Taskmaster:
An Interactive, Graphical Environment for Task Specification, Execution and Monitoring

James D. Arthur and K. S Raghu

TR 89-15

Taskmaster: An Interactive, Graphical Environment for Task Specification, Execution and Monitoring

James D. Arthur and *K.S. Raghu*

Department of Computer Sciences
Virginia Tech
Blacksburg, Virginia 24061
(703) 961-7538

ABSTRACT

Taskmaster is an interactive environment that employs a unique blend of graphic technologies and iconic images to support user task specification. In this environment, problem solving is based on the selection, specification, and composition of tools that correspond to natural sets of ordered operations. The Taskmaster environment is novel in that it:

- provides an interactive, visual-based approach to user task specification,
- encourages and supports task specification and refinement processes from *both* the top-down and bottom-up perspectives, and
- enables one to specify parallel tasks in a natural and convenient manner.

To "program" a given task within the Taskmaster environment, one decomposes it into an ordered set of conceptually simple, high-level operations, and then combines (composes) a corresponding *network* of software tools that implements these operations. Execution of the specified network provides a task solution. Major system components supporting user task specification include a *network editor*, a *tools database* and a *network execution monitor.*

CR Categories and Subject Descriptors: D.2.6 [**Software Engineering**]: Interactive Programming Environments; H.1.2 [**User/Machine Interaction**]: Human information Processing

General Terms: Interface Abstractions, Top-Down and Bottom-up Task Specification, Partitioned Menu Networks, Graphical Networks

Additional Keywords: Nodes, Arcs, Communication Paths, Cutsets

Taskmaster:

An Interactive, Graphical Environment for Task Specification, Execution and Monitoring

James D. Arthur and *K.S. Raghu*

Department of Computer Science

Virginia Tech

Blacksburg, VA 24061

1.0 Introduction

Over the past decade, several trends have led to new attitudes about software development, and correspondingly, user task specification. First, the proliferation of personal workstations with extensive graphics capabilities has put local computing power in the hands of many people and has created a demand for high quality, user-friendly interfaces. Second, software development tools have evolved from an *ad hoc* collection of independent programs to integrated sets of complementary tools forming a basis for user support *environments*, e.g. Interlisp [TEIW81], Smalltalk [TESL81] and PECAN [REIS84]. Several of these environments, such as PICT [GLIE84], PegaSys [MORM85] and DMS [EHRR86], exploit graphical interfaces to enhance user productivity. A third trend is the specification of task solutions through software composition techniques rather than line-at-a-time coding [ARTJ87]. This third trend is particularly attractive because it help reduce coding errors and implementation time as well as facilitates software reusability. What has been lacking, however, is an environment that embraces all three of these trends, that is, a visual-based environment for high-level, task specification. In recognition of this

void, the authors have pursued a research direction leading to the development of *Taskmaster: An Interactive, Graphical Environment for Task Specification, Execution, and Monitoring.*

Taskmaster employs a unique blend of graphic technologies and interaction formats to visually support the specification, instantiation and monitoring of physical tasks. Within the Taskmaster environment, one begins a task specification by conceptually breaking the task into a partially ordered collection of high-level operations. The decomposition is then expressed as a graphical network where nodes represent high-level operations (supported by underlying software tools) and arcs represent directed communication paths between the nodes. Instantiation of that network provides a solution to the correspondingly specified task. In support of this visual approach to "programming", Taskmaster provides a flexible, user-directed dialogue format based on *complementary* interaction mechanisms. In particular, functional abstractions, supported through visually-oriented icons and primitives, provide an integrated *top-down* and *bottom-up* task specification interface.

The remainder of this paper focuses on concepts intrinsic to and issues underlying the development of the Taskmaster environment. Because the operational aspects of the environment play such a crucial role in our discussion, an overview of the system is presented first. Included in that presentation is a description of the environment's major components which visually and textually support user task specification. Section 3 follows and presents a discussion of abstractions used in support of top-down task specification. The discussion includes a description of (a) *partitioned menu networks* supporting *multi-level, menu-based interaction,* and (b) the *expand node* operation. Section 4 describes *composite tool abstractions* and discusses its relevance to bottom-up, user task specification. Included in this discussion is an overview of *tool-composite* operations. Finally, Section 5 provides a summary of the paper and briefly discusses the current status of the Taskmaster environment.

2.0 The Taskmaster Environment: An Overview

The Taskmaster environment has been a product of evolution. Its initial predecessor, OMNI [ARTJ87], was *textually* oriented and supported interactive user task specification based on a "loose" composition of program filters. Taskmaster's immediate predecessor, GETS [ARTJ88], exploited *graphics-based* task specification but, like OMNI, was still restricted to *rigid* specification constraints enforced by menu-based interaction. Learning from our experiences with OMNI and GETS, the Taskmaster environment has been purposely designed to support user task specification from a graphics-oriented perspective while utilizing abstraction mechanisms that minimize the interaction rigidity inherent to menu-based systems.

2.1 A Task Specification Example

The high-level operations identified during task specification are supported through a collection of software tools present in a tools database. A *tool,* as used in this paper, refers to a filter program (*a la* UNIX[1] *sort*) which performs a single operation with minor variations. Each tool can have multiple input and output ports through which it communicates with other tools. To "program" a given task within the Taskmaster environment, one decomposes the task into a partially ordered set of conceptually simple, high-level subtasks (or operations), and then composes a corresponding network of software tools that implement those subtasks. This decomposition/composition process is supported through and depicted as a graphical network in which nodes correspond to subtasks and arcs represent directed data paths between the nodes. For example, Figure 1 illustrates a task specification that:

- retrieves the contents of a file,
- creates two copies of the file contents,

[1] Unix is a trademark of AT&T.

- selects records *in parallel* based on independent selection criteria,
- merges the records selected by the parallel selections,
- sorts the selected records, and
- stores the selected and sorted records in a file.

The resulting network topology captures, from both a visual and operational perspective, the set and sequence of operations needed to compute a solution to the user task specification.

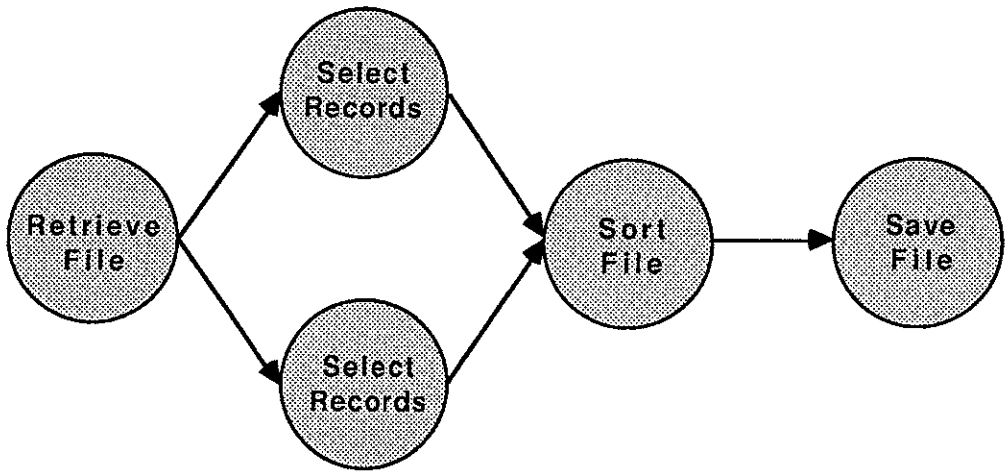

Figure 1

A High-Level, Task Specification Network for Sorting and Saving Selected Records

During the task specification process leading to the network illustrated in Figure 1, the user could have chosen to specify the task from (a) the top-down perspective by creating a single node representing the entire task and then expanding it into a multiple node network representing various levels of task specification refinement, (b) the bottom-up approach by creating an initial network topology where each node represents a distinct operation intrinsic to the the proper resolution of the task, or (3) a combination of both the top-down and bottom-up approach. The authors note that

although the topology shown in Figure 1 is in its final format, the operations associated with each node are still specified using their generic name. As illustrated in Figure 4, further interaction is still required to bind each node to a specific tool.

2.2 The Major Components of the Taskmaster Environment

The Taskmaster environment is an integrated user support environment that exploits visual programming concepts, tool composition, and structured data flow. It is composed of three major cooperating components:

- the Network Editor,
- the Network Execution Monitor, and
- the Tools Database.

The Network Editor provides a graphical interface for constructing task networks. It guides the user through the task specification process by supporting top-down and bottom-up interaction formats. Once a task is fully specified, the corresponding network is ready for execution. A network representation is forwarded to the Execution Monitor for instantiation and monitoring. The Tools Database plays a supportive role in that it provides access to all the information pertaining to the basic tool set. This information includes a detailed description of each tool present in the database, its interface structure and the dialogue for refining its function.

Physically, the environment is partitioned across two machines connected by a high speed communication link. The Network Editor resides on a VAXstation[2] I running MicroVMS 4.2 (local workstation). The Execution Monitor resides on a VAX 11/785 running Ultrix-32 (host

[2] VAXstation, VAX, Ultrix and MicroVMS are all trademarks of the Digital Equipment Corporation.

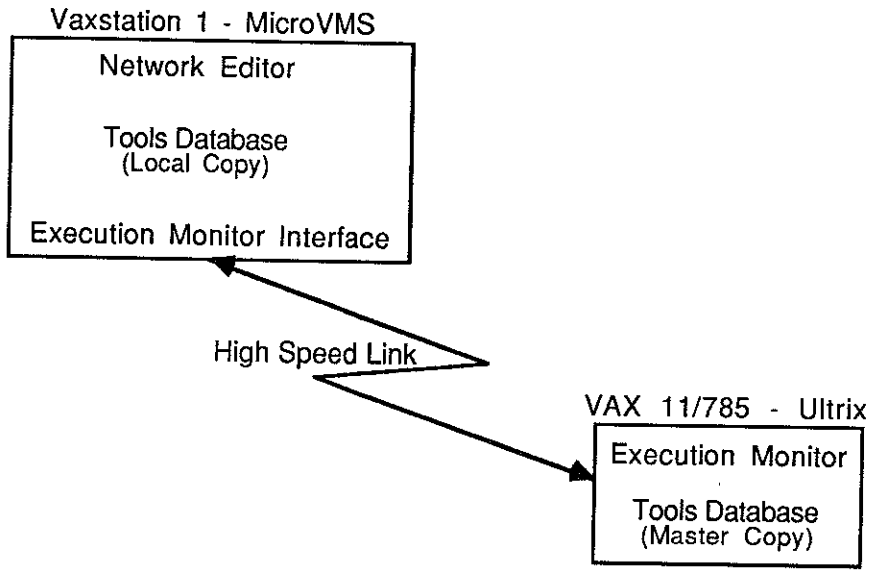

Figure 2

Taskmaster System Configuration

computer). The Tools Database resides on the host machine but gets copied over to the local workstation on every update. Although the current configuration has a single local workstation, we envision a set of local workstations all connected to the host, in future. The overall system configuration is shown in Figure 2.

2.2.1 Taskmaster User Interface: The Network Editor

The Network Editor provides an interactive, graphical interface for constructing task specifications. Programming in the Taskmaster environment consists of transforming a conceptual task into a network whose nodes represent operations (tools) and whose arcs represent the communication path between the nodes. The Network Editor supports the specification process by providing editor primitives for building *generic* networks and for specifying nodes and arcs through menu-based interaction. For clarity, a generic network is viewed as a directed graph with *unspecified* nodes and arcs. Following the construction of a generic network, additional

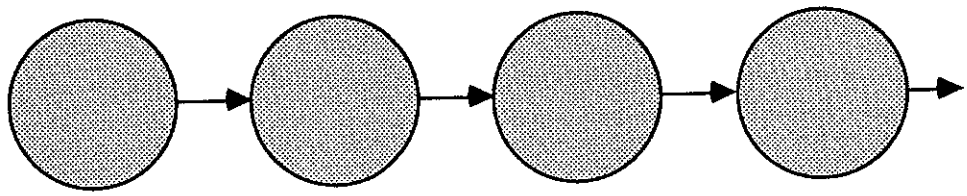

Figure 3

A Generic Network for Record Selection, Sorting and Saving

interaction leads to the "binding" of nodes to operations and arcs to tool-specific ports. Figure 3 illustrates one generic network from which the fully *specified* network in Figure 4 can be derived. Note, however, that the second node will require further refinement through an expansion process.

The Network Editor is primarily menu-driven and makes extensive use of logical windowing and mouse input. Similar to the Dialogue Management System [EHRR86] and PICT [GLIE84], the Network Editor incorporates many of the human engineering principles related to graphical user interface design. For example, ergonomic features include direct manipulation, visual feedback, user error recovery, choice confirmation, default selection, operational and representational consistency, pop-up menus and so forth. The Editor display consists of a large window detailing the topology of the network being edited and auxiliary pop-up windows for displaying

- menus,
- multiple views of nodes and arcs,
- user instructions and help messages,
- error messages,
- user confirmation requests, and
- textual information pertinent to tool and communication path specification.

The Network Editor provides access to many editing primitives, the majority of which support either network *construction*, network *specification*, or network *inquiry* operations. The network construction operations are used to create and operate on node and arc icons. When first created by the *create node* (*create arc*) operation, node (arc) icons serve as visual place-holders and have no initial semantic meaning with respect to the task being solved. *Pan* and *zoom* operations provide for the selective viewing of relatively complex networks. The *collapse* operation allows one to abstract a subnetwork performing some high-level operation into a single "super-node". The *explode* operation reverses the effect of a corresponding collapse operation. The *expand* operation provides a new network topology window in which to define a subnetwork associated with the node being expanded. Sections 3 and 4 presents a more detailed discussion of the expand, collapse and explode operations relative to specifying functional abstractions.

The network *specification* operations are used to specify the node and arc icons. Node icons are specified by attaching to them, a fully refined tool or pseudotool from the tools database. Arc icons are specified by making all the appropriate connections between the tools associated with the nodes connected by the arcs. Thus, an arc can be specified only after both incident nodes are fully specified, and subsequently, tool interfaces are known.

The network *inquiry* operations provide characteristic information based on the current specification status of nodes and arcs. The *view node* operation provides a detailed view of the tool attached to a specified node. Taskmaster also provides a textual view of each specified node containing the description of the associated tool, its attributes and its input and output ports. The *view communication path* operation provides a detailed view of the interface between the tools connected by an arc.

Additional operations provide for various "backup" and "restore" capabilities. At anytime during a editing session, the current state of the network can be saved to disk or a previously saved

network can be restored from disk using the *save network* and the *restore network* operations respectively. The *undo* operation supports error recovery by undoing the effect of the most recent topology modifying operation.

Finally, after a network is created and completely specified, the *execute network* operation can be used to send the network to the Execution Monitor for instantiation. Upon selection of this operation, the Network Editor performs consistency checks and network validation, and then sends an internal representation of the network over a high-speed link to the remote hosted Monitor for instantiation.

2.2.2 Taskmaster Executive - the Execution Monitor

Problem solving in the Taskmaster environment consists of (1) specifying the problem and (2) computing the solution. The Network Execution Monitor supports the second stage of this process by performing the following functions:

- reading the task network representation forwarded by the Network Editor,
- validating the network,
- spawning computational processes based on the network topology, and
- monitoring the network execution.

Before initiating execution of the network, the Monitor first performs a modified breadth-first traversal (BFS) of the network checking for network connectivity and data path consistency. After confirming network consistency, the Monitor instantiates the network by spawning a process for each node in the network, allocating a UNIX "pipe" for each arc and connecting those pipes to the appropriate nodes. The node instantiation is also performed in the BFS traversal order in an attempt to satisfy certain interprocess communication constraints imposed by the operating system.

The network execution, however, is independent of the instantiation order because it is based solely on *data flow*. The Execution Monitor sends status messages back to the Network Editor indicating the instantiation, execution and termination of each node-associated process.

2.2.3 Taskmaster Knowledge Base - the Tools Database

The third component of the Taskmaster environment is the Tools Database. In the Taskmaster environment, the Tools Database plays a major role in isolating and encapsulating all application-specific information, and presenting it in a generic form to the other two components of the environment. This approach has significant advantages in that:

- defining a new application domain requires only that the Tools Database be redefined accordingly, and
- integrating into the new Database into the Taskmaster environment is an operation that is transparent to the rest of the system.

More specifically, the Tools Database contains information about all the tools available in the environment. This information includes tool communication requirements, tool arguments and complete textual descriptions of each tool and its input and output ports. The Tools Database also contains all the information supporting the multi-level, menu-based dialogue process for node specification. The Network Editor directly uses this information to drive node specification. Effectively, the specification process can be viewed as a finite state machine driven by the Tools Database menu dialogue "table" [RABM59].

3.0 Abstractions in Support of Top-Down Task Specification

To specify a task within the Taskmaster environment, the user first defines a generic network reflecting an conceptual ordering of one or more high-level operations. The initial network can be a single node representing the entire task, or a network of nodes representing a task specification overview. From this initial configuration, top-down specification can be employed to refine the network. Top-down task specification is the successive decomposition of a high-level task into lower level subtasks until the lowest level subtasks are directly identifiable with available tools or pseudotools defined in the tools database. In the Taskmaster environment, top-down task specification is supported through two distinct interaction formats, each embracing different decomposition philosophies and employing distinct abstraction mechanisms. The first approach exploits *partitioned* menu networks through a *multi-level,* menu-based interface [ARTJ85]. As with any menu-based system, the specification/decomposition paths are *predefined.* The second approach, however, employs node *expansion* activities and supports *user-directed* specification/decomposition. As discussed below, both approaches encourage top-down, task specification. Multi-level, menu-based interaction assumes that each node being specified represents one operation and will be attached to a single tool. Node expansion, on the other hand, assumes that the selected node is to be "expanded" into a subnetwork of nodes, each representing distinct operations.

3.1 Top-Down Specification through Multi-Level, Menu-Based Interaction

Multi-level, menu-based interaction assumes that the node being specified is to be directly bound to a primitive tool defined in the tools database. As with any menu-based dialogue format, the interaction process is restricted to predefined sets of refinement paths that correspond to the the underlying menu network hierarchy. The novelty of this approach is not the menu-based interaction *per se*, but the specification sequence induced by a *partitioning* of the underlying menu

network. That is, the predefined menu network is partitioned into multiple levels, where each level (or layer) represents a refinement abstraction across the *entire* menu network. As illustrated below, the partitioning induces *interface layers* that permit the user to

- specify a task overview based on predefined high-level operations, and then
- successively refine that overview through subsequent menu-based interaction.

Although we choose to restrict the discussion of partitioned networks and the multi-level, menu-based interaction to systems that support user task specification, the concepts presented in this section are applicable to most general menu-driven systems and their corresponding application domains.

In specifying a task, the user first constructs a generic network that provides a framework for sequencing and specifying a particular set of operations. Through conventional menu-based interaction, for each node in the generic network the user selects the appropriate sequence of menu frame items that *identifies* the high-level operation, *binds* that node to a tool which implements the operation, and then *refines* the execution behavior of that tool. This scenario implies that a selected node is fully specified before another node is considered. For example, suppose that a user has access to a menu-driven, *file transformation* system and wants to retrieve a file, select certain records from a specified file, sort them, and then save them for later processing. First, the user selects the sequence of frame items whose corresponding actions solicits the name of the file to be *retrieved* and infers all associated physical attributes. Next the user chooses a sequence of frame items that indicates the *select-record* operation as well as the criteria for selecting the appropriate records. The user then chooses a sequence of menu items that leads to a description of the *sort* operation and all refinements that specify record format and the desired sort sequence. Finally, frame items are selected that denote the *file-save* operation and that define all characteristics relating to the destination file. Figure 4 illustrates one possible network to accomplish the above specified

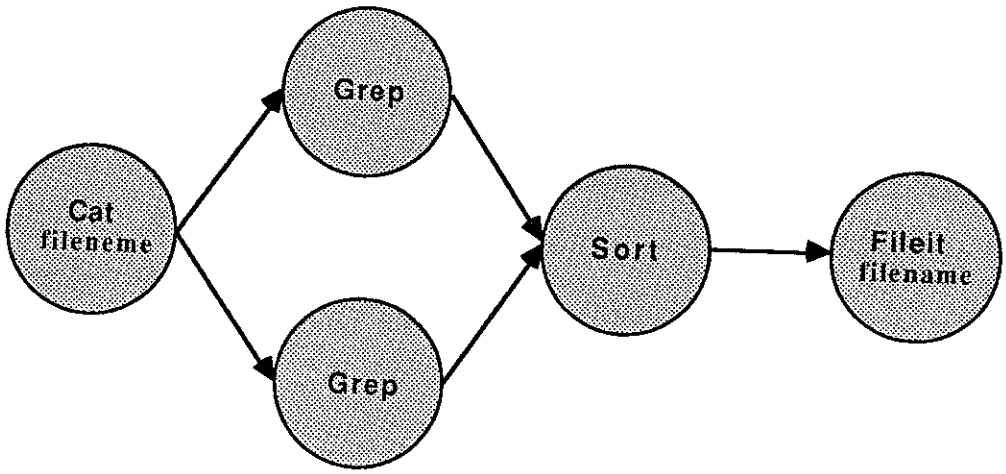

Figure 4

Fully Specified Network to Sort and Save Selected Records

task. In this fully specified network *cat* is the file retrieval tool, *grep* is the select tool, *sort* is the sort tool and *fileit* is file creation tool.

The problem with the specification approach described above is that the user is *forced* to select one node at a time and fully specify its operational details before moving to another node in the network. Such rigidity, enforced by conventional menu networks, tends to obscure the user's *overall* perception of the task solution. For complex tasks, forcing the user to contend with details before firmly establishing a task overview can have adverse, if not devastating repercussions.

In the Taskmaster environment, however, menu interaction is based on partitioned menu networks that support and encourage partial node specification through defined interface layers. Intuitively, an interface layer can be viewed as a horizontal "slice" through the menu network that delimits menu frames possessing a common level of specification refinement. For the above file transformation example, a (simplified) conventional menu network might look similar to the one illustrated in Figure 5. In the Taskmaster file transformation environment, however, Figure 6 shows the same menu network after partitioning. Note that the network defined at Hierarchical

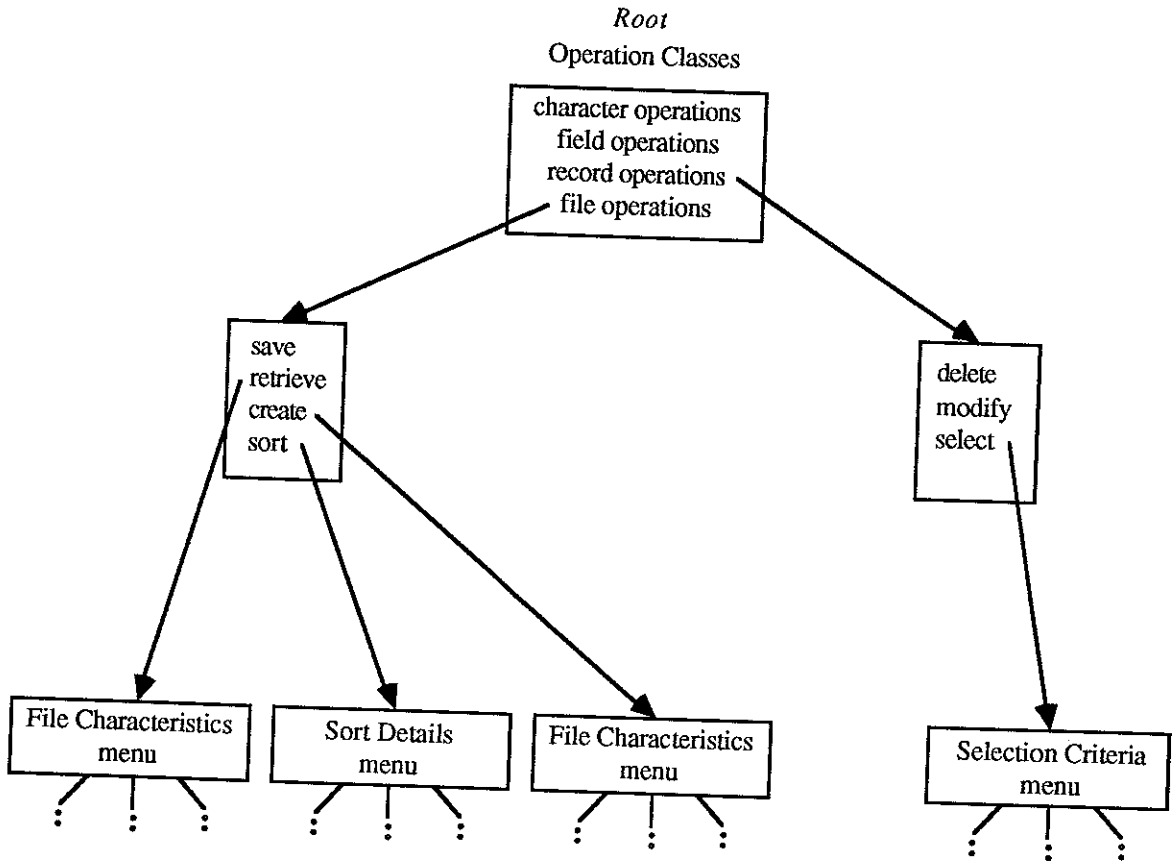

Figure 5

Conventional Menu Network for File Transformation

Level 1 "terminates" with the selection of a high-level operations. Hence, the user can traverse the menu network in a conventional manner, specify a task overview (without being encumbered by refinement details), and then *continue* with individual node refinement through interaction guided by the second-level menu networks. That is, the user first constructs a generic network (Figure 7a), specifies an overview by associating high-level operations with each node (Figure 7b), and then refines each high-level operation through continued menu interaction on the second hierarchical level. The final result is a fully specified network identical to the one shown in Figure 4. We emphasize that partitioned networks provide the *capability* for the user to specify an

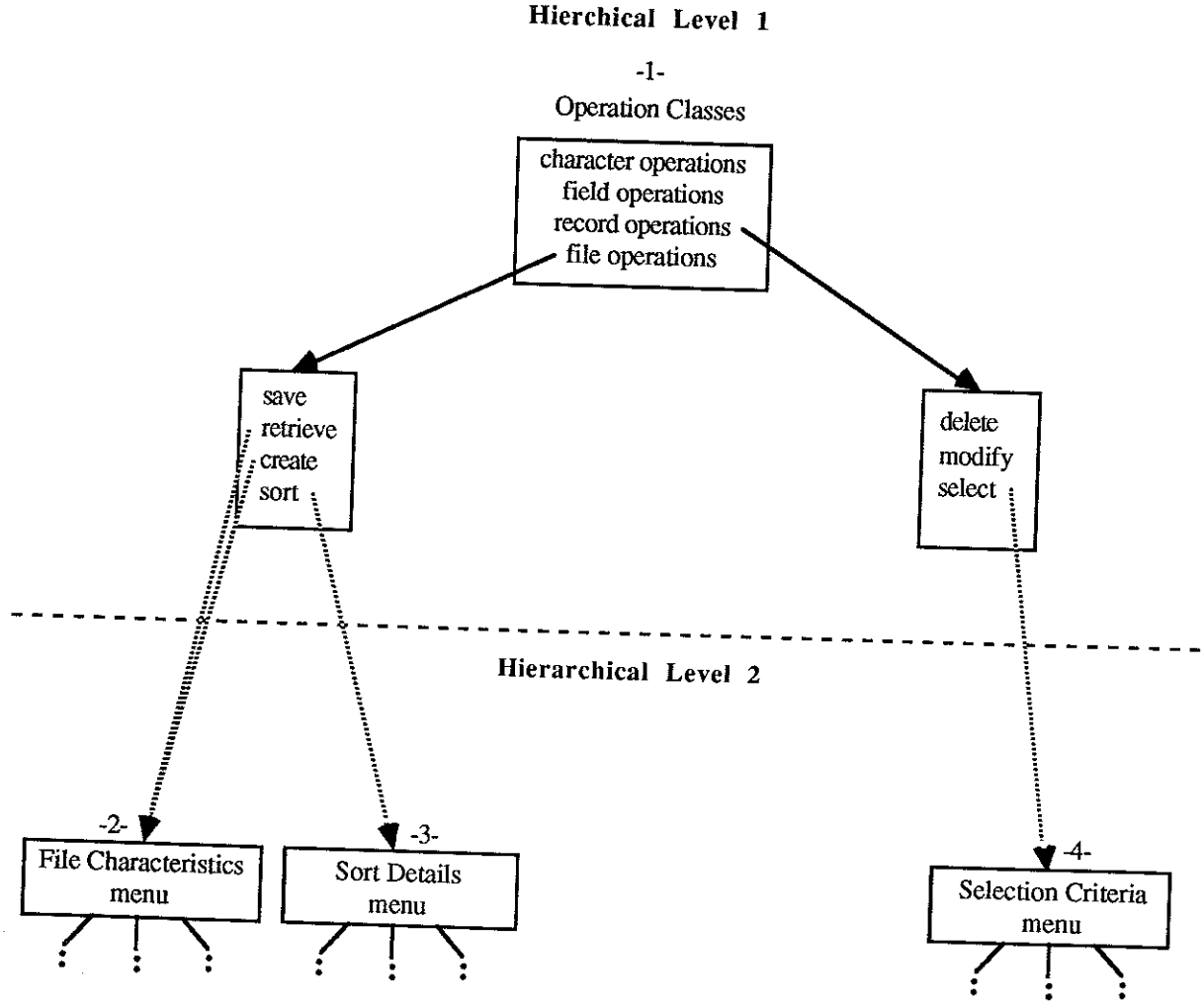

Figure 6

Partitioned Menu Network Supporting File Transformation

overview through the Taskmaster interface. The final choice remains with the *user* as to whether the specification sequence follows a "breadth-first" overview or the conventional "depth-first" orientation.

Although partitioned menu networks is a powerful mechanism for supporting multi-level, menu-based interaction, the user is still forced to follow a set of paths defined by the menu

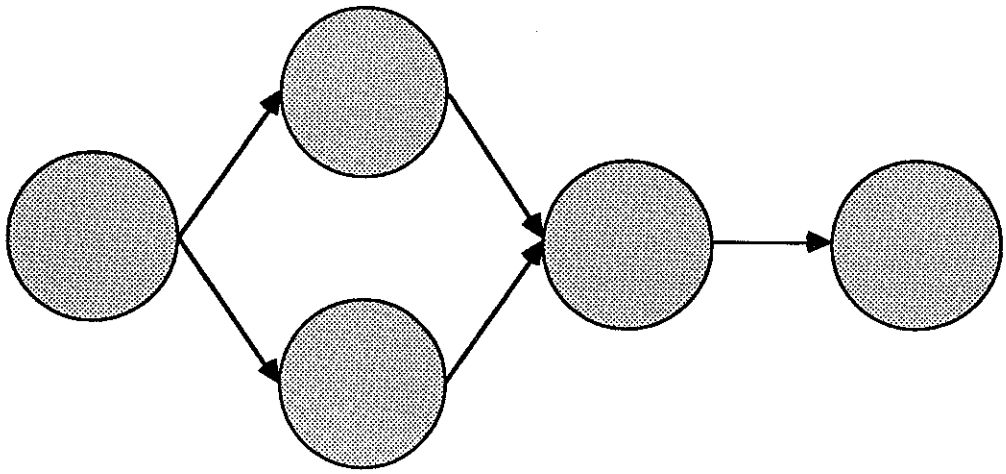

Figure 7a

Generic Network to Sort and Save Selected Records

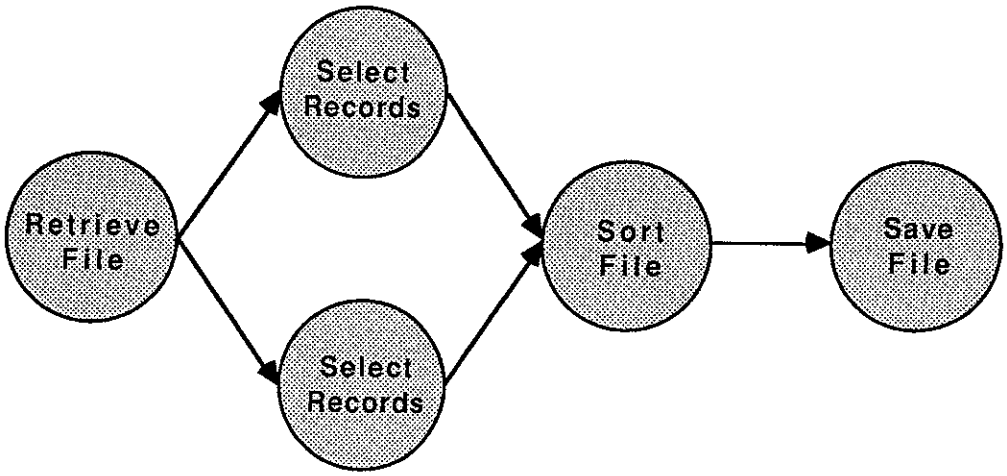

Figure 7b

Overview Network to Sort and Save Selected Records

(sub)networks. The next section describes an alternate top-down specification scenario that allows the *user* to control the specification process.

3.2 Top-Down Specification Through Node Expansion

Given a generic network topology, top-down task specification entails

- the selection of an unspecified node, and
- the binding of that node to a tool or pseudotool through an iterative refinement process.

As described in the previous section, one method for associating a tool with a selected network node is through menu-based interaction. With this approach the user is forced to follow a predefined set of paths leading to the selection of a tool in the database. To introduce more flexibility in the top-down specification process and to encourage user creativity, the Taskmaster environment provides a node *expansion* primitive that allows the *user* to direct the specification process. Intuitively, *expand node* "opens up" a single, unspecified network node and permits the user to fully specify a subnetwork within that node.

The node expansion operation provides the user with a separate "node expansion" window to construct the subnetwork to be integrated. This window is two-thirds the size of the network topology window but logically the same. The expand node operation is recursive to five levels and only limited there for controlling the complexity of the display. Thus, if a node in an expansion window is also selected for expansion, a new node expansion window appears, overlaying the previous one. Only one expansion window is active at any particular instant and the underlying inactive windows are clearly identified as such.

In effect, the user can specify multiple levels of abstraction reflecting his/her own perception of an operation or task, and have all levels appear as one node at the outermost level. Moreover, because all normal editing operations are supported by the expansion window, the user can choose the method by which each individual subnetwork node is subsequently specified.

4.0 Abstractions in Support of Bottom-Up Task Specification

Top-down task specification involves the successive decomposition of a task into lower level subtasks until the lowest level subtasks are directly identifiable with available tools in the tools database. In many instances top-down specification is most natural, e.g. when concentrating on the specification of a single network node. Within the framework of task specification, however, it is often convenient for the user to consider groups of nodes as a single abstraction supporting one *high-level* operation. Although not specifically stated, the expand node operation provides such a view but from a top-down perspective.

Bottom-up task specification involves successive abstractions of fully specified lower level subtasks into higher level subtasks. At the lowest level of abstraction a network is specified where each node is directly bound to a tool or pseudotool in the database. The specified network usually defines some low-level, yet not quite primitive, function. Reflecting a bottom-up specification strategy, this network is collapsed into a "super-node" and becomes a single node in a higher level network. This successive abstraction toward higher level functionalities culminates in a fully specified subnetwork that performs a specific *user defined* function.

The remainder of this section describes how successive abstraction is integrated into the Taskmaster environment. In particular, Section 4.1 describes a model defining the semantic framework associated with successive abstraction. Section 4.2 describes the editing primitives supporting abstraction from a user's perspective.

4.1 Abstraction based on Cutsets

Abstraction is itself an abstract term which has manifold meanings. Abstraction as used here means the hiding of unnecessary detail, or equivalently, showing only those aspects essential to solving a given problem. It is important to note that the criteria used in abstraction are dependent on the projected use of the abstracted object or the target environment. Abstraction is the best way to deal with complexity since it reduces the apparent complexity by the elimination of irrelevant detail. Of particular interest here, is the abstracting of a composite tool representing a single, high-level operation from a collection of "networked" tools. The following paragraph defines terms that will be helpful in relating our model of abstraction to bottom-up specification.

As described earlier, a *tool* is the basic entity in the tool composition paradigm and performs a single operation. Each tool has one or more *ports* with which it communicates with other tools via *links*. A composite tool or a *tool-composite* is a collection of tools grouped together forming a new tool, or *pseudotool*. A *cutset* of tools is a sub-network of tools delineated from the whole by a closed polygon. For example, Figure 8 shows a network where a closed polygon forms a cutset comprising the tools labelled C,D,E and F. Within the Taskmaster environment, cutsets visually and physically define pseudotools which, correspondingly, define high-level operations from a bottom-up perspective.

Any cutset can be considered to have two distinct contexts. The first is its internal context which includes only the internal links and all the tools comprising the cutset. The second is the external

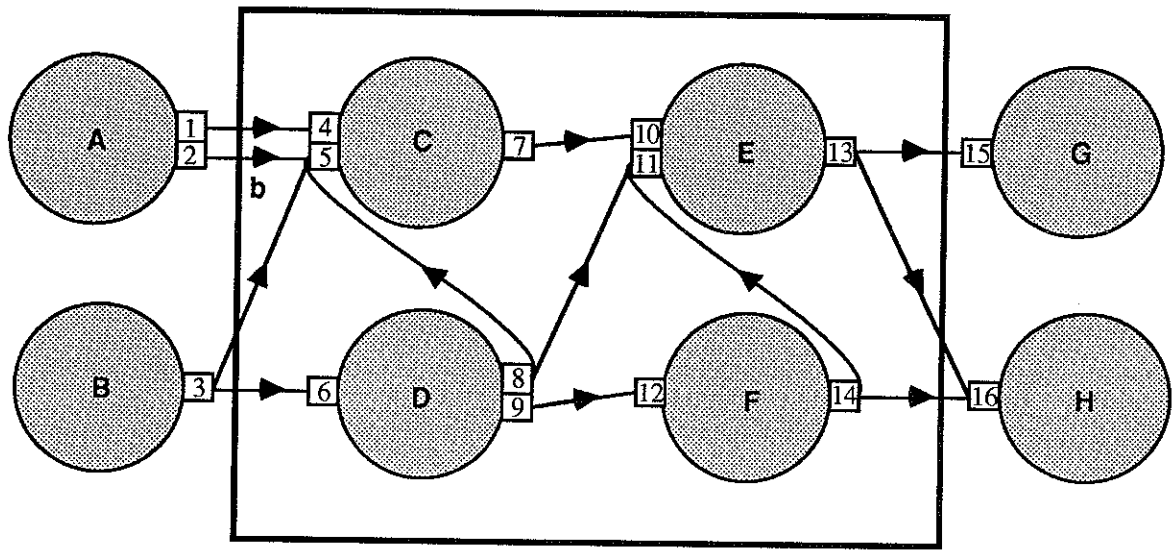

Figure 8

Example Network to Illustrate Cutset

context of a cutset which entails information about the cutset *vis-a-vis* the rest of the network. By virtue of identifying a cutset one automatically implies the preservation of its internal context. In terms of the external context, however, Taskmaster records only enough information to preserve the *functional* identity of the selected set of tools. This approach is particularly appealing because it promotes reusability at the functional level.

4.2 Editor Operations Supporting the Cutset Abstraction

The Network Editor supports four operations that enable the user to create and manipulate tool-composites (pseudotool), namely *save tool-composite, attach tool-composite, collapse* and *explode*. In particular,

- the *collapse* operation is used to define a *new* pseudotool *in-place,*

- the *attach tool-composite* operation enables the user to insert a pre-defined pseudotool (defined with the *save tool-composite* operation) into the currently defined network, and finally
- the *expand node* operation supports the definition of a complete sub-network relative to an existing node in the currently specified network.

The remainder of this section illustrates how of each of these primitives operate, except for expand node which is discussed in Section 3.2.

4.2.1 The Collapse and Explode Operations

The collapse operation creates an abstraction representing the collection of tools identified by the user. The resulting "super-node" is viewed as implementing some high-level operation. Collapse can be used recursively in the sense that it may be applied to a cutset which already contains collapsed pseudotools. The effect of the collapse operation is to redraw the network with the collapsed tool-composite being represented by a special "super-node" icon containing the user-supplied label. Figure 9a shows a cutset in the process of being collapsed. The *rubber band* polyline drawn with the mouse to delineate the cutset can also be observed in the figure. Figure 9b shows the network after the collapse operation is completed. The "super-node" icon with a brick-pattern ring represents the newly defined pseudotool named *Process File*.

During the collapse operation the Network Editor automatically constructs pseudotool port descriptions from the correspondingly encapsulated tools. The pseudotool name and description are solicited from the user before performing the collapse. Other than the *specify node* operation all other generic node operations can be performed on the "super-node". If the user chooses to *explode* the *Process File* pseudotool, the network will be redrawn to show its pre-collapse state of Figure 9a. In addition to reversing the effect of a *collapse* operation, the explode operation can

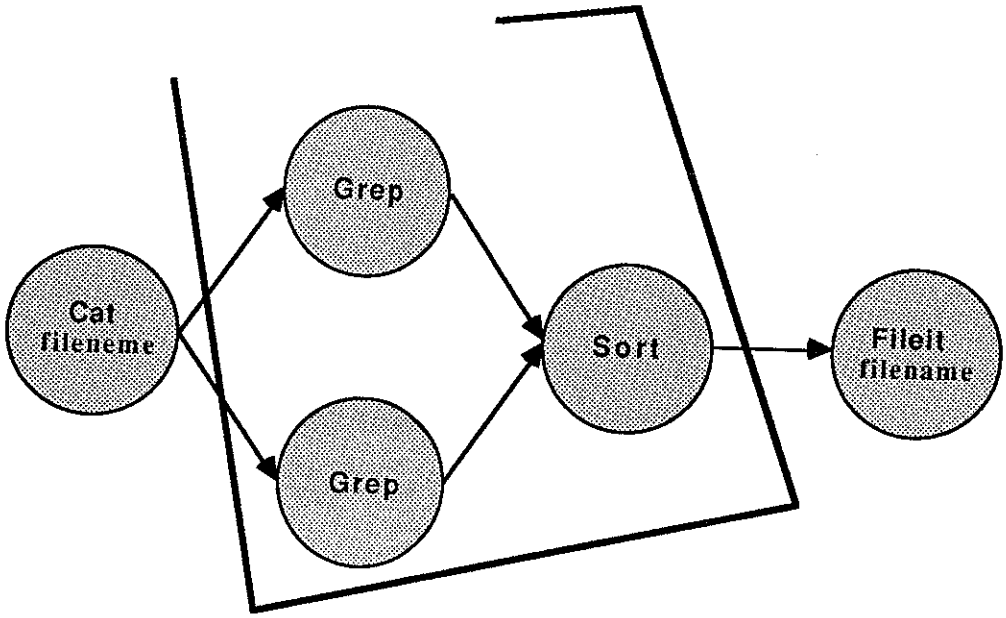

Figure 9a

Network Showing a Cutset Before the Collapse Operation

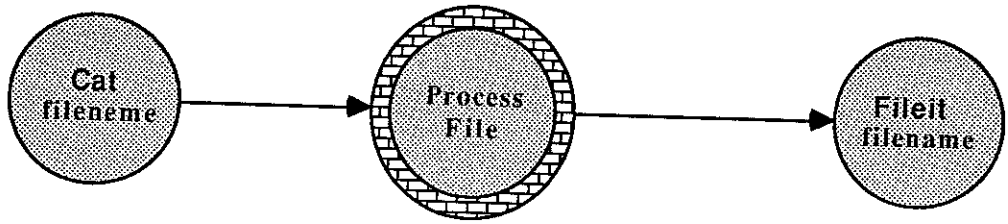

Figure 9b

Network Topology Display After the Collapse Operation

also be used on pseudotools "defined" by the *attach tool-composite* or through the *expand node* operation.

4.2.2 The Save Tool-Composite and Attach Tool-Composite Operations

The *save tool-composite* operation also creates an abstraction of the cutset similar to the collapse. While the collapse operation does an in-place replacement of the cutset with the pseudotool, the save tool-composite builds the pseudotool (based on a selected cutset) and stores it for later reuse. The network topology display is not altered by a save tool-composite operation. The *attach tool-composite* operation is used to attach an already saved pseudotool to an unspecified node, and hence, is one way of specifying an unspecified node.

5.0 Summary and Conclusions

Graphical images and conceptual abstractions are extremely useful in characterizing the complexities of user/machine interaction as related to tools-based, task specification. Within the Taskmaster environment, top-down task specification is achieved through the successive refinement of a graphical network where nodes represent operations and arcs correspond to communication paths between those operations. In support of the top-down specification process, Taskmaster effectively exploits the inherent powers of *multi-level*, menu-based interaction and the *node expansion* operation. On the other hand, bottom-up task specification is achieved through the *successive abstraction* of fully specified lower level networks into higher level operations. Additional applications of successive abstraction lead to the desired task specification and, as a by-product, a powerful set of *reusable* pseudotools. As touted by Boehm [BOEB84] and Munsil [MUNW85], the provision for reusable components can have a significantly beneficial impact on productivity.

Currently two prototype Taskmaster applications exploit abstraction in support of user task specification:

- a Unix-based, dataflow command shell supporting file transformation task specifications, and
- a matrix manipulation environment supported through selected LINPACK [DONJ79] routines.

Knowledge gained from the synthesis of these two application environments, and their current use as experimental test-beds, has contributed significantly toward the development of *complementary* interface abstractions. A realization of those abstractions within the Taskmaster environment embody a unique blend of top-down and bottom-up task specification capabilities, all oriented around visual programming concepts.

References

[ARTJ88] Arthur, J., "GETS: A Graphical Environment for Task Specification," *Proceedings of The Seventh Annual Phoenix Conference on Computers and Communications*, Scottsdale, Arizona, March 1988, To Appear.

[ARTJ87] Arthur, J. and Comer, D., "An Interactive Environment for Tool Selection, Specification, and Composition," *International Journal of Man-Machine Studies,* Vol. 26., No. 5, May 1987, pp. 581-596.

[ARTJ85] Arthur, J., "Partitioned Frame Networks for Multi-Level, Menu-Based Interaction", *Proceedings of the Fourth Annual Phoenix Conference on Computers and Communications,* Scottsdale, Arizona, March 1985, pp. 34-39.

[BOEB84] Boehm, B., et al., "A Software Development Environment for Improving Productivity," *IEEE Computer,* Vol. 17 No. 6, June 1984, pp. 30-42.

[DONJ79] Dongarra, J., et al., *Linpack User's Guide*, SIAM, Philadelphia, Pennsylvania, 1979.

[EHRR86] Ehrich, R. and Williges, R., *Human Computer Dialogue Design*, Vol. 2, Elsevier Amsterdam, 1986.

[GLIE84] Glinert, E. and Tanimoto, S., "Pict: An Interactive Graphical Programming Environment," *IEEE Computer*, Vol. 11, No. 11, November 1984, pp.7-25.

[MORM85] Moriconi, M. and Hare, D., "Visualizing Program Design Through PegaSys," *IEEE Computer*, Vol. 18, No 8, pp. 72-85.

[MUNW85] Munsil, W., "The Language Translation Task: Toward Reusable Components," *Proceedings of the Fourth Annual Phoenix Conference on Computers and Communications,* Phoenix, Arizona, March 1985, pp. 46-52.

[RABM59] Rabin, M. and Scott, D., "Finite Automata and Their Decision Problems," *IBM Journal of Research and Development,* Vol. 3, 1959, pp. 114-125.

[REIS86] Reiss, S., "GARDEN Tools: Support for Graphical Programming," *Proceedings of the Workshop on Advanced Programming Environments,* Trondheim, Norway, June 1986, pp. 519-536.

[TEIW81] Teitelman, W. and Masinter, L., "The Interlisp Programming Environment," *IEEE Computer*, Vol 14, No. 4, pp. 25-33.

[TESL81] Tesler, L., "The Smalltalk Environment," Byte, August, 1981, pp. 90-147