 in *Software Development Environments and CASE Technology*, A. Endres, H. Weber (Ed.), No. 509, Springer-Verlag, Berlin (1991).

Poh94 Pohl, K., R. Dömges and M. Jarke, "PRO-ART: PROcess based Approach to Requirements Traceability," in *Poster Outlines: 6th Conference on Advanced Information Systems Engineering, Utrecht, Netherlands, June 1994* (1994).

Pro94 ProtoSoft Inc., "*Paradigm Plus/ Cadre Edition Reference Manual,*" ProtoSoft Inc. (1994).

Ros92 Rossi, M., M. Gustafsson, K. Smolander, L.-Å. Johansson and K. Lyytinen, "Metamodeling editor as a front end tool for a case-shell," pp. 547–567 in *Advanced Information Systems Engineering*, P. Loucopoulos (Ed.), Springer Verlag, Berlin, Germany (1992).

Ros94 Rossi, M., J.-P. Tolvanen, "*Metamodeling approach to method comparison: A survey of a set of ISD methods,*" Working Paper, University of Jyväskylä, Jyväskylä (1994).

Ros95a Rossi, M., S. Brinkkemper, "Metrics in Method Engineering," pp. 200-216 in *Advanced Information Systems Engineering, Proceedings of the 7th International Conference CAiSE'95*, J. Iivari, K. Lyytinen and M. Rossi (Ed.)No. 932, Springer-Verlag, Berlin (1995).

Ros95b Rossi, M., "*The MetaEdit CAME environment,*" Proceedings of the MetaCase 95, University of Sunderland press, Sunderland (1995).

Smo91a Smolander, Kari, Kalle Lyytinen, Veli-Pekka Tahvanainen and Pentti Marttiin, "MetaEdit — A Flexible Graphical Environment for Methodology Modelling," in *Advanced Information Systems Engineering, Proceedings of the Third International Conference CAiSE'91, Trondheim, Norway, May 1991*, R. Andersen, J. A. Bubenko jr. and A. Solvberg (Ed.), Springer-Verlag, Berlin (1991).

Smo91b Smolander, Kari, "OPRR: A Model for Modelling Systems Development Methods," in *Next Generation CASE Tools*, K. Lyytinen and V.-P. Tahvanainen (Ed.), IOS Press, Amsterdam, the Netherlands (1991).

Smo93a Smolander, Kari, "*MetaEdit+ Protocols and standard operations for processing GOPRR information structures: the Application Programmer's Interface,*" Internal Technical Document, MetaPHOR project, Univ. of Jyväskylä, Jyväskylä, Finland (1993).

Smo93b Smolander, Kari, "*GOPRR: a proposal for a meta level model,*" University of Jyväskylä, Finland (1993).

Sor88 Sorenson, Paul G., Jean-Paul Tremblay and Andrew J. McAllister, "*The Metaview System for Many Specification Environments,*" IEEE SOFTWARE (March 1988) pp.30–38.

Ste93 Stegwee, Robert A., Ria M. C. van Waes, "Flexible CASE tools for Information Systems Planning," pp. 248–292 in *Computer-Aided Software Engineering — Issues and Trends for the 1990s and Beyond*, T. Bergin (Ed.), Idea Group Publishing (1993).

Tei77 Teichroew, Daniel, Ernest A. Hershey III, "*PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems,*" IEEE Transactions on Software Engineering (1977).

Tei80 Teichroew, Daniel, Petar Macasovic, Ernest A. Hershey III and Yuzo Yamamoto, "Application of the entity-relationship approach to information processing systems modeling," pp. 15–38 in *Entity-Relationship Approach to Systems Analysis and Design*, P. P. Chen (Ed.), North-Holland (1980).

Wan93 Wand, Yair, Ron Weber, "*On the ontological expressiveness of systems analysis and design grammars,*" Journal of Information Systems (1993).

Wel92 Welke, R. J., "The CASE Repository: More than another database application," in *Challenges and Strategies for Research in Systems Development*, William W. Cotterman and James A. Senn (Eds.) (Ed.), Wiley, Chichester UK (1992).

Wij90 Wijers, G. M., H. E. van Dort, "*Experiences with the use of CASE-tools in the Netherlands,*" Advanced Information Systems Engineering (1990) pp.5–20.

Wij91 Wijers, G. M., "*Modelling Support in Information Systems Development,*" Ph.D. Thesis, Delft University of Technology, Thesis Publishers, Amsterdam (1991).

Postscript

Since this article was written the Symbol Editor has been significantly extended. Although the article only describes the Symbol Editor in passing, the functionality was already sufficient for the majority of methods. Symbols could be made of a variety of shapes (lines, ellipses, polygons, splines) and labels which held properties. In addition, role symbols specified the colour and thickness of their line type. Text could be aligned horizontally to be left, centre or right-justified in the box that specified the extent of the label. This state of the Symbol Editor was largely the work of Pentti Marttiin.

After I took over development of the Symbol Editor, I added several new features. Labels now also have vertical alignment, and their text may be coloured. More significantly, I introduced the notion of conditional elements of symbols: for each part (shape or label), a condition may be specified as a string (possibly with wildcards) to match the value of a certain property. Only if the string matches will that element be displayed. This allows modelling of the frequent case that a symbol changes graphically according to the value of a property, e.g. the symbol at the end of an Aggregation Whole role changes according to the cardinality: it may be a +, a * or a circle.

I also added a new property data type to GOPRR, 'Vector Graphic': thus whilst normally a property's contents will be a string or a number, here the user can draw a picture, diagram, map etc. The basic editor for drawing these maps is a specialisation of the Symbol Editor. Such 'Vector Graphic' properties may also be included as a 'label' in their object's symbol. As might be expected, the actual appearance of such a label is the contents of the 'Vector Graphic' property, scaled to fit the rectangle specified for the label. In this way more complex symbols can be made up, allowing the modelling of more methods with more precision. Other property data types could also be added, including audio, still picture, video etc., although some of these would not be directly representable in symbols.