CHAPTER 4

METAEDIT+: CASE FUNCTIONALITY TO SUPPORT PRODUCTION, COORDINATION AND ORGANIZATIONAL CONTROL AND INNOVATION

Manuscript (submitted), 1997

METAEDIT+: CASE FUNCTIONALITY TO SUPPORT PRODUCTION, COORDINATION AND ORGANIZATIONAL CONTROL AND INNOVATION*

Steven Kelly, Kalle Lyytinen

Hui Liu, Pentti Marttiin, Harri Oinas-Kukkonen, Matti Rossi, Juha-Pekka Tolvanen

Abstract. Computer Aided Software Engineering (CASE) tools support creation, maintenance, manipulation, retrieval and representation of design information; co-ordinate task execution among developers, provide aids to control activities; and standardise procedures and development performance. In addition, they can foster organisational learning and innovation by supporting creation, abstraction, sharing and maintenance of knowledge about system development products and processes. This paper discusses what functionality a CASE environment should have to support these functions, and highlights the rationale behind the MetaEdit+ environment, which attempts to implement them. We discuss the functionality of MetaEdit+ and assess how it provides services and tools that support production, co-ordination and organisational control and learning. All these tools are generic, i.e. not specific to a given method: methods can be defined and modified freely in MetaEdit+. Much of the functionality is abstracted into a central MetaEngine, significantly reducing the work required to implement each tool. The tool suite includes a diagram editor, a matrix editor, a table editor, a query tool, browsing tools, a code generation and reporting tool, and hypertext and design rationale tools. These tools use the MetaEngine services to help achieve the functionality needed to support organisational learning and innovation.

Keywords:

D2.2 Tools and Techniques: Computer-aided software engineering (CASE)
K.6.3 Software Management: Software development
I.6.5 Model Development: Modeling methodologies
MetaCASE
Tool integration

1 Introduction

CASE (Computer Aided Systems Engineering) environments have been one of the major technological innovations in systems development. Henderson and Cooprider (16) classified CASE functionality into three functions, the first of which is the *production function*, whereby CASE is expected to support

* This research was in part funded by the Ministry of Education, University of Jyväskylä, University of Oulu and the Academy of Finland. The authors' address is Department of Computer Science and Information Systems, University of Jyväskylä, PL 35, FIN-40351 JKL, Finland, apart from Oinas-Kukkonen: Department of Information Processing Science, University of Oulu, Linnanmaa, FIN-90570 Oulu, Finland. Email should be sent to kelly@cs.jyu.fi.

representation, analysis, and transformation of an information system (IS) or its components according to a *method*. The production function deals with the impact of CASE on the capacity of an individual(s) to generate planning or design decisions and subsequent artefacts or products. Second, there is the *co-ordination function*, whereby CASE is expected to enable developers to plan for and enforce rules, policies and priorities that will govern or restrict their activities during the development process, and to enable users to exchange information and or co-ordinate activities for the purpose of influencing the concept, process or product of the development activity. Finally, the *organisational function* covers how through embedded procedures and policies CASE can influence the way the production and co-ordination functionalities are enacted in a given organisational context. In this paper we recognise a fourth, additional dimension of CASE functionality. This deals with a CASE environment's capacity to foster learning, and thereby incorporate process, product and concept innovations into systems development, and further to disseminate and institutionalise such innovations. This function we call the *learning* and *innovation function*.

In their analysis of CASE tools Henderson and Cooprider (16) observed that CASE tools were mostly limited in their functionality to the production function. Here environments supported representation tasks, but largely lacked functionality in analysis and transformation tasks. The survey of Vessey and Sravanapudi showed that nearly all CASE tools were void of any support for co-operative and organisational functions (45). The analysis did not recognise the role of CASE technology in organisational learning and therefore CASE-related product and process innovations were not discussed. Based on the current status of CASE technology we can deem that this kind of support is low (30, 31). Despite these dismaying results researchers in CASE have not discussed why CASE functionality is limited. We think that there are at least two reasons for this. Firstly, CASE development has been product-oriented rather than use-oriented, and thus has normally considered only the production or co-ordination functions. Secondly, CASE developers have tended to focus on one function at a time, and thus CASE tool architectures and platforms have not been designed so that they support and integrate several functions simultaneously.

In this paper our goal is to advance the integrated support of the four desired functionalities in CASE. This will be accomplished by discussing the design of a CASE tool architecture, environment and tool suite which is better equipped to meet the demands of all four functionalities. We demonstrate the viability of our design by reporting its implementation in the MetaEdit+ metaCASE environment (24). The environment offers multi-user, multi-tool, multi-method and multi-form CASE functionality. Thereby it addresses most of the requirements related to production and co-ordination functions, i.e. concurrent creation, maintenance, manipulation, retrieval and analysis of design objects. It also offers functionalities to embed documentation, standards and guidelines for method use, plus support for free yet typed hyperlinking of all model components including traceability and design rationale, thus enhancing the organisational role of CASE. Finally, as a metaCASE and

Computer Aided Method Engineering (CAME) tool, MetaEdit+ provides a flexible and easy-to-use environment for method specification, management, integration and re-use, thus fostering method and process learning and innovation. The metaCASE and CAME side is one of the main advances in MetaEdit+; however, as it is already covered in (24) we shall not discuss it here.

This paper will highlight and evaluate the architecture and features of the MetaEdit+ CASE environment by which it more fully addresses the requirements related to all four functionalities. First we will establish some definitions and examine related research, and then look at the MetaEdit+ environment, its architecture and tools.

2 Background and related research

A number of definitions are in order:

- A *service* is an atomic operation offered by the CASE environment which a user cannot interrupt once started such as **paste**, **delete object** or **generate code.** On the process side it usually corresponds to a production task that has been automated. It may also deal with a co-ordination task such as **notify**, or an organisational task such as **help.**
- A *tool* consists of a set of services providing support for a broader task domain within a function. Examples of tools are a diagram editor or a project scheduler.
- A *tool set* is a small, well-integrated collection of tools that have been organized together in terms of a specific purpose or goal. Tool sets can be composed around process roles (such as an analyst), generic functions such as model editing (as part of the production function), or other criteria such as the scope of tasks executed by the tools.
- A *tool suite* is a larger collection of tool sets and individual tools which together define and provide the current suite of services offered by the CASE environment.
- Finally, *an environment* is a repository and tool suite that follow a specific *integration policy*. By an integration policy we mean those protocols and services which make communication and services possible between multiple users and multiple tools.

Integration policies followed in CASE can be divided into *data, control, presentation,* and *process* integration and their combinations (43). Of high importance here is the way in which data and control integration are carried out. The goal of the former is to ensure that all design information in the environment is managed as a consistent whole regardless of how it is operated on and transformed. This can take place through a shared repository, or by defining the interface formats and services through which different tools can communicate with one another in a software bus. The goal of the latter is to

combine an environment's services according to designers' preferences. It is largely driven by the designer's view of the tasks and processes that the environment is expected to support and how he sees the need for user control. Control can take place by defining the responsibilities between services and by defining invocation and procedure calls between services. Services and their ease of integration depend to a large extent on the integration policies. For example, the lack of co-ordination support in earlier CASE can be largely explained by the lack of consistent data integration mechanisms needed in a multi-user environment, as well as by the lack of control integration mechanisms necessary to achieve multi-user functionality.

Historically, the idea of a suite of integrated services in CASE was outlined in environments such as PSL/PSA (41) and Plexsys (26). PSL/PSA, for example, embedded production function by including model entry and manipulation, model analysis through generic and dedicated tools, and a number of model derivation and output services. Some co-ordination tasks were accommodated by incorporating project management data into the repository data. Moreover, all data in PSL/PSA resided in a centralised database thus achieving a high level of data integration (though rigid and simple). This also offered some possibilities for task co-ordination. A similar strategy was later followed in IBM's A/D Cycle (32). Despite these achievements neither these nor other CASE environments were based on an articulated model for integrating and deriving services. In this regard the most comprehensive study is Chen's dissertation (8), which sought to address both production and co-ordination functions by analyzing how to integrate front end CASE tools with collaborative tasks and group activities (9). Some studies have sought to integrate production-related CASE services with co-operative services by using hypertext technologies (13), or knowledge based technologies (22, 21). Some of these address the need for organisational learning by incorporating metamodeling functionality into the environment (22).

Overall, existing environments offer powerful services for some production or co-operative tasks. However, they neglect many important services including flexible concurrent access, or providing salient support for some production tasks like model editing services. In particular, no other current metaCASE tool offers method definition in the same multi-user environment as CASE with multiple integrated methods and editors. In addition, although many CASE and even a few practically usable metaCASE tools are available today, their development and design are largely unreported in the literature, two welcome exceptions being Software through Pictures (46) in 1986 and ToolBuilder (1) in 1991.

Alongside actual CASE tool development, another research stream has focused on integration policies and their strengths and weaknesses. This has resulted in knowledge of mechanisms through which varying levels of tool and service integration can be achieved (4, 43, 15, 33). The contribution of this stream has been, in particular, in analyzing the integration policies of available services, and the resulting tool properties, their coherency and performance. This research has also emphasised much more strongly the importance of process based integration, i.e. how to ensure that tools interact effectively to

support a defined process, or desired process features. A notable deficiency in this research stream is, however, that it has never asked what services along the four functions of CASE are needed for the defined process, and how these should be organized into tools and tool suites.

3 The MetaEdit+ environment

MetaEdit+ is a metaCASE environment which offers the following properties:

- **multi-user**, i.e. several users can operate concurrently on the repository,
- **multi-tool**, i.e. each user can operate several tools simultaneously where each tool provides a set of services with a different view to the same object,
- **multi-method**, i.e. the environment offers several mechanisms for method integration and consequent consistency checking,
- **multi-form**, i.e. the environment provides several representation formats for the same design object, and
- **multi-level**, i.e. the environment is a true metaCASE environment in that the user can modify and manage both IS models and its design methods within the same environment.

The environment seeks to improve the usability of CASE by supporting multiple simultaneous users, representations forms, several methods and a varying set of services offered by an open tool suite. It is designed for flexibility in that it offers a multi-tool, multi-method approach to CASE use and adoption. It is an open environment in that it fosters environmental evolution by enabling plugging of new tools through well-defined service protocols. By doing so the environment is built for growth in response to the demands that may arise from needs in the production, co-ordination, organisational or learning functions of CASE.

The design principle in developing MetaEdit+ has been to base its architecture in *conceptual modeling, layered data base architectures*, and *object orientation.* We apply conceptual modeling by regarding that the implementation of a method is akin to the development of a conceptual schema for a software repository. Accordingly, the design of a tool resembles a design of an external view to a conceptual schema thereby recognising the need for distinguishing between internal, conceptual and external views of design objects (2). Hence, a method specification language forms the conceptual modeling language of the repository schema and is located at the meta-metamodel level in terms of the IRDS standard (18). The adoption of object orientation enables flexible organisation 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 and method organisation) thus making the environment open.

The use of conceptual modeling and object orientation suggests three principles for the design of MetaEdit+ system architecture: *data independence, representation independence,* and *level independence. Data independence* is defined in a similar way as in data base theory, i.e. tools can operate on design information without "knowledge" of its physical organisation or logical access structure. *Representation independence* forms a continuation of the data independence in that it allows design objects to exist independently of their alternative representations as text, matrix or graphical representations. This principle permits flexible addition of new tools, each one only responsible for its own fundamentally different view on the same underlying design object. *Level independence* means that the environment follows a symmetrical approach in its treatment of data and metadata (24). Accordingly, the specifications of methods and their behaviours can be managed and manipulated in a similar fashion to any other object in the environment (therefore the name metaCASE). Moreover, the specifications can be concurrently operated on through the same or somewhat specialised tools in the environment.

We think that these principles are important in achieving full support for the production, co-ordination, organisational and learning functions of CASE, as will be depicted below.

4 MetaEdit+ components and architecture

The architecture of MetaEdit+ is illustrated in Figure 1. MetaEdit+ runs simultaneously on many clients connected by a network to a server. Each client has its own running instance of MetaEdit+, including all its tools and the MetaEngine. The MetaEngine forms the heart of the environment, from three important points of view: conceptual data, services and communication. Firstly, it incorporates the GOPRR conceptual data model (24), which describes the basic elements of CASE at both model and method level (*level independence*). Secondly, it extends this purely data-based model with behaviour in an object-oriented fashion, handling all operations on the underlying conceptual data through a well-defined service protocol. Tools in a client request services from their MetaEngine while accessing and manipulating repository data: thereby we avoid the need to duplicate the manipulation code. This design choice also allows flexible addition of new tools, each only responsible for its own paradigmatically different view (including view-related services) on the same underlying repository data (*representation independence*).

Within a client the MetaEngine performs a function similar to that of the message server in control integration approaches using message passing (5), e.g. in H-P Softbench (6). The difference is in the sending of messages: in control integration, tools must explicitly send messages when changing data; in MetaEdit+, tools' changes to data handled by the MetaEngine cause automatic messages to be generated from the MetaEngine to other tools, using Smalltalk's dependency mechanism. This clearly reduces the amount of work needed in

tool implementation, as tools no longer need to explicitly send messages when changing data. Even some of the receiving tools' responses are defined by the MetaEngine. Whilst this form of integration is currently only within each client, a similar system, TDE (40), has been able to add this integration between clients via their own notification server, running in addition to the data integration provided by the repository. The disadvantage of their approach is the breaking of the atomicity and consistency provided by an ACID transaction.

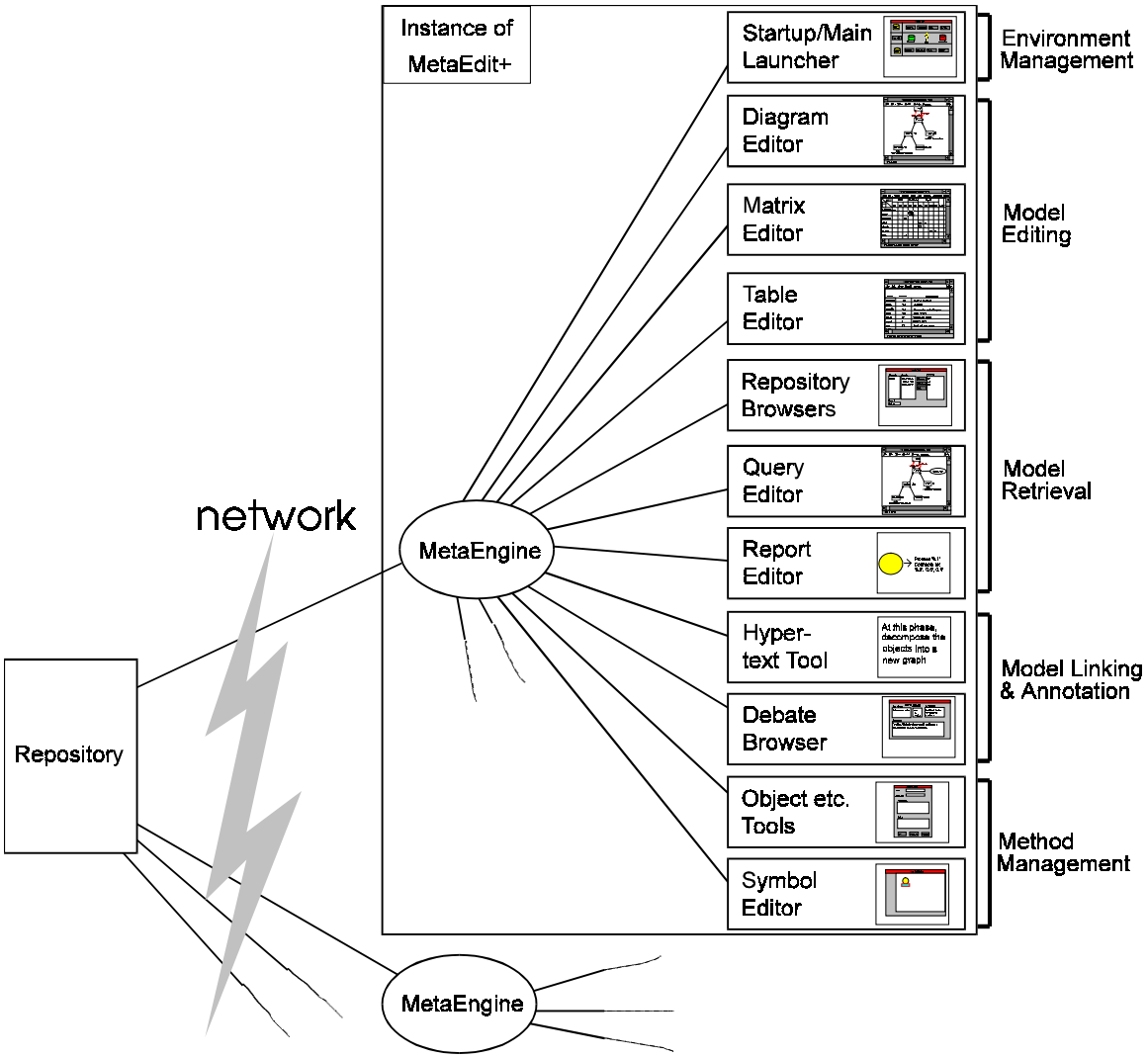


FIGURE 1 MetaEdit+ Architecture

Thirdly, the MetaEngine takes care of all communication both for tools and for the repository (*data independence*). Both data access / integration and control integration with other tools are handled by the MetaEngine, again reducing the overhead of implementing new tools. Similarly, the MetaEngine handles all communication with the server, including concurrency management. Clients do not communicate with each other directly, but rather only through the shared data in the server, accessed via their MetaEngines. Thus the major integration mechanism applied between clients is data integration.

In these three sides of its functionality the MetaEngine thus provides the data independence, representation independence and level independence we

posit as requirements for an integrated CASE architecture, by using object-oriented conceptual modeling over a layered database architecture.

In our database architecture the server is passive, with the workload centred in the clients. The MetaEdit+ 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 *conceptual types*, i.e. semantic method specifications, including rules, code generation and reports; *representational types*, i.e. symbol specifications needed to represent design objects; *conceptual instances*, i.e. the models based on the conceptual types, and *representational instances*, i.e. the representations of those models. The server provides few services, largely restricted to data access (including access rights and locking), and session and transaction management.

The architecture of a MetaEdit+ client is similar to that of the ECMA reference model (14) and PCTE (42):

- Basic user interface services (look and feel) are separated from tools.
- Basic object management services (repository access) are separated from tools.
- New tools can be inserted easily.

However, it differs in several important respects:

- Tools handle only their representational data themselves: operations on conceptual data are requested from the MetaEngine.
- Tools do not communicate with each other directly, but only through the MetaEngine via shared data or simple requests to start another tool.
- Data-related user interface services, e.g. dialogs, are provided by the MetaEngine rather than the User Interface Service.

Thus the MetaEngine has a more important role than in the ECMA reference model: it implements many operations considered there as being in the User Interface Services or tools. We find this to be a more object-oriented approach: conceptual data behaviour (operations, dialogs etc.) is associated with the types that store the data. This abstraction out of conceptual operations from the tools also makes for more lightweight tools: defining a new tool is significantly easier. Further, coupling of tools to the rest of the environment is simplified by moving data operations from the User Interface Services to the MetaEngine: tools use the MetaEngine for all their data and control needs.

This architecture also allows new tools to add functionality to existing tools by a novel mechanism, which we call embedding. For instance, a menu for the Hypertext tool (see Section 7) is added by the MetaEngine to all model editing tools as they follow the standard opening protocol. The tools have no direct link to the Hypertext tool, and no special code to call or use it, nor does the Hypertext tool have a direct link to the tools, or special code to interface with each tool, yet all the functionality of the Hypertext tool is available to the user within each editor, fully and seamlessly embedded there. The Process

Management Services, present in PCTE but currently only under development in MetaEdit+, are being added in a similar way.

4.1 The MetaEngine and its data architecture

Because all method specifications in MetaEdit+ are interpreted as high level conceptual models the kernel of the MetaEdit+ functionality is determined by the underlying conceptual data model called GOPRR (24) which is used as a universal meta-metamodel. Very little if any method "knowledge" is buried into the code in tools. The MetaEngine embodies the implementation of the GOPRR model and its behaviour.

The basic GOPRR elements, called *metatypes*, are Graph, Object, Property, Relationship and Role. These elements are used symmetrically on both the type and instance levels (level independence). An element of a particular method is called a *type*, and belongs to a metatype: e.g. Process in a Data Flow Diagram is a GOPRR type whose metatype is Object. The MetaEngine specifies how *instances* of types (e.g. a specific Process with number 2.1) behave using information from two sources: the fixed definitions of how an instance of a metatype behaves in general, modulated by the specific information contained in the Process type. For example, when creating an Object instance the MetaEngine knows that it must create some Property instances (fixed metatype information), and the Process type specifies how many Properties and of what types (specific type information). A tool, in contrast, knows neither of these; it simply knows the user has asked to create a Process, and passes this request on to the MetaEngine, which creates the Process and its properties, allows the user to fill in the properties, and then returns the created Process to the tool.

The use of MetaEdit+ is thus not radically different from other CASE tools: all the same basic functionality is present. The major architectural difference is that MetaEdit+'s tools can be regarded as *generic*, syntax-oriented editors: the syntax of the method has been defined through a GOPRR specification, and the tool follows universal mapping rules which determine how each GOPRR metatype is represented and behaves. The benefit of this approach is that it makes possible fast and flexible specification of a new method editor, and provides a uniform user interface across all "specialised" method based editors. Its drawback is its generality: it cannot offer the fine details of individual method support in the same way as a diagram editor dedicated to a single method. Such fine details include honouring specialised semantics attached to spatial locations, or the exact representation and rules of complex graphical symbols. Support for details such as these could be added to MetaEdit+, but at the expense of much greater complexity of the metamodelling process: building CASE tool support for the method would then involve programming, reducing the metamodelling facilities to little more than a library of useful pieces of code for CASE tool implementation.

4.2 Tool sets

In the design of the environment we have organized tools into five distinct tool sets according to their purpose and underlying common functionality. These functionalities and their organisation into tool sets has been derived from our analysis of needs to support production, co-ordination, organisational support and learning tasks. From the viewpoint of conceptual data and its behaviour each tool set portrays similar demands in terms of manipulation, locking and retrieval, though different representational paradigms followed in tools may pose additional demands. These have to be dealt with individually in each tool. Each tool set contains one or more tools (Table 1). The five tool sets are the following:

- **Environment management tools**: These tools help manage features of the environment, and the other main components are launched from here.
- **Model editing tools**: These tools primarily serve the production function and therefore tools within this set can be used to create, modify and delete model instances or their parts. In addition, these tools can be used to view the model instances from alternative representational viewpoints, and/or to derive new information from existing design information. The editing tools can provide extensive information on the current metamodels and their use, thus supporting the learning and organisation functions.
- **Model Retrieval tools**: These tools primarily serve production functions in that the tools retrieve design objects from the repository for reuse and review. The tools can operate on metamodels as well as models, and thus also support the learning and organisation functions.
- **Model linking and annotation tools**: These tools serve all four functions due to their versatility. They link design objects for traceability and memorisation, annotate model instances for learning and standardisation, find specific "locations" in the design space for manipulation, or maintain conversations about design instances or method instances (learning).
- **Method management tools**: These tools serve all four functions at the meta-level, as the methods resulting from their production functionality embody the other functionalities when taken into use. Their primary purpose is to support the method production function, but by doing so they also enable learning and innovation within and through the environment. Specific tools in this tool set include tools for method specification, management and retrieval. The method management toolset is described in (24).

In the following sections we will examine the three tool sets that relate to models, considering how they support the four CASE functionalities.

TABLE 1 MetaEdit+ Tool Families

Tool Set	Tool	Tool Functionality
Environment Management	*Startup Launcher* *Main Launcher* *Options Tool*	Initialisation of the environment, login, launching of other tools, modification of run time parameters
Model Editing	*Diagram Editor*	Manipulation and creation of models where objects and relationships can be viewed and manipulated as graphical diagrams
	Matrix Editor	Manipulation and creation of models viewed and edited as matrices, and algorithms performed on them to aid design decisions (23)
	Table Editor	Manipulation and creation of objects in models and all their properties at once. This is especially useful for requirements analysis.
Model Retrieval	*Repository Browsers*	Allows hierarchical access to models and metamodels stored in the repository;
	Query Editor	Allows formulation and use of graphical queries on the models stored in the repository (27)
	Report Editor	Generates textual descriptions of the models stored in the repository using a procedural query and data manipulation language.
Model Linking and Annotation	*Linking Ability hypertext tools*	Provides functionality to add notes and links to any design objects in the repository which can then be seen by any other editing tool.
	Debate Browser	Provides functionality to maintain and trace conversations about design decisions
Method Management	*Object tool* *Property tool* *Relationship tool* *Role tool* *Graph tool*	Specification of conceptual types and their rules, links and integration (24)
	Symbol Editor	Specification and design of graphical objects and their behaviours. Linking of graphical objects to conceptual object types

5 Model editing tools

Model editing tools form MetaEdit+'s key functionality from the users' point of view when viewed as *production technology*. In particular, the tools help create, modify and delete model instances. All model editing tools are similar insofar as their main purpose is to help manipulating and creating models. They differ in terms of their *focus* and *representational paradigm supported.* Some tools are object-centered, whereas others are relationship- or property-centered. Taken as a whole this tool set provides a great variety of representation, manipulation and analysis services on the underlying conceptual design objects.

5.1 Diagram Editor

Most current methodologies use graphical techniques to represent models. Examples galore: structured methodologies like ISAC (28) and SA (48), or object-oriented methodologies like OMT (38), OOA/OOD (10), Fusion (11), and Shlaer-Mellor (39). This makes a graphical editor or a diagram editor, as we call it, *sine qua non* for the usability of any CASE environment. Consequently, all recent CASE environments have been built around a dedicated graphical editor(s) that supports the chosen diagrammatic methods.

MetaEdit+'s Diagram Editor supports all the normal functions found in other CASE tools' graphical editors, automatically adapting its menus, symbols etc. to the information specified in the types of the method being used. For instance, the Graph menu (see Figure 2) includes functionality to view information and properties of this graph, open this graph or other graphs in other editors, and run or create reports and code generation for this graph. All this functionality is supplied by the MetaEngine, and works and looks the same in all editors. Similarly the toolbar (bottom left), Types menu and Help menu functionality is provided by the MetaEngine, as are some actions on the Edit menu.

User interaction is object oriented in two important ways: it follows the 'select-operate' ordering, and the available actions in the popup menu associated with each element are different for different selected elements, e.g. only relationships have the 'straighten line' menu option.

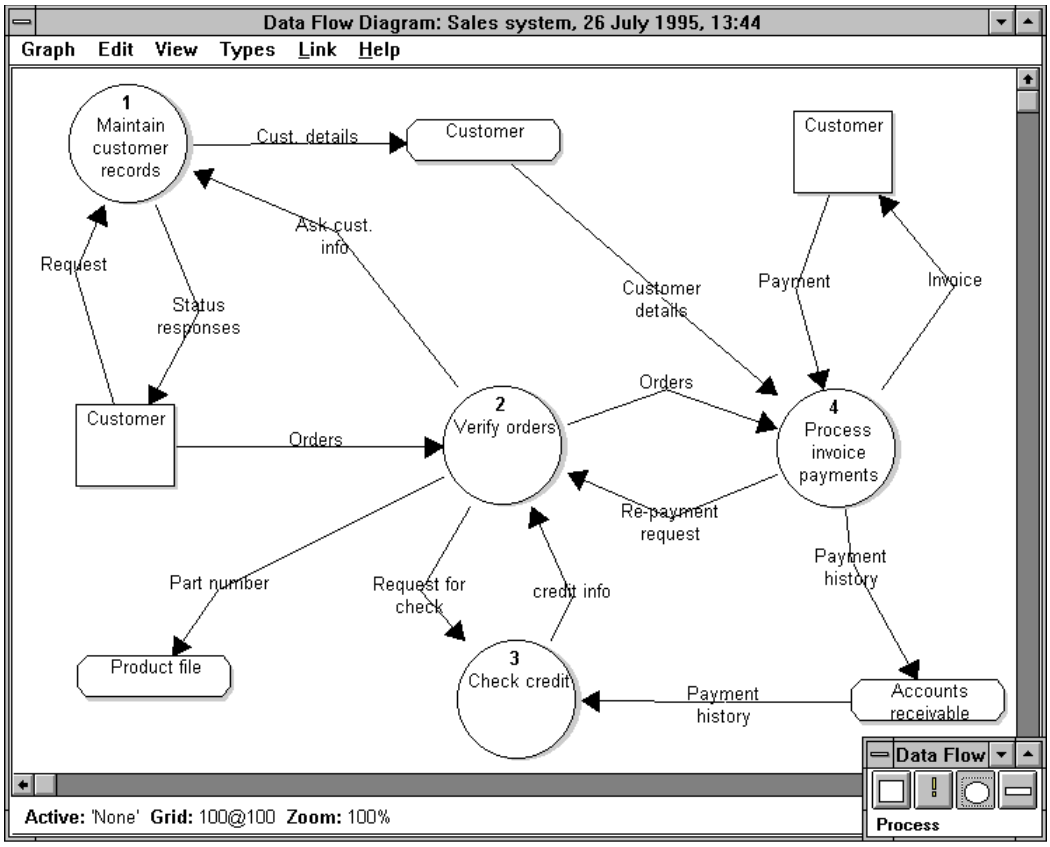


FIGURE 2 The Diagram Editor showing a Data Flow Diagram

5.2 Matrix Editor

Matrix representations are used in representational and analysis functions in many methods, such as structured methods (28) and business process re-engineering (17, 29). Whilst CRUD matrices are the best known, any model can also be represented in a matrix form. The benefit of this form is that a matrix representation focuses on relationships and helps users infer global properties of the whole system or its parts based on the nature of the focused relationships. Another advantage of the matrix representation is that it allows fast input of relationships between objects and their properties.

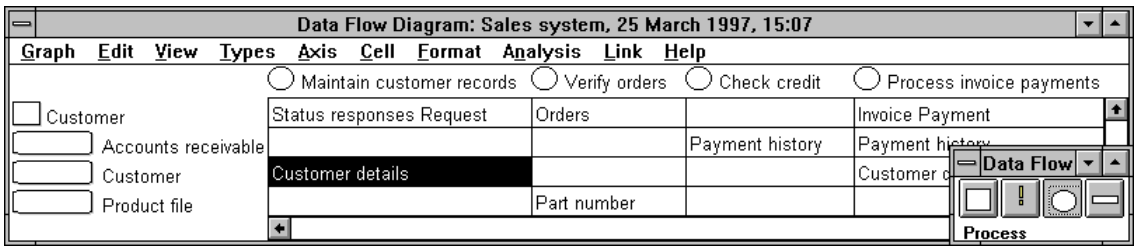


FIGURE 3 The Matrix Editor showing a Data Flow Diagram

In the example in Figure 3, a user has opened a Matrix Editor on a Data Flow Diagram of a Sales System (cf. Figure 2). The selected element 'Customer

details' on the left shows that there is a flow of customer details between the 'Maintain customer records' Process and the 'Customer' Store.

The user can choose to view any graph instantly as a matrix, and thereby easily benefit from the matrix representation in those tasks where the matrix is a better-suited representation paradigm. Such tasks include organising large collections of objects into groups, especially when there are many relationships involved which would make rearranging a diagram representation difficult. The user can arrange groups of axis items, divide up the functionality, data etc. of the modeled system, using a combination of sorting, diagonalisation and manual approaches not possible in diagram form. As the many extra menus (between the Types and Help menus) show, there is much added matrix-specific functionality that is not available in other editors: most of this relates to different ways of building and showing the matrix. When creating a new matrix, the editor automatically makes initial formatting choices on the basis of the metamodel, saving the user work. The matrix editing functionality is described more fully in (23).

5.3 Table Editor

A Table (form) representation complements the matrix and diagram representations and offers a third way to represent and analyze design data. It has been included into the model editing tool set because of its two useful features. First, it provides a tabular or form based view on conceptual design objects: this is often needed in data gathering and review with users who often find filling in a form for each item easy compared to establishing relationships between items. Second, it lends support for methods that use organized lists or forms in capturing or organising design data. These methods include for example Critical Success Factors (36), root definitions (7), problem and goal lists (28), use cases (19), or even requirements specification standards such as DOD standards.

In the Table Editor design objects are represented as rows and properties of these objects form columns. Thereby the editor provides a natural way to view simultaneously design information concerning multiple objects in a compact form. Figure 4 shows our example of a Data Flow Diagram as seen through the Table Editor (cf. Figure 2 and Figure 3). As can be seen, the Table Editor does not use the MetaEngine's toolbar and Type menu services: there is no need for them as each table only displays objects of one type, here Processes.

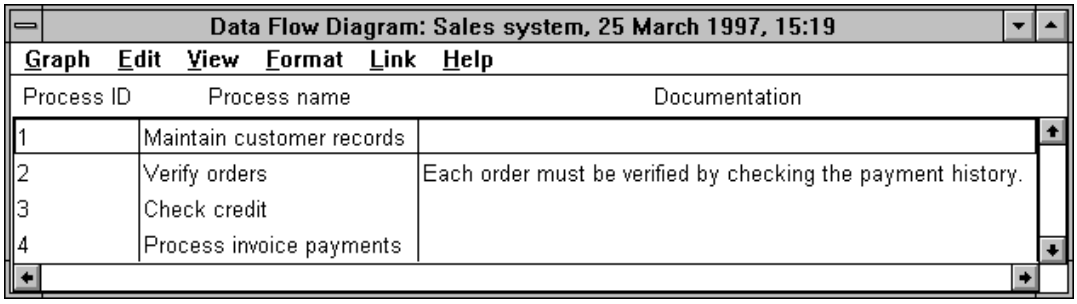


FIGURE 4 The Table Editor showing Processes from a Data Flow Diagram

6 Model retrieval tools

Model retrieval tools become essential when the size of the repository (both at the model instance level and at the method level) grows. They are needed to navigate through and retrieve model instances that meet specific retrieval conditions. Moreover, these components can often be modified or transformed. Consequently, method retrieval tools are necessary if wider re-use of method components or design objects is sought for. All tools in this family agree with their main purpose, i.e. to retrieve model instances and components from the repository but differ in terms of access functionality, ease-of-use and preferred retrieval strategy. Some of the tools are navigational, whereas others are set- or predicate-oriented. They also differ in how the retrieval functionality is represented to the user. Taken as a whole the family provides a rich variety of retrieval forms and functionality for the underlying conceptual data.

6.1 Repository Browsers

MetaEdit+ browsers can be used for viewing, navigating through, and editing individual objects in the repository. All browsers operate on the conceptual level, i.e. when an object has been selected it can be viewed through different tools as a diagram or a matrix. Overall, browsers provide a quick list-based textual view of the defined models and their components and are extremely useful in model re-use by helping to find objects that meet some criteria: they include basic wild card searches, and restriction by type, graph, or project. In this way they foster the organisational and learning functions, although their main function should still be seen as production.

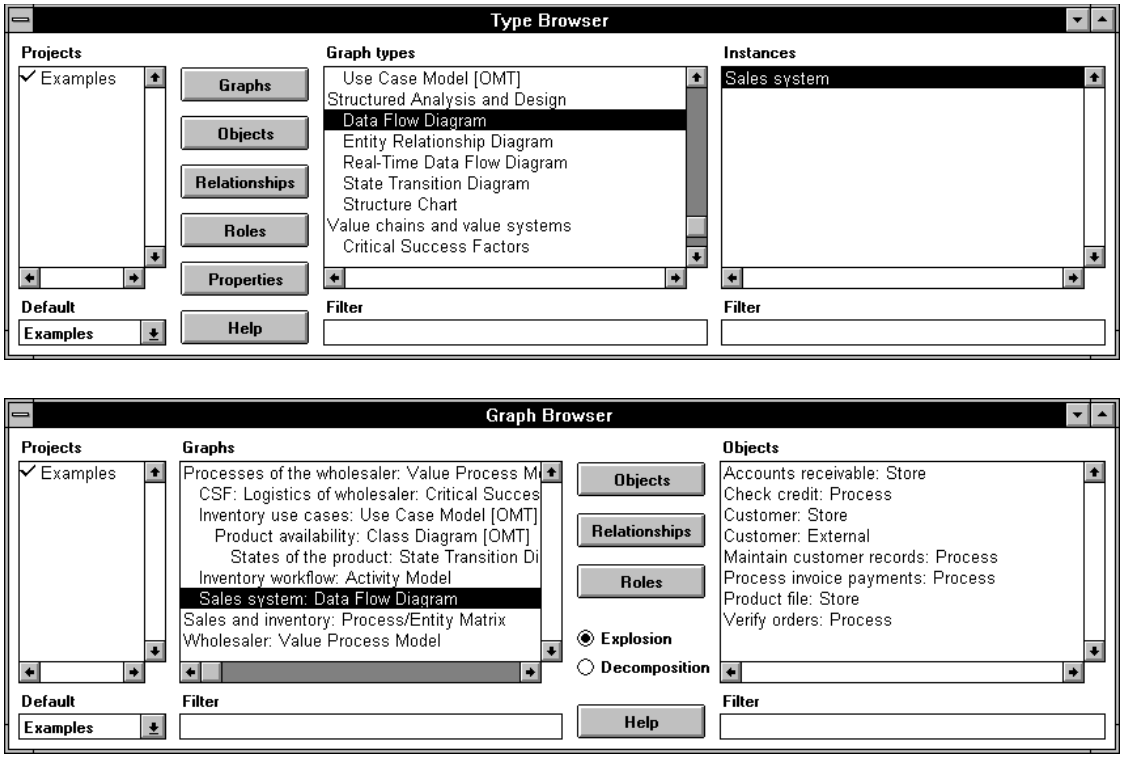


FIGURE 5 Type & Graph Browsers

To accommodate different browsing needs, MetaEdit+ includes two browsers: a Type Browser and a Graph Browser, shown in Figure 5. The list on the left hand side of both browsers shows all the currently open projects, and the middle list shows a hierarchical list of all the types or graphs in the selected project(s). In the right hand list, the Type Browser shows all instances of the selected type in the selected project, whereas the Graph Browser shows all the objects, roles or relationships in the selected graph. Both Browsers allow the users to edit the selected instance or type by opening the appropriate model editing tool or property window. Use and definition information about the selected item in the form of separate Info Window, which gives detailed information about the status of the item, and where it is used.

6.2 Graphical Query Tool

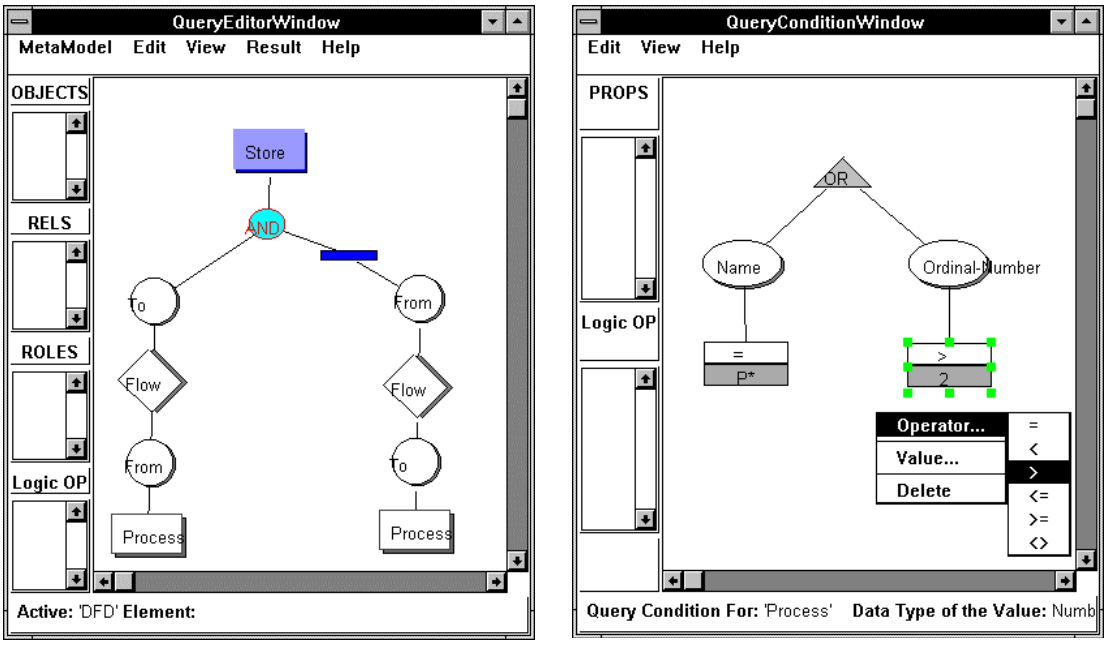
The query tool allows the user to query the repository using a high level graphical query language, providing help in the production function and also the learning function: new users can easily find information they need based on queries, rather than needing to know where the information is stored. Two aspects have played a central role in the development of the query tool: user friendliness and expressive power. As the visual query paradigm (3) has been shown to be capable of addressing both aspects (37, 47), it has been adapted as the basis of the tool's user interface. The expressive power of the query tool's powerful pattern matching facilities fosters informed re-use, while the clear visual paradigm provides for ad hoc searching and information needs.

To improve usability, query formulation is performed by syntax-directed editing (27). In addition, the tool provides instant feedback during the course of the query formulation in that it can execute a query at any intermediate step of the formulation process. These strategies minimise the possibility that the user will construct an erroneous query, and relieve the unnecessary cognitive load of having to remember the syntax of the query language.

Figure 6 displays a sample query in the query tool over the example DFD. The query window (A) shows the conditions 1a and 2, and the condition window (B) shows the query condition 1b for the process selected on the left of the query window (A).

Find those stores in a Data Flow Diagram which meet the following two conditions:

1a. The store receives a Data Flow from a process that:
1b. **either** has a name with capital starting letter P, **or** its Ordinal-Number is greater than 2; **and**
2. The store does not direct any Data Flow to process XXX.



(A) Query window (B) Query condition window

FIGURE 6 The graphical query representation

6.3 Report Editor

Important abilities of any CASE tool are to check models, obtain textual reports of the contents of models, and generate program code from models. Reflecting its importance, this functionality addresses issues in all four of our functions: production, co-ordination and organisation (by defining and running checking reports we plan for and enforce rules), and learning and innovation (by automating the generation of code and documentation). In MetaEdit+ these

functionalities are supplied via a structured, GOPRR-oriented reporting language, which supports flow control, text output and basic text formatting. Reports are defined in the Report Editor, which supports straight text entry of the report language and also template-based addition of the flow control statements and GOPRR elements: all the Object, Relationship and Role types, as well as their various properties, can be selected directly from the template lists, avoiding typing errors and relieving the user of the burden of remembering all the method constructs. The syntax of the reporting language is presented in (44).

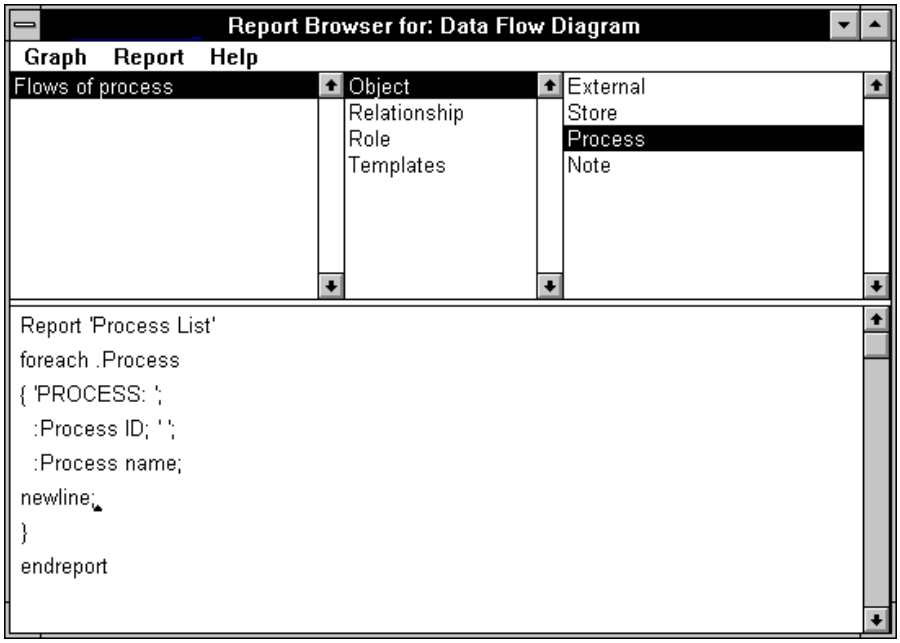


FIGURE 7 Report Editor

For example, if we were interested in listing all the Processes in a Data Flow Diagram with their Process IDs and Names we could write the code shown in Figure 7.

7 Model linking and annotation tools

In addition to basic tools that manipulate and retrieve models based on the well defined method specification (grammar), one needs tools that enable the creation, modification and deletion of navigational hyper-links between models or parts of them (13). Such a family of tools we call *model linking and annotation tools*. Currently most CASE tools enable the creation and manipulation of models conforming to the method specification, but hardly ever lend support for representing ad hoc or informative model connections. This results in weak model integration and poor transparency of use, and the consequent lack of traceability and design history recording has been shown to be one cause for design and maintenance problems (20).

Allowing free hypertext linking of all model components supports the production. Traceability links (e.g. between a requirement and the design object that implements it) can be used as a co-ordination function to provide an access structure complementary to the normal hierarchy of graphs. In addition the ability to create subtypes and keywords for links and to form glossaries supports the learning and innovation function.

7.1 Linking Ability: Hypertext editor

In MetaEdit+ we use "embedded" CASE hypertext functionality to offer a generic linking ability across all model editing tools (see 35). Several link types are used: association between two GOPRR elements, descriptive annotation of an element, traceability (e.g. from a requirements object to the corresponding design object), and debate links. A debate link connects an element to a debate in the Debate Browser, which captures the design rationale for that element in the form of a debate graph of questions, answers and arguments, similar to e.g. gIBIS (12).

As an example, see Figure 8. The 'Customer' Store in the Data Flow Diagram on the left has two links, as shown by the '2' by its symbol. One link is to an Annotation, 'Customer database', which explains the linking of the Store with an existing database: note that the annotation itself contains two further hypertext links, shown underlined. The second link from the DFD 'Customer' Store is an Association to the 'Customer' Class in an OMT Class Diagram: the Class is the implementation of the Store. The activation of any of these links opens up the corresponding node, e.g. we can activate the 'Customer' Store association link, which will open the Class Diagram and select and scroll to the 'Customer' Class.

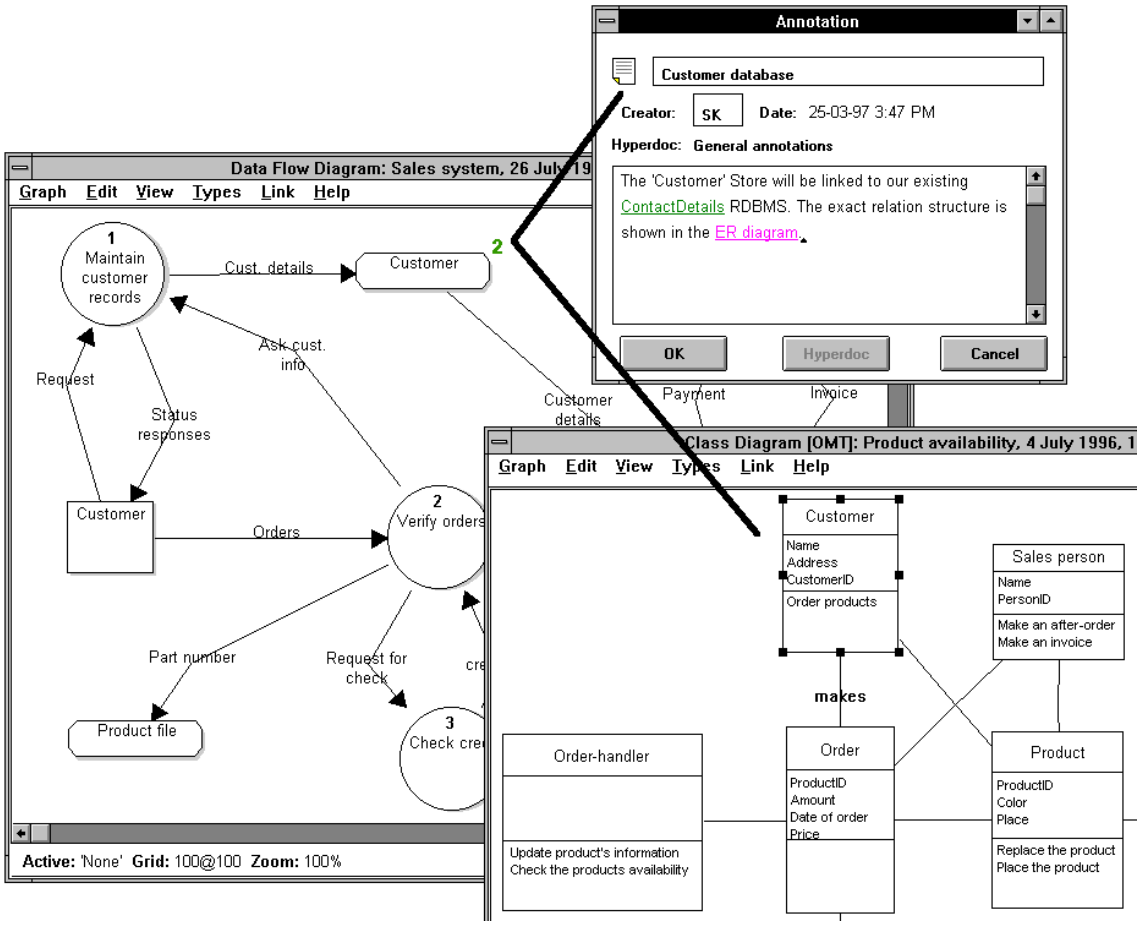


FIGURE 8 Hyper-links and annotations in diagrams

To help manage the added complexity of hyper-links, a variety of navigational aids have been implemented. These include bookmarking capabilities, a history list, filters, and text string searches. Queries may be made to find links based on properties including label, creator, creation timestamp, subtype, and keywords.

7.2 Debate Browser

Debate Browser is an annotation and review tool that supports the capture and exploitation of arguments behind design decisions. These are expressed using three kinds of nodes, questions, answers, and arguments, each focusing on articulating one key *aspect* of the design problem. The Debate Browser thus co-ordinates design resolution amongst the users, and fosters learning and innovation by allowing discussion and recording of design decisions and their reasoning in a structured reusable format: it is described more fully in (34).

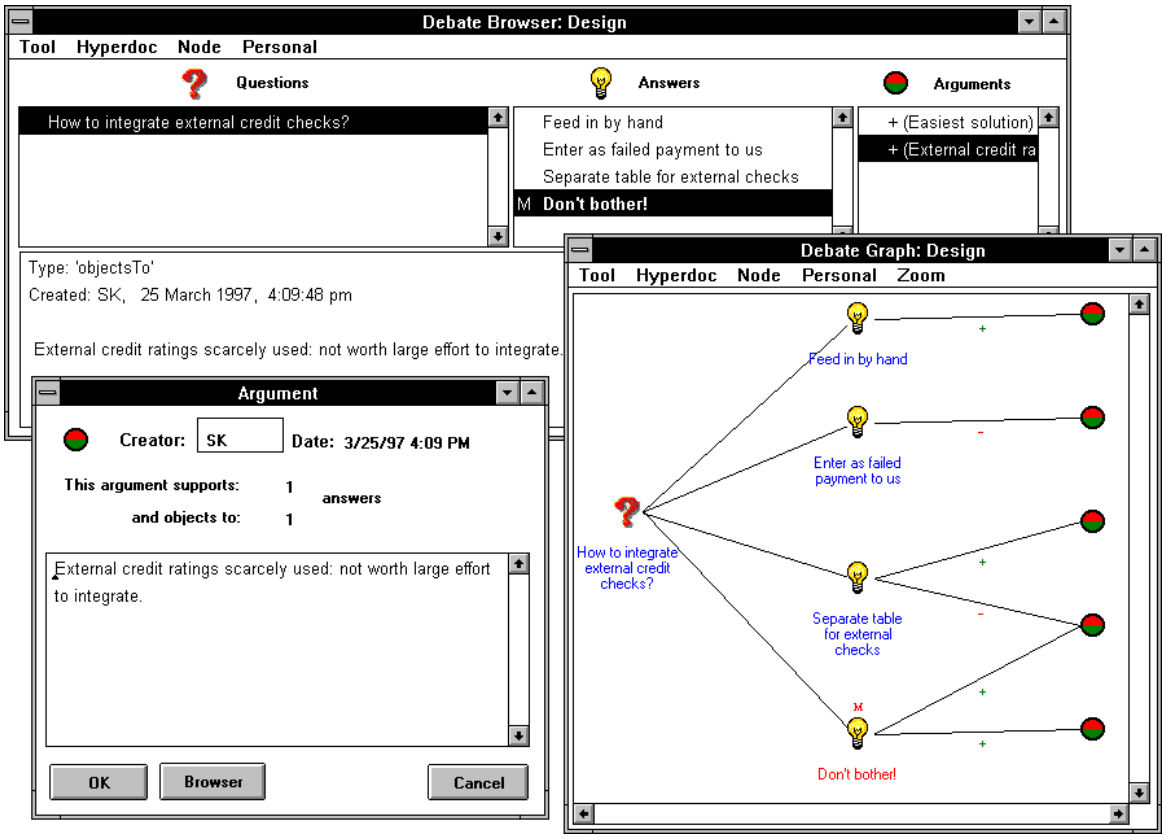


FIGURE 9 Debate Browser

The Debate Browser has two tools for investigating the design rationale, a textual browser and a graphical form. As an example, Figure 9 shows a debate about the credit checking for the Sales System, with both Browser and Graph representations of the same debate, with a dialog opened on the argument selected in the Browser.

8 Discussion and conclusions

Overall, we have sought to develop MetaEdit+ as a multi-tool platform for trying out different tools and tool construction principles, and the use of object oriented architecture in designing and implementing a CASE tool. This is well reflected in its current implementation. MetaEdit+ has been implemented in a highly object-oriented fashion using the VisualWorks Smalltalk environment with the ArtBase object repository system and NEDT graphical extensions. This approach has allowed us to concentrate on new functionality and the synthesis of existing separate advances, rather than basic graphics behaviour or user interface programming.

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

- 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 and pluggable tools;

- 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 encountered in systems development with some new functionality such as matrices, hypertext tools and the query tool;
- includes flexible mechanisms for tool integration and both vertical and horizontal method integration support;
- is the first CASE tool to offer ad hoc hypertext linking in models, with CASE-specific link types including design rationale and traceability;
- provides symmetrical treatment of IS models and metamodels, and thus enables re-use, metamodel management and utilisation within the same environment;
- provides support for alternative representational paradigms including matrices, tables and hypertext.

As we have shown, MetaEdit+ offers the necessary functionality for supporting the production function (through multiple editors and reporting tools), co-ordination function (through multi-user support and powerful integrity enforcement mechanisms, and hypertext), organisational function (through powerful help functionality, query systems and hypertext), and learning and innovation function (through metamodeling capabilities, symmetrical access to metamodels and hypertext functionality). We believe that with these features MetaEdit+ addresses many flaws found in current CASE tools (16).

MetaEdit+ also offers some innovative approaches to method management and use. First, through its novel method integration mechanisms it provides ways to organise methods and method families into methodologies, and also to organise methodologies with alternative levels of connectedness and inter-method integrity constraints. Second, 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. Third, 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 and functions. First, the current version of MetaEdit+ does not provide a uniform environment for process specification, enactment and enforcement which is critical for improved co-ordination support. This functionality is currently under development (25). Second, MetaEdit+ supports only a single multi-user repository, and thus does

not address the need for multiple distributed repositories which is typical for software development in the large and by many. Third, it does not yet provide flexible mechanisms for integration with external co-ordination or production function tools (such as electronic publishing or CSCW tools).

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

Acknowledgements

We are grateful to many colleagues, including Kari Smolander, Janne Kaipala, Pentti Kerola, Minna Koskinen, Janne Luoma, Juha Pirhonen, Risto Pohjonen, Marko Somppi, and Veli-Pekka Tahvanainen who have been involved in designing and implementing some parts of the system.

References

1. Alderson, Albert Meta-CASE Technology, Software Development Environments and CASE Technology, Proceedings of European Symposium, Königswinter, June 17-19, A. Endres and H. Weber (ed.), 509, Springer-Verlag, Berlin (1991), 81-91.
2. ANSI, Study Group on Data Base Management Systems: Interim Report 75-02-08, ACM SIGMOD Newsletter 7, 2 (1975).
3. Batini, C., Catarci, T., Costabile, M.F. and Levialdi, S. Visual Query Systems, Report 04.91, Università Degli Studi di Roma 'La Sapienza' (1991).
4. Brown, A., McDermid, J.A. Learning from IPSE's mistakes, IEEE Software (March 1992), 23–28.
5. Brown, Alan W. Control Integration through Message Passing in a Software Development Environment, CMU/SEI-92-TR-35, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (1992).
6. Cagan, Martin R. The HP SoftBench environment: an architecture for a new generation of software tools, Hewlett-Packard Journal 41, 3 (1990), 36–47.
7. Checkland, P.B. *Systems Thinking, Systems Practice*, J. Wiley, New York (1981).

8. Chen, M. The Integration of Organization and Information Systems Modeling: A Metasystem Approach to the Generation of Group Decision Support Systems and Computer-aided Software Engineering, PhD Thesis, University of Arizona, Tuscon, USA (1988).
9. Chen, M., Nunamaker, J.F. Jr. and Mason, G. The Architecture And Design Of A Collaborative Environment For Systems Definition, Database (Winter/Spring 1991), 22–28.
10. Coad, P., Yourdon, E. *Object-Oriented Analysis*, Englewood Cliffs, New Jersey (1990).
11. Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P. *Object-Oriented Development: The Fusion Method*, Prentice Hall, Englewood Cliffs (1994).
12. Conklin, J., Begeman, M.L. gIBIS: A Hypertext Tool for Exploratory Policy Discussion, ACM Transactions on Office Information Systems 6, 4 (1988), 303–331.
13. Cybulski, J.L., Reed, K. A Hypertext-Based Software Engineering Environment, IEEE Software (March 1992), 62–68.
14. ECMA, Reference Model for Frameworks of Software Engineering Environments, Technical Report ECMA TR/55, 2nd Edition (1991).
15. Fernström, C., Närfelt, K.-H.n and Ohlsson, L. Software Factory Principles, Architectures, Experiments, IEEE Software (March 1992), 36–44.
16. Henderson, J., Cooprider, J. Dimensions of IS Planning and Design Aids: a functional model of CASE technology, Information Systems Research 1, 3 (1990), 227–254.
17. IBM Corporation, Business Systems Planning — Information Systems Planning Guide, Publication #GE20-0527-4, IBM (1975).
18. ISO, Information processing systems: Information Resource Dictionary System (IRDS) Framework, Draft International Standard ISO/IEC DIS 10027 (1989).
19. Jacobson, I., Christeson, M., Jonsson, P. and Övergaard, G., *Object-Oriented Software Engineering — A Use Case Driven Approach*, Addison-Wesley, Reading, USA (1992).
20. Jarczyk, A.P.J., Löffler, P. and Shipman, F.M. Design Rationale for Software Engineering: A Survey, Proceedings of the 25th HICSS, Hawaii, 2, IEEE Computer Society Press, Los Alamitos, CA (1992), 577–586.
21. Jarke, M., Jeusfeld, M. and Rose, T. A Software Process Data Model For Knowledge Engineering In Information Systems, Information Systems 15, 1 (1990), 85–116.
22. Jarke, M. Strategies for Integrating CASE Environments, IEEE Software (March 1992), 54–61.
23. Kelly, S. A Matrix Editor for a MetaCASE Environment, Information and Software Technology 36, 6 (1994), 361–371.

24. Kelly, S., Lyytinen, K. and Rossi, M. MetaEdit+: A fully configurable multi-user and multi-tool CASE and CAME environment, Advanced Information Systems Engineering, proceedings of the 8th International Conference CAISE'96, P. Constapoulos, J. Mylopoulos and Y. Vassiliou (Ed.), Springer-Verlag (1996), 1–21.
25. Koskinen, M., Marttiin, P. Process Support in MetaCASE: Implementing the Conceptual Basis for Enactable Process Models in MetaEdit+, Proceedings of Software Engineering Environments, SEE'97, Gottbus, Germany (1997), 110–122.
26. Kottemann, J.E., Konsynski, B.R. Dynamic Metasystems for Information Systems Development, Proceedings of the Fifth International Conference on Information Systems (1984), 187–204.
27. 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).
28. Lundeberg, M., Goldkuhl, G. and Nilsson, A. *Information Systems Development: a systematic approach*, Prentice-Hall (1981).
29. Martin, J. *Strategic Information Planning Methodologies*, Prentice-Hall, Englewood Cliffs, NJ (1989).
30. Marttiin, P., Rossi, M., Tahvanainen, V.-P. and Lyytinen, K. A Comparative review of CASE shells: A preliminary framework and research outcomes, Information & Management 25 (1993), 11-31.
31. Marttiin, P., Harmsen, F. and Rossi, M. Evaluation of two CAME environments using a functional framework: findings on Maestro II/Decamerone and MetaEdit+, Method Engineering, Principles of method construction and support, Proceedings of the Method Engineering '96, Proceedings of IFIP 8.1/8.2 Working Conference on Method Engineering, S. Brinkkemper, K. Lyytinen and R. Welke (Ed.), Chapman-Hall, London (1996), 63–86.
32. Mercurio, V.F., Meyers, B.F., Nisbet, A.M. and Radin, G. AD/Cycle strategy and architecture, IBM Systems Journal 29, 2 (1990), 170–188.
33. Mi, P., Scacchi, W. Process Integration in CASE Environments, IEEE Software (March 1992), 45–53.
34. Oinas-Kukkonen, H. Debate Browser — An Argumentation Tool for the MetaEdit+ Environment, Proceedings of 7th Workshop on the Next Generation of CASE Tools, NGCT'96, Crete, Greece (1996), 77–86.
35. Oinas-Kukkonen, H. Towards Greater Flexibility in Software Design Systems through Hypermedia Functionality, Information & Software Technology, 39, 6 (1997), 391–397.
36. Rockart, J. Chief Executives Define Their Own Data Needs, Harward Business Review 57, 2 (1979).
37. Rosengren, P. Using Visual ER Query Systems in Real World Applications, Proc. of the 6th international Conference, CAiSE'94, G. Wijers, S. Brinkkemper and T. Wasserman (Ed.), Wiley, Utrecht, The Netherlands (1994), 394–405.

38. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, NJ, USA (1991).
39. Shlaer, S., Mellor, S.J. *Object Lifecycles: Modelling the World in States*, Prentice Hall, Yourdon Press Computing Series, Englewood Cliffs, NJ, USA (1992).
40. Taivalsaari, A. and Vaaraniemi, S. TDE: Supporting Geographically Distributed Software Design with Shared, Collaborative Workspaces, Proceedings of CAiSE '97, Barcelona, Catalonia, Spain, June 16–20, A. Olivé and J. A. Pastor (ed.), 1250, Springer, Berlin (1997), 389–408.
41. Teichroew, D., Hershey, E.A. III. PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems, IEEE Transactions on Software Engineering (1977).
42. Thomas, I. PCTE Interfaces: Supporting Tools in Software-Engineering Environments, IEEE SOFTWARE 6, 6 (1989), 15–23.
43. Thomas, I., Nejmeh, B.A. Definitions of Tool Integration for Environments, IEEE Software (March 1992), 29–35.
44. Tolvanen, J.-P., Kelly, S. MetaEdit+ User's Guide, Technical Report, TKTL, University of Jyväskylä, Jyväskylä (1995).
45. Vessey, I., Sravanapudi, A.P. CASE tools as collaborative support technologies, CACM 38, 1 (1995), 83–95.
46. Wasserman, A. I. and Pircher, P. A. A Graphical, Extensible Integrated Environment for Software Development, Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments (December 1986), 131–142.
47. Weiland, W.J., Shneiderman, B.A. A Graphical Query Interface Based on Aggregation/Generalization Hierarchies, Information Systems 18, 4 (1993), 215–232.
48. Yourdon, E. *Modern Structured Analysis*, Prentice-Hall, Englewood Cliffs, NJ, USA (1989).