HyperEDIT:
An Object-Oriented Diagram Meta-Editor

Jacob L. Cybulski and Arthur Proestakis
Amdahl Australian Intelligent Tools Program
Department of Computer Science and Computer Engineering
La Trobe University, Bundoora, Vic 3083, Australia
Phone: +613 479 1270, Fax: +613 470 4915

Abstract

This document describes the function and the implementation of HyperEDIT, an object-oriented diagram meta-editor. The editor may be used in a stand-alone mode as a fully featured graphical tool for the production of high quality software design documents, such as Data Flow, Entity-Relationship, or State Transition diagrams. Alternatively, thanks to its sophisticated event-protocol, it may be incorporated in other products as their extendible graphical user interface, e.g. in Hypertext or CASE tools.

1. Introduction

HyperEDIT

A picture says a thousand words, and in software engineering this is certainly the case. From the work of De Marco, Yourdon, Chen, Bachman, Nijsson, Gane and Sarson, Warnier and Orr, Jackson, etc., diagramming has become the preferred medium for describing software systems [1]. Because hand drawing of large and complex diagrams is a tedious and highly inefficient task, especially when describing evolving, dynamic systems the technology of CASE was developed.

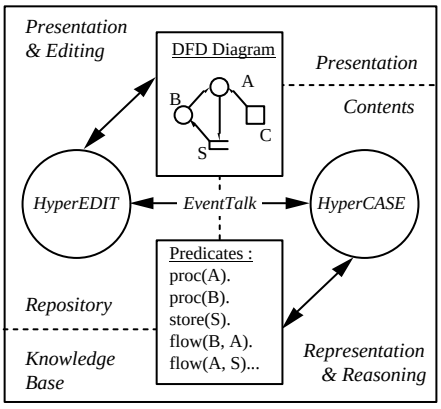

Figure 1 - HyperEDIT architecture

This technology revolves around diagramming techniques in describing software systems. Popular CASE tools (IEW, Auto-Mate, IDMS Architecture, etc.) utilise a limited suite of diagram editors (DFDs, ERs etc) in order to define the functionality of systems pictorially [2]. As consistent with CASE development technology, CASE tools attempt to provide integrated and complete development facilities for the production of software systems, subsequently locking software shops into closed architecture, single vendor CASE environments.

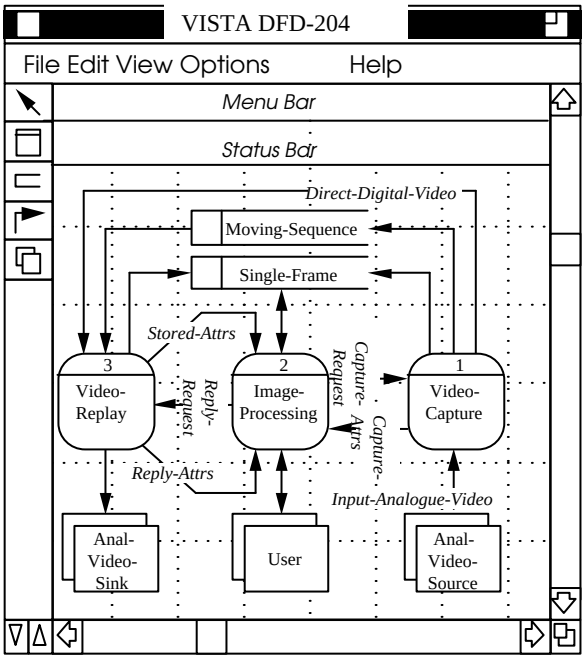

Figure 2 - A sample data flow diagram

HyperEDIT, a graphical editor construction tool, was designed specifically to counteract some of the above-mentioned problems (Figure 1). It may be used as either a suite of stand-alone diagram editors (see Figures 2, 3, 4) or it can be interfaced via EventTALK protocol to some other software products, e.g. HyperCASE [3]. As many other graphics editors or GUIs (e.g. MacDraw, CorelDraw, or FrameMaker), HyperEDIT incorporates windows, menus, object-oriented graphics, and mouse control. Unlike other tools, its customisation is fully interactive and totally user-driven. There is no need for HyperEDIT or the application re-programming, recompilation, nor re-linking. Such extendibility is achieved by the availability of a HyperEDIT meta-editor in which users can de-fine tailor-made editors, their window and page layout, the type and look of editable graphical objects, finally the editor buttons, controls, menus and their behaviour.

Once the editor is defined, it may be readily used to create custom-made diagrams and charts. In all cases, HyperEDIT is responsible for the diagram visual presentation, i.e. its *presentation* (e.g. diagram components shape and size, colour and position, etc.). However, any user-instigated alteration of previously designated object attributes contributing to the diagram meaning, i.e. its *contents*

(e.g. diagram components name, interconnection, value attributes, etc.), is conveyed to the controlling program via EventTALK (Cf. Figure 1).

Architecture

Three major sub-systems may be identified in the HyperEDIT architecture, ISDUIMS, SAT and EventTALK. Their brief description follows. (For more information on each component see [3])

ISDUIMS (ISD User Interface Management System)

The core of HyperEDIT presentation layer consists of a number of text and graphic primitives. The primitives are of sufficiently high level to facilitate functional expression, ease of use, and flexibility in the creation of windows, dialogue boxes, menus, palettes, buttons, text and graphics, all to be mouse and keyboard controlled. Although the current prototype is being developed under OSF/Motif [4] in X [5] on Unix, in future, through the use of window-independent development toolkit (e.g. OIT from Neuron Data for X, MacOS, OS/2 or Microsoft Windows), HyperEDIT will be available on a number of hardware and operating system platforms, i.e. Unix X-Windows, VMS DEC-windows, Microsoft Windows, MacOS, OS/2.

SAT (System Analysis Tools)

HyperEDIT allows its users to define variety of diagramming tools. Such a customisation process may require quite a lengthy interaction with the system, i.e. specification of all graphical objects, their presentation and contents, interaction with EventTALK protocol, definition of menus, buttons, etc. However, this interaction is normally graphical manipulation with the use of the pointing device (mouse, light pen etc.) and therefore requires minimal coding effort. (cf. CIRS). This interaction is

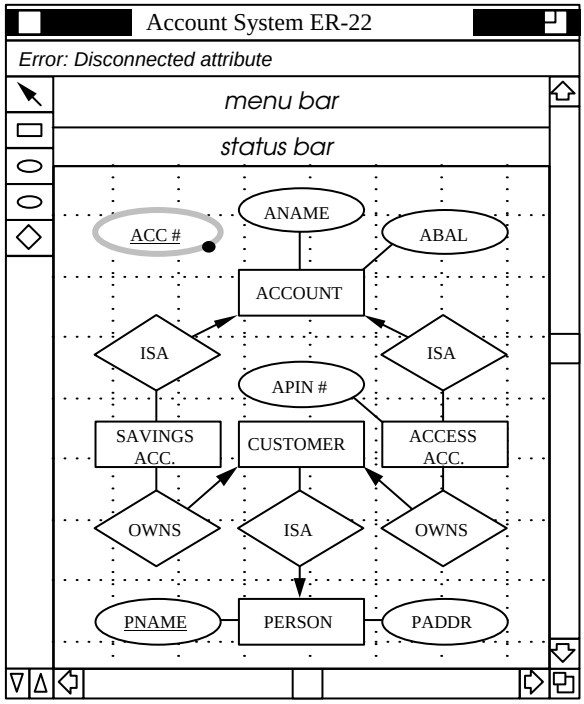

Figure 3 - A sample entity-relationship diagram

described in more detail in Section 2.

Therefore, to assist with quick acceptance and efficient cross-over to HyperEDIT, and taking into consideration the potential applications of HyperEDIT (i.e. software documentation, CASE, etc.), the system is equipped with a number of standard text and graphics editors, which could be used in the construction of system analysis and design documents, e.g. data flow diagrams, entity-relationship diagrams, HIPO charts, Petri networks, state-transition diagrams, flowcharts or Nassi-Shneiderman diagrams, etc. (Cf. Fig. 2, 3, 4.)

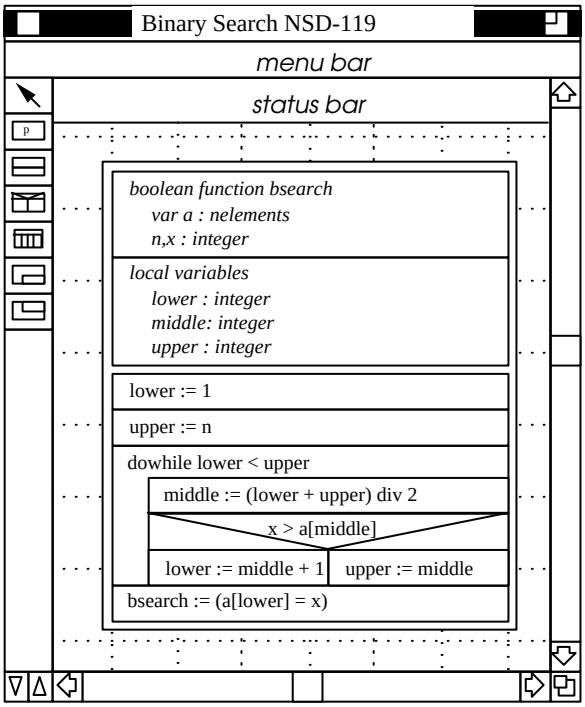

Figure 4 - Sample Nassi-Shneiderman diagram

EventTALK

HyperEDIT allows full control over text and graphics editors from some other user program via a specially devised communication protocol - EventTALK. The main objective of EventTALK is to advise the controlling program of all user-instigated changes to the document contents associated with the creation, deletion and editing of its components (e.g. creation or deletion of entities, relationships and attributes in E-R diagrams), so that it could perform validation of user actions. After the transaction is completed (e.g. deletion of E-R entity), additional changes to the document may be triggered by the user program (e.g. deletion of all connected relationships and attributes, issuing an error message, or undoing user actions), the reverse flow of EventTALK commands may also occur to reflect the visual representation of the document. The change to the document form (e.g. repositioning or resizing of graphical objects) is not communicated to the controlling programs. EventTALK communication may also allow for storage and loading of diagram components in and from remote databases (rather than in local document repository), opening and closing

documents in response to user actions (e.g. in hypertext browsing), etc. In addition, analysis of the EventTALK logs will aid the implementation of a user-interface tracking system, which could enable the system accounting, debugging or recording/replying of captured transactions for the demonstration purposes.

2. Diagram Editors

The purpose of all of the HyperEDIT diagram editors (SAT) is to provide HyperEDIT users with a GUI facilitating the creation of documents (esp. system design diagrams) composed of a set of application-specific objects. Regardless of the application and the inherent objects type, HyperEDIT editors provide a consistent page and window layout, similar procedures in object creation and editing, uniform mouse and keyboard control, etc. This section describes the details of the HyperEDIT diagram editor user interface.

Window Layout

Each of the user-defined or pre-defined SAT editors displays and manipulates its diagrams (e.g. Fig. 2, 3, 4) in windows of a standard layout dependent only on the features of a proprietary presentation manager, e.g. a Macintosh implementation will equip its windows with a named title bar, scroll bars, close, zoom, magnify/reduce and size controls [6] (cf. Fig. 5); in other environments, e.g. X with

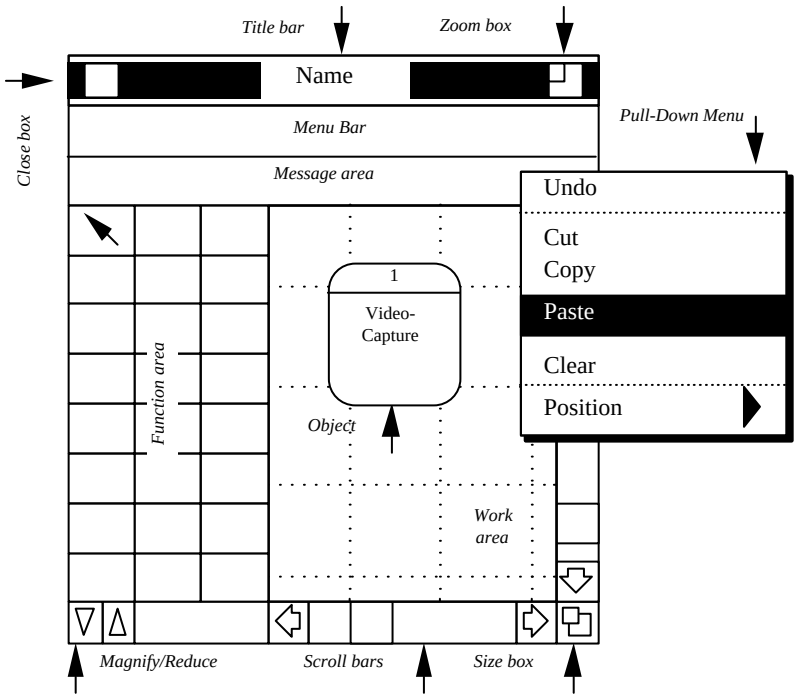

Figure 5 - HyperEDIT window layout

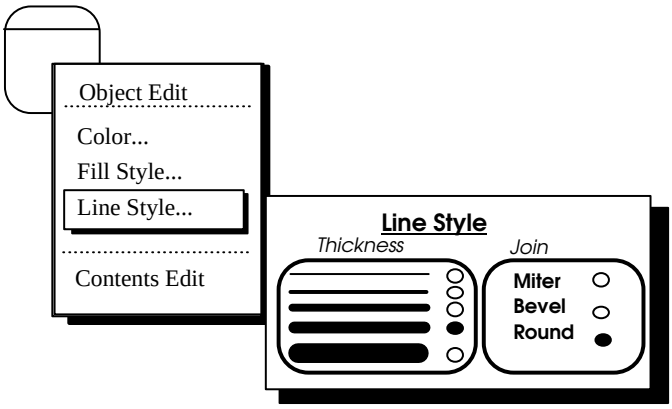

Figure 6 - Object menus

OSF/Motif, or Microsoft Windows, the same functions may be provided by some other means, e.g. dialogue boxes, menu items, etc. Irrespective of windows native look and their characteristic controls, their contents is split into four major panes :- menu, message, function and work areas.

Object Creation

The function area consists of a set of iconic buttons each representing the type of a graphical object that may be added to a HyperEDIT diagram. Once the function is selected, by clicking and highlighting, it traps the mouse clicks, in response to which a new document object is created, sized and subsequently positioned within the work area.

Object Editing

All document objects drawn within the work area may be selected and, if the mode of operation permits, deleted or modified. The modification may consist in editing object text fields, its re-positioning and re-sizing, changing its colour and fill pattern, varying the text font or style, etc. All of the functions applicable to the selected object are listed in a pop-up menu associated with its generic type (e.g. Fig. 6). Having selected a number of objects of different types, the common set of functions will only be displayed in a pop-up menu, e.g. color, fill pattern or re-position.

Certain object attributes may be modified not only via menus but are also subject to direct mouse control, e.g. object re-positioning by dragging, re-sizing by handle manipulation, destroying by pressing the delete key, etc. The presence and the type of object handles indicates the kind of functions that may be applied to it (e.g. in Fig. 7, a - an object can be moved and resized in all

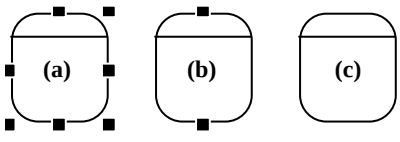

Figure 7 - Object handles.

directions, b - moved anywhere but resized in vertical plane only, c - an object can be moved but not resized).

Background Attributes

It should be noted that the diagram background is also treated as an object itself, thus the method of setting its visual attributes is identical to that of any other object belonging to the diagram. The background is automatically selected when no other object is highlighted, its attributes include page size, rulers, grid, default font, its size and style, colour, help level, etc.

Utility Functions

Administrative and utility functions, such as saving and loading diagrams, requesting help, setting options etc., may be accessed from the menu bar in the form of pull-down menus. Common functions, such as cut, paste etc., can also be accessed via the menu bar.

Messages

Depending on the help level, users may be given instructions in a message area at each step of their processing, alternatively warnings, error and system messages are displayed as well.

3. Presentation and Contents Editors

HyperEDIT permits software developers to design their own, custom-built diagram editors, capable of creating and modifying documents consisting of objects appropriate to user-specific application (cf. figures 2, 3, and 4). The process of new editor specification requires a detailed definition of all objects that are to be manipulated by the editor.

Presentation Editor

The graphical attributes of graphical objects, i.e. composition, shape, colour, patterns, line thickness, etc., can be defined interactively with a *presentation editor*, being itself a simple graphical editor (Fig. 8). As the shape of a new object type is drawn by the user with one of the form editor tools, its attributes are captured and stored in an intermediate form accessible by any diagram editor declared to use objects of this specific type.

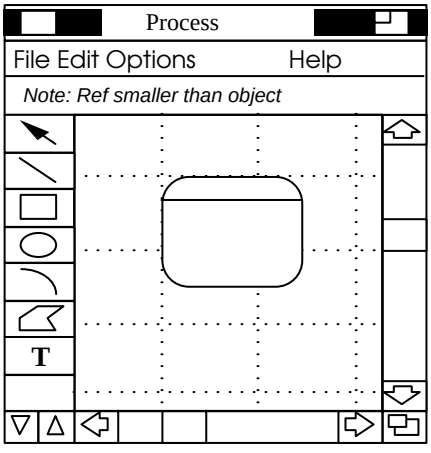

Figure 8 - Presentation editor

Contents Editor

Once the visual attributes are defined, the *contents editor* may be invoked to add other, non-graphical, object characteristics, e.g. attribute protection, object constraints, EventTALK flags, etc (Fig. 9).

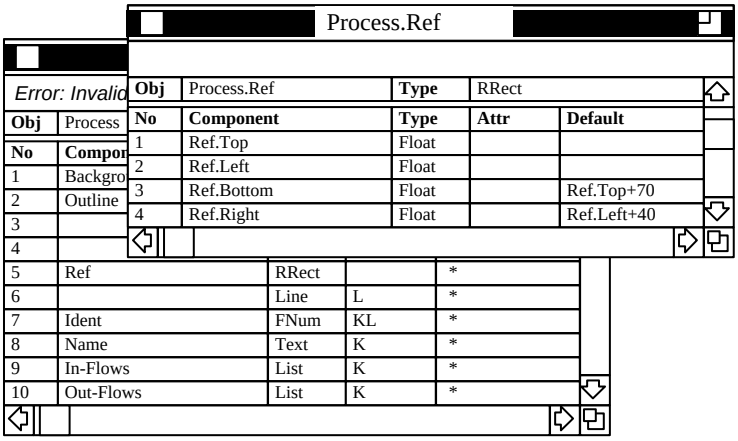

Figure 9 - Contents editor

Object Types

The main aim of both form and contents editors is to define new, generic types of graphical objects or classes of objects. In general a *type* (or *object class*) defines a collection of objects manipulable by HyperEDIT and sharing similar visual, representational, and functional attributes [7]. Types may be constructed either by another type specialisation (i.e. specifying and constraining of its attributes), or by composition of other types (i.e. specifying its sub-components).

All object types form a hierarchy of object definitions. Each type also defines a number of associated actions that may be applied to its objects. E.g. a quadrangle is a composition of four lines and the fill colour of its area, a rectangle is its sub-type by appropriate constraining of lines coordinates, black and red rectangles are again sub-types of a rectangle by attribute value specification. Note that the composition of lines and the fill colour to define a quadrangle requires addition of a specialised function capable of filling the quadrangle shape area which is not present in the set of the type sub-components. In database terminology, an object type corresponds to the file or table.

Primitive Objects

A set of *primitives* (or *primitive objects*), i.e. functionally non-decomposable, types are needed to describe all graphical objects. HyperEDIT engages the standard set of geometrical and visual primitives of available graphics toolkits, i.e. points, lines, rectangles, ovals, arcs, polygons, text, patterns, colour, line thickness, fonts, their sizes and styles, etc. In databases, an object primitive corresponds to the standard field type.

Object Instances

An *object* (or *object instance*) is a data structure representing an instance of a specific type. Graphical objects are directly manipulable by HyperEDIT users on the screen. All known object attributes are filled with specific values conforming to the type constraints. In database terminology, an object corresponds to the file record.

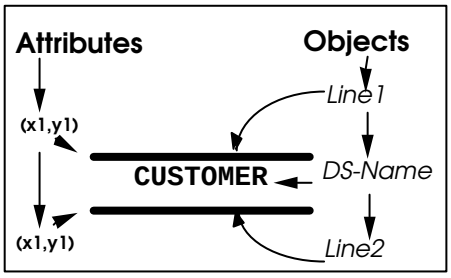

Figure 11. Data Store - Yourdon/De Marco

Object Attributes

Object *attributes* define its characteristics. Attributes are always defined in types but are instantiated with specific values at the object level. In the graphical domain they specify object size, colour, coordinates or entire sub-components.

Attribute Aspects

Each attribute may have a number of different aspects, one of which is the attribute value, others constrain the object values, set their defaults, specify whether the attribute value may be changed at object level, or whether the attribute is of interest to the EventTALK protocol.

Attribute Constraints

Constraints are placed on the attributes of an object in order to specify the behaviour of that object. A C-type grammar is used to define an attributes relationship to the object as a whole.

For example, a data flow diagram symbol, the data store (Fig. 11) could posses the following attribute constraints:

Object.Attribute		Aspects/Constraints
Line1.movable	**=**	**True**
Line1.resize	**=**	**False**
Line2.resize	**=**	**False**
Line2.y1	**=**	**Line1.y1+20**
DS-Name.resize	**=**	**False**
DS-Name.x1	**=**	**Line1.x1+5**
DS-Name.y1	**=**	**Line1.y1+10**

Object Manipulation

The attributes of the objects can be manipulated only according to the corresponding constraints for the object type or specific object instance. For example the above definition allows the user to select and drag Line1 to another position; the constraints will ensure that Line2 and DS-Name are re-positioned accordingly. However, since none of the constituent objects of the data store can

be resized, the data store itself cannot be resized. In this way the relationships between constituent components define the behaviour of the object as a whole (Fig. 12).

Figure 12. Data store presentation after Constraints

4. Meta Editor

The HyperEdit Meta Editor is in itself a graphic editor in which Systems Analysis Tools, specifically diagram editors are created. The Meta Editor allows the system designer to define the functionality of the editor, the objects which the designer can use to create the diagram and the 'look and feel' of the editor. In order to create a new diagram editor, the physical layout of the main functional areas (menu-bar, function area, message area etc.) must be considered. The contents editor defined the behaviour of objects with respect to each other in a class, however it is also necessary to define the inter-relationships between classes in order to implement the rules inherent in existing diagram types (DFD, ER, etc.) and for new HyperEdit created diagram types. For example, it is illogical for a flow of data to enter a process and none to be output in a Data Flow Diagram. The Class Inter-Relation Specifier (or CIRS, pronounced *curse*) specifies the inter-relationships between symbols.

Layout Definition

HyperEdit is being implemented using a data-driven approach, in that most of its properties and behaviour have been defined in resource and parameter files. The Layout Editor is a graphical front-end editor for these files. The Layout editor allows the user to customise the colours, fill patterns and most graphical attributes of the diagram editor, as well as defining the placement of the four main functional areas (Fig. 13).

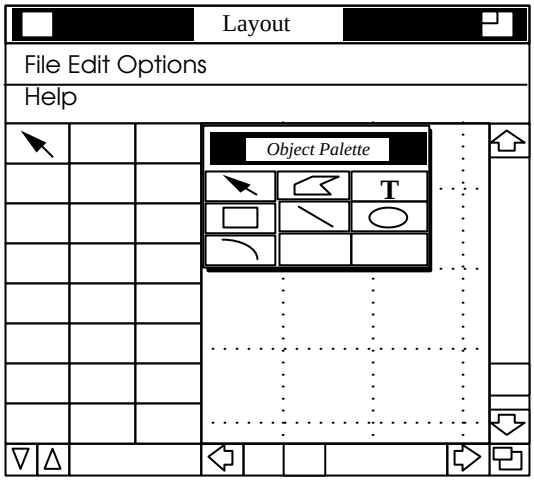

Figure 13. Layout Editor

Class Inter-relation Specifier

In order for the meta-editor to construct graphical document editors that display intelligence, a technique for specifying rules of association between the objects of the document

is required. The CIRS method incorporates Entity-Relationship diagram [8] type relationship specifying links (Fig. 14) as well as a "path" method to fully specify the relationships (Fig. 15). A path consists of objects and relationship links, and represents allowable path flows of a particular diagram type.

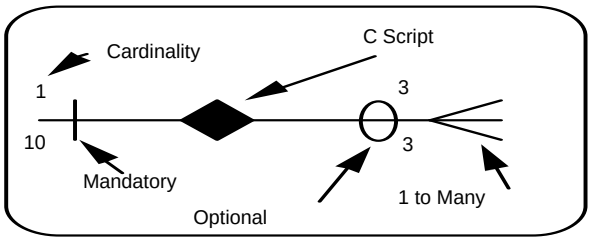

Figure 14. CIRS relationship Link

If any diagram rule, or peculiar relationship cannot be described by the CIRS notation, HyperEDIT provides the capability to attach C code scripts to relationship links. Figure 15 displays a simple allowable path specification between three data flow diagram symbols, a process, an output flow and a data store. The relationship specifies that a valid path can consist of a process connected by one or many output flows, each of which can then be attached to one data store. The mandatory symbols ensure that for an output flow to exist a process must exist. The optional symbol specifies that the output flow does not necessarily have to attach to a data store. Multiple paths can be declared in this way to specify the rules of object association for any diagram type through the CIRS editor. The CIRS editor is a standard HyperEDIT diagram editor, which allows users to pictorially define the relationships.

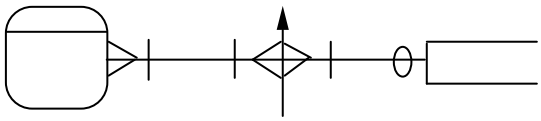

Figure 15. Simple Path

5. Summary

HyperEDIT is a software engineering document production tool. It is highly customisable; the user can create any type of diagram editor required to support the system development approach of the organisation. It accomplishes this through the use of three editors, the presentation, the contents and the meta editors. The presentation editor defines visual properties of objects that are used

within the diagramming tools. The contents editor defines non-graphical attributes of these objects, such as the relationships between constituent components, EventTalk messages etc. Finally the meta-editor defines the characteristics of the new editor, both in terms of physical appearance and operation.

The HyperEdit system has been designed to operate in stand-alone or as a graphical front-end for some other user system, such as HyperCase, through the use of the high level communications protocol EventTalk. Its open architecture does not restrict the user organisation as to the choice or vendor of tools, and integration into user environments is simple through appropriate customisation.

As the graphical user interface to HyperCase, HyperEDIT provides the necessary button mechanisms for a hypertext system and through EventTalk it supports hypertext navigational requirements. In order to provide an intelligent front-end to a suite of intelligent tools, HyperEDIT was designed with ease of use, customisation, portability, and connectivity as major design goals. Its intelligence springs from the constraints placed on individual objects through the contents editor and between objects through the CIRS editor. It is envisaged that HyperEDIT coupled with HyperCASE will enhance computer professionals' efficiency and productivity in the early stages of the software development lifecycle and help them deal with the ever-increasing complexity of today's software systems.

Bibliography

1. Rock-Evans, R., Engelien, B. 1989. Analysis Techniques for CASE: a detailed Evaluation. Ovum London.

2. Rock-Evans, R. 1989. CASE Analyst Workbenches: a Detailed Product Evaluation. Ovum London

3. Cybulski, J., Reed, K. 1991 "HyperCASE: A HyperText Based Software Engineering Environment". AAITP Internal Report.

4. Johnson, F. E., Reichard, K. 1991. "Power Programming: Motif" MIS Press, Portland Oregon.

5. Scheifler, R. W., Gettys, J. 1990 "X Window System" Digital Press

6. Apple Computer Inc. 1989 "Inside Macintosh Vol. 1" Addison-Wesley Inc. Reading Massachusetts

7. Vlissides, J. M., Linton, M. 1988 "Applying Object-Oriented Design to Structured Graphics" Usenix C++ Conference Proceedings.

8. Chen, P. 1976. The Entity Relationship Model - toward a Unified View of Data. ACM Transactions on Database Systems. March pp 9-36