

178
25

Consistency and Tool Abstraction: Issues in the Taskmaster Environment

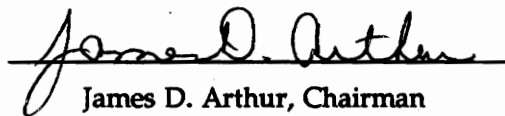
by

Brenda J. Jackels

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Science and Applications

April, 1990

APPROVED:


James D. Arthur, Chairman


Dennis G. Kafura


Rex Hartson

C 2

LD

5655

V855

g 322

1990

C. 2

Consistency and Tool Abstraction: Issues in the Taskmaster Environment

by

Brenda J. Jackels

James D. Arthur, Chairman

Computer Science and Applications

(Abstract)

This thesis presents Taskmaster.2, a graphical environment for interactive task specification, execution and monitoring. Problem solving in the Taskmaster environment can be accomplished with top-down programming, bottom-up programming, or a mixture of the two. The use of top-down programming permits the user to start with a high level task and refine this task into successively lower level subtasks until, at the lowest level, each subtasks represents a software tool. Bottom-up programming is accomplished by beginning with the lowest level subtasks, software tools, and then combining these tools into successively higher level subtasks until, at the highest level, the high level subtask represents the original problem task. These programming methods provide the user with abstraction capabilities. Another abstraction capability within the Taskmaster.2 environment is the network tools. The user creates network tools by selecting several software tools that, combined, provide a certain functionality. These network tools can then be reused in solving other problem tasks. In fact, these tools appear no different to the user than the low level software tools: they are both single indivisible units. Providing complete abstraction capabilities, i.e. mixing programming styles (top-down and bottom-up) and network tools, maintains the consistency of the Taskmaster.2 environment. This makes the environment an easy one to learn, as well as remember.

Key word and phrases:: Bottom-Up Programming, Consistency, Functional Abstraction, Network Tools, Portability, Top-Down Programming.

Acknowledgements

First and foremost, I would like to thank my parents for supporting me. Nope, I don't just mean financially. They've helped me through some pretty rough times and I can never thank them enough for that. Thank you Mom and Dad.

I would also like to thank my advisor, James "Sean" Arthur. He put up with me always being right (Ha!) and his guidance has been invaluable. For their time and assistance, I owe my thanks to my committee members, Rex Hartson and Dennis Kafura.

Table of Contents

Introduction	1
1.1 User Interface Development	1
1.2 The Taskmaster Environment and its Programming Philosophy	2
1.3 Motivations for This Research.....	3
1.4 Goals.....	4
1.5 Plan of the Thesis	5
Background & Approach	6
2.1 Taskmaster Environment	6
2.1.1 Terminology.....	8
2.1.2 An Example: The Programming Philosophy and Terminology	9
2.1.3 The Network Editor	14
2.1.4 The Execution Monitor.....	28
2.1.5 The Tools Database.....	31
2.1.6 The Interaction Among Taskmaster Components	32
2.2 Achieving the Goals of this Research.....	33
2.2.1 Portability.....	33
2.2.2 User Interface Consistency.....	36
2.2.3 Summary of Goals Achievement.....	39
Achieving a Consistent Interface	40
3.1 Introduction	40
3.2 Consistency.....	41
3.3 Inconsistency in the Taskmaster.1 Interface.....	42

3.3.1	Terminology.....	42
3.3.2	The Inconsistencies in the Taskmaster.1 User Interface	43
3.4	The Consistency of Arcs	44
3.4.1	Consistency in Arc Creation.....	45
3.4.2	Consistency in Arc Specification	47
3.4.3	Maintaining Arc Consistency: Ramifications on the Collapse Operation	47
3.4.4	Maintaining Arc Consistency: Ramifications on the Expand Operation.....	48
3.4.5	Maintaining Arc Consistency: Ramifications on the Explode Operation.....	49
3.4.6	Maintaining Arc Consistency: Ramifications on Tool Consistency	50
3.5	The Consistency of Tools.....	51
3.5.1	The User Interface: Tools.....	52
3.5.1.1	The User Interface: Network Tool Retrieval	52
3.5.1.2	The User Interface: Network Tool Creation.....	53
3.5.2	The Implementation: Tool Consistency.....	58
3.5.2.1	The Implementation: Network Tool Creation.....	58
3.5.2.2	The Implementation: Network Tool Retrieval	59
3.5.3	A Ramification of Maintaining Network Tool Consistency.....	60
3.6	The Consistency of Nodes	61
3.6.1	The Ports of a Super Node.....	61
3.6.2	Node Consistency and the Expand Operation	64
3.6.3	Node Consistency and the Delete Node Operation	65
3.7	Additional Enhancements to the User Interface.....	65
	The Port.....	69
4.1	Introduction	69
4.2	GKS0b versus X11.....	70

4.2.1	General Background	71
4.2.1.1	The X Windowing System.....	71
4.2.1.2	GKS0b	71
4.2.2	Windows and Their Contents.....	72
4.2.3	Input Capabilities	73
4.2.4	Output Capabilities.....	74
4.2.4.1	Coordinates	74
4.2.4.2	Setting Graphics Attributes.....	75
4.2.4.3	Text.....	76
4.2.4.4	Fill.....	77
4.2.4.5	Magnification.....	77
4.3	Changes due to Graphics Package.....	78
4.3.1	Zoom Operations.....	78
4.3.2	Rubberbanding	79
4.3.3	Menus and Other Input	79
4.3.4	The Display	80
4.4	System.....	82
4.5	Hardware.....	82
Conclusions		84
5.1	Contributions of this Research.....	84
5.1.1	Portability	84
5.1.2	Consistency in the User Interface	85
5.2	Limitations.....	86
5.3	Future Research Possibilities.....	88
References.....		90

Vita..... 94

List of Figures

Hardware Configuration:.....	Figure 2.1	7
Graphical Node Depiction:.....	Figure 2.2	10
Task Design:	Figure 2.3	12
Subtask Decomposition:.....	Figure 2.4	13
Operations Allowed on Nodes:	Figure 2.5	16
Initial Window Configuration:.....	Figure 2.6	20
A Taskmaster Network:	Figure 2.7	21
Explosion of a Super Node:	Figure 2.8	22
Collapse Operation:.....	Figure 2.9	23
Node Expansion:	Figure 2.10	24
Arc Specification:	Figure 2.11	25
View Node:	Figure 2.12	26
Graphical Node Depiction During Execution:.....	Figure 2.13	30
Collapsing with Arc Preservation:.....	Figure 3.1	46
Specifying a Node:.....	Figure 3.2	55
Network Display of Tools Database Hierarchy:.....	Figure 3.3	56
Network with Ports Shown:	Figure 3.4	62
The Three Port Models:	Figure 3.5	63
Tool Menu Examples:.....	Figure 4.1	81

Chapter 1

Introduction

1.1 User Interface Development

The importance of a quality user interface for computer software applications is increasing dramatically as the community of computer users grows. Due to the advances in computer hardware and software there are many possible interfaces for a given application. These range from a textual interface using a command language to a verbal interface using natural language. A useful interface, regardless of type, embraces the concept of consistency. A consistent interface can facilitate ease of learning by using the same command for the same operation in different situations. The Macintosh¹ interface offers an example of a consistent command. When the user desires to delete an item, be it an application, document or folder, the icon representing the item to be deleted is “thrown away”, i.e. dragged to an icon representing a repository of items to be removed. Remembering the proper usage of an application is aided by a consistent interface; the user need not remember multiple methods of performing the same operation.

¹Macintosh and A/UX are trademarks of Apple Computers

1.2 The Taskmaster Environment and its Programming Philosophy

Taskmaster is an interactive graphical environment for task specification and execution. To “program a given task within the Taskmaster environment, one decomposes the task into an ordered set of conceptually simple, high-level operations, and then combines (composes) a corresponding network of software tools that implements those operations” [ARTH83]. A tools database containing a collection of software tools supports the task specification operations. The Taskmaster environment allows the user to solve problems using either a top down or bottom up specification approach [ARTH88]. The top down approach can be viewed as the decomposition of a problem task into a set of high-level subtasks. Subtasks are progressively decomposed into smaller subtasks until, at the lowest level, each subtask represents a tool from a pre-existing tools database. The bottom up approach can be viewed as a composition of tools into higher level subtasks until, at the highest level, the subtasks represent the problem task. The final set of subtasks from either approach can be executed, providing the problem solution.

The development of the Taskmaster environment has been an evolutionary one. Each stage of its development was concerned with different aspects of the environment. Originally, the task specification environment was a textually oriented one [ARTH87]. The second stage of development concentrated on developing a graphical interface for the environment, as well as providing visual abstractions for subtasks [MULH87]. Visual abstraction allowed the user to group subtasks into higher level subtasks, although no functional information about the low level subtasks underlying the high level subtasks was retained. Functional abstraction and refinement was the focus during the third stage of development [RAGH88]. Functional abstraction provides the user with the ability to save high level subtasks for future use while retaining information about the underlying low level subtasks, i.e. functional information.

Taskmaster, the result of the above evolution, is not without its limitations. Consistency of the interface did not play an important role during its development. Moreover, its support for user defined functional abstraction was not well integrated.

1.3 Motivations for This Research

Taskmaster is designed to support user task specification. Although each stage of its development reached the goals of that stage, the following deficiencies are present in the Taskmaster environment:

- lack of portability,
- an inconsistent presentation of the environment to the user, and
- limited functional abstraction capabilities.

The lack of portability stems from the initial implementation constraints. In particular, Taskmaster communicates between two machines that have different operating systems, VMS² and UNIX³. Also contributing to the lack of portability is the graphics package used: the Graphical Kernel System (GKS). Part of this research has focused on porting Taskmaster to machines that have similar operating systems, i.e. UNIX. The port from VMS to UNIX has allowed the use of a different graphics package: The X Windowing System. This graphics package is becoming the de facto standard and has significantly increased the portability of the Taskmaster environment.

The original use of the GKS graphics package also contributed to an inconsistent presentation of the environment to the user. GKS does not provide flexibility to the programmer. Attendant

² VMS, VAXstation, VAX, Ultrix, DECnet, and MicroVMS are all trademarks of the Digital Equipment Corporation

³ UNIX is a trademark of AT&T

restrictions are reflected in applications developed using this package. The X Windowing System provides the programmer with complete control over the display device. This allows the design of an interface that presents a consistent view of the environment.

The incomplete integration of support for functional abstraction contributes to the conceptual inconsistency of the Taskmaster environment. The functional abstraction provided to the user in the last stage of Taskmaster's development allows the user to create and save high level subtasks. However, the implementation of this abstraction leaves much to be desired. When reusing a previously saved high level subtask, the user cannot treat the subtask as one indivisible unit, and must manipulate the individual low level subtasks underlying the high level one. As mentioned earlier, these low level subtasks ultimately represent software tools.

Another drawback caused by the implementation of functional abstraction is the lack of support for the mixing of top-down and bottom-up programming. Direct modification of a high level subtask is not permitted. In order to alter the functionality of a high level subtask, the user must first replace it with its underlying low level subtasks. The user can then modify these subtasks.

1.4 Goals

The first goal of this research is the consistency of the Taskmaster interface. As mentioned earlier, a consistent interface can facilitate the use of an application, as well as aid the learning process. Inconsistencies in the interface are exhibited in

- the display of the environment,
- the relationship between software tools and subtasks, and
- the capabilities for modification of subtasks.

The second goal of this research is the portability of the Taskmaster environment. The portability of Taskmaster is effected mainly by the package used to provide graphics and by the operating systems between which the environment must communicate.

1.5 Plan of the Thesis

A detailed description of the Taskmaster environment which contrasts Taskmaster.2 (the version resulting from this research effort) with Taskmaster.1 is provided in Chapter 2. This chapter also briefly outlines the achievement of the goals of this thesis. Chapter 3 elaborates on the achievement of the first goal of this thesis, consistency. The areas of Taskmaster.1 lacking in consistency are identified, and the possible methods used to maintain consistency in these areas are explained. The advantages and disadvantages of each method are explored previous to presenting the method chosen. These methods impact areas of the Taskmaster environment that do not lack in consistency, and this impact is also discussed. Chapter 4 discusses the achievement of the second goal of this thesis, portability. The methods used to achieve portability are discussed, and the differences between the implementations of the two version of Taskmaster are pointed out. Chapter 5 summarizes the contributions of this thesis, addresses the limitations of the current implementation of the Taskmaster environment and explores future research possibilities.

Chapter 2

Background & Approach

2.1 Taskmaster Environment

The Taskmaster environment is comprised of three components – the network editor, the execution monitor and the tools database - which work together to provide an environment supporting the Taskmaster programming philosophy explained above. The network editor resides on a Macintosh II running A/UX (local workstation), and the execution monitor on a VAXstation 2000 running Ultrix-32 (host computer). These two components are connected by a high speed communication link, as depicted in Figure 2.1. A master copy of the tools database resides on the host computer with the execution monitor, and a local copy resides on the user workstation with the network editor.

The network editor provides the graphical interface to the user for creating networks and for sending networks to the execution monitor for instantiation. Once a network is instantiated by the execution monitor, the execution of the network is monitored and information pertaining to the execution status of the network is sent back to the network editor. The network editor then

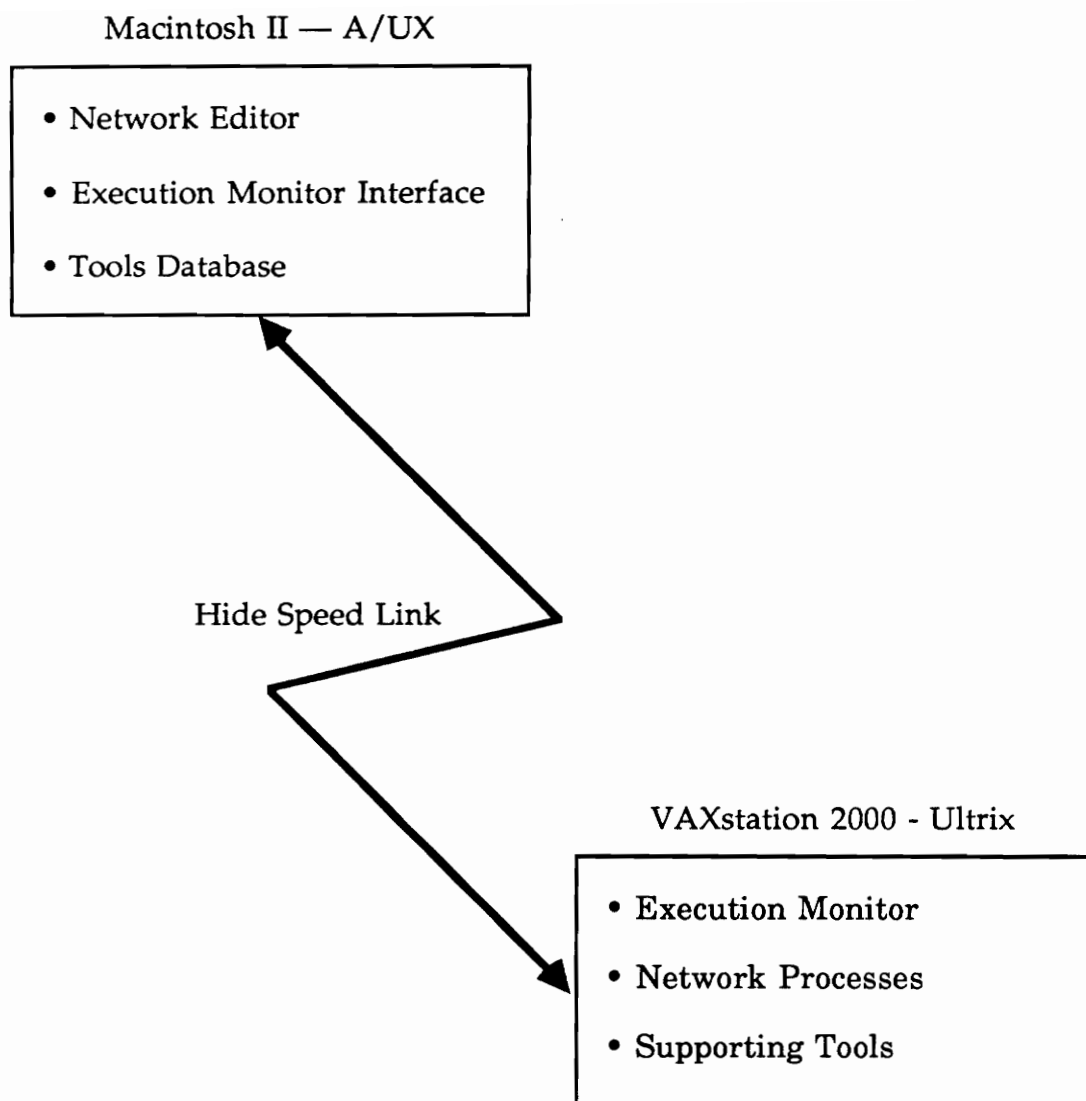


Figure 2.1
Hardware Configuration

conveys this information to the user. The tools database provides tools, and information about the tools, to the network editor. The three components of Taskmaster are described in detail in the following sections, beginning with an explanation of the terminology. To achieve the goals of this research, as briefly discussed in Section 1.3 and elaborated on in Section 2.2, an enhancement of the Taskmaster package is required. The original version is referred to as Taskmaster.1 and the enhanced version as Taskmaster.2. Both Taskmaster.1 and its successor, Taskmaster.2, are discussed in the following sections.

2.1.1 Terminology

As mentioned previously, problem solving in the Taskmaster environment involves decomposing a task into subtasks. Subtasks are depicted graphically as nodes and the communication between the subtasks as arcs. The collection of nodes and arcs is referred to as a network. Subtasks are progressively decomposed into lower subtasks until each subtask, depicted as a node, represents a tool from a pre-existing database. A tool, as used in this thesis, is a program which performs a single high-level operation with minor variations. There are two types of tools, atomic tools and network tools. As is implied by the name, an atomic tool is a single unit and cannot be decomposed. A network tool, however, is a tool composed of many tools. Note that both types of tools perform a single high-level operation. Network tools are created by the user and stored in the tools database for future use. Ports are used to indicate whether or not a tool, network or atomic, performs any input or output function, with one port for each input and each output stream. These ports are used for communication with other tools.

Two types of nodes are present in the Taskmaster environment: atomic nodes and super nodes. As implied by the name, atomic nodes are single units and cannot be decomposed. A super node represents a collection of nodes, i.e. a subnetwork. Both types of nodes can be either specified or

unspecified. A specified atomic node is a node with a tool from the database attached to it, and an unspecified atomic node has no tool attached to it. A specified super node represents a subnetwork in which all the nodes are specified. An unspecified super node represents a subnetwork in which at least one node is unspecified. Note that the definition of a specified or unspecified super node is recursive in nature. The different nodes are distinguished by their graphical depictions (see Figure 2.2):

- specified atomic node - solid black;
- unspecified atomic node - grey;
- specified super node - solid black surrounded by grey ring;
- unspecified super node - grey surrounded by grey ring.

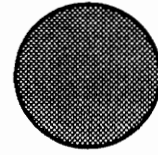
Communication between the nodes, represented by arcs, is in one direction only. The arcs are graphically depicted as line segments, with an arrow on each line indicating the direction of the data flow. A specified arc is an arc for which the communication between the nodes at the ends of the arc has been completely defined; at least one connection has been made between an output port of one node and an input port on the other node. Specified and unspecified arcs can be distinguished by their graphical representation. Specified arcs are represented by solid lines and unspecified arcs by dashed lines. A generic, or unspecified, network is a network that contains any nodes or arcs that are unspecified. A specified network is a network in which all nodes and all arcs are specified.

2.1.2 An Example: The Programming Philosophy and Terminology

The following example, provided by [RAGH88], illustrates the top down programming paradigm. A matrix multiplication scheme with vector operations can be specified with a Taskmaster network, and allows the user to exploit the parallelism offered by vectorization. A



Specified



Unspecified

Atomic Nodes



Specified



Unspecified

Super Nodes

Figure 2.2
Graphical Node Depiction

high-level diagram representing such a design is pictured in Figure 2.3. The pre-multiplier matrix A is vectorized by row by the subtask "Vectorize by row" and the post-multiplier matrix B is vectorized by column by the subtask "Vectorize by column". The product matrix elements are individually computed in parallel by the subtask "Perform parallel vector multiplications", whose output is sent to the subtask "Compose product matrix". This subtask produces the final output, the problem solution, by composing the product matrix. The details of how these subtasks achieve their goals are not addressed at this level of decomposition. Given the primitive tools from the pre-existing tools database "vectorize by row", "vectorize by column", "combine row elements", "combine row vectors", and "multiply row by column" ("mult" in Figure 2.4), for the case where both A and B are two-dimensional matrices, the high level task diagram can be refined to the fully defined diagram shown in Figure 2.4. Each high level subtask is replaced with the tools that are appropriate for accomplishing the given subtask.

By looking at the above example in reverse, the bottom up approach is demonstrated. The lowest level subtasks are created first and are associated with tools from the database, as shown in Figure 2.4. These subtasks are grouped together to form the higher level subtasks shown in Figure 2.3.

This conceptual specification is supported via tools from the database, nodes and arcs. The subtasks "vectorize by row", "vectorize by column", "mult", "combine row elements", and "combine row vectors" in Figure 2.4 correspond to atomic nodes representing tools from the database. The lines connecting the subtasks correspond to arcs representing communication paths between the tools. The subtasks "vectorize by row", "vectorize by column", "perform parallel vector multiplications" and "compose product matrix" from Figure 2.3 correspond to

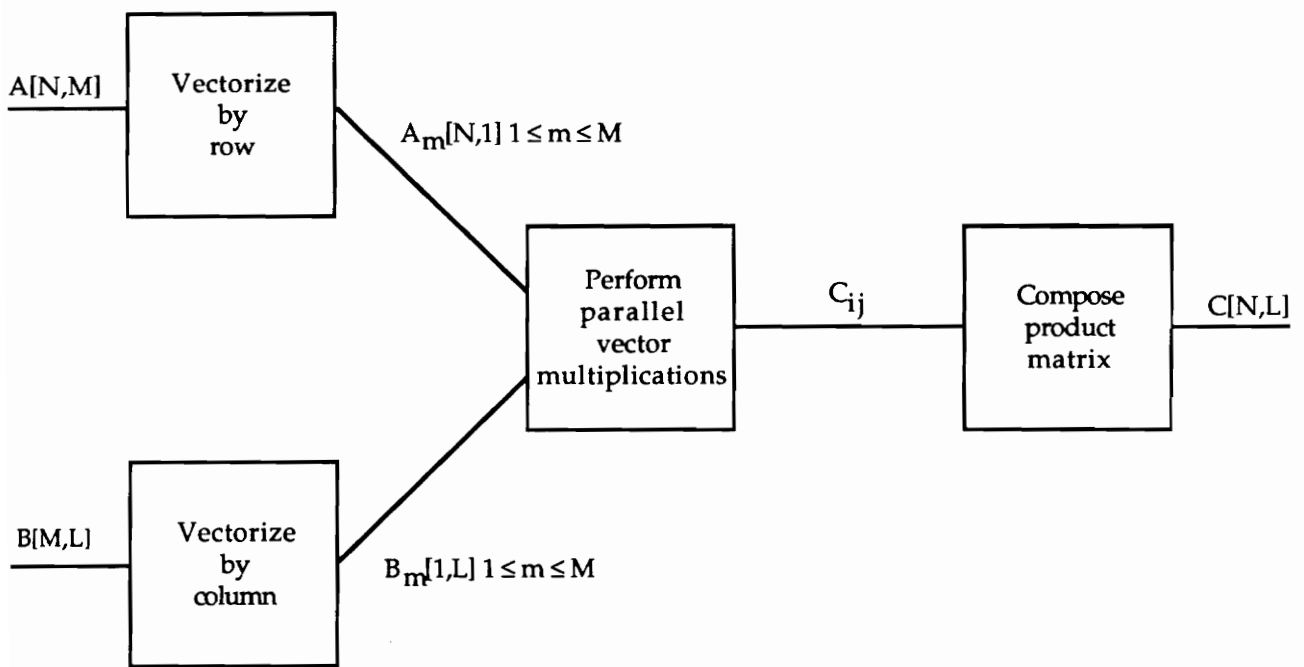


Figure 2.3
Task Design

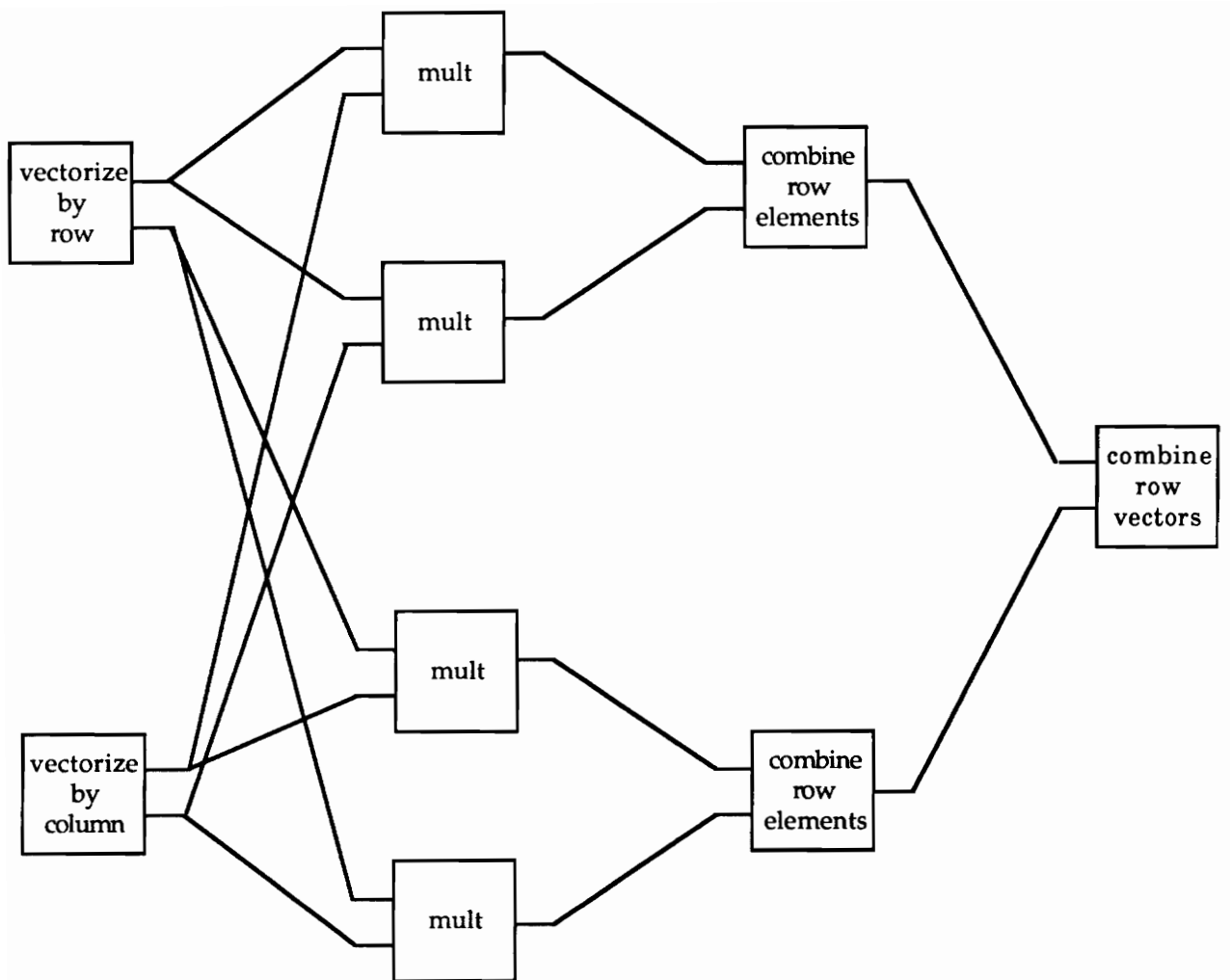


Figure 2.4
Subtask Decomposition

super nodes. Both figures in their entirety correspond to a Taskmaster network.

2.1.3 The Network Editor

With the use of the windowing and graphics capabilities of the X Windowing System [NYE88a], [NYE88b], the network editor provides a graphical user interface for network development. This interface provides operations to create generic networks and to specify the nodes and arcs using a menu-driven interaction process. This process makes use of windowing and mouse input, and assists the user with instructions and help and error messages.

The operations supplied by the network editor can be categorized as follows:

- generic network development
 - create node, delete node,
 - create arc, delete arc
 - explode node, expand node,
 - collapse
- network specification
 - specify node, specify arc
- network viewing
 - move node,
 - view node, view arc
 - zoom in, zoom out
 - pan
- network storage and retrieval
 - save network, create new network, open network,

create network tool , attach network tool

- error recovery

undo last operation

- network instantiation

execute network.

The application of these operation often depends on the type of the nodes involved, as can be seen in Figure 2.5. Many of these operations have different names in Taskmaster.1. What the differences are and the reasons for the differences are discussed in Section 3.7. These operations are supported through the use of a large window displaying the network topology currently being edited, and additional windows for displaying

- menus,
- multiple views of nodes and arcs,
- instructions,
- help and error messages, and
- user confirmation requests.

These windows and their uses are explained in more detail in the following paragraphs. The initial window configuration containing the topology window, instruction window and main menu is shown in Figure 2.6.

The **create node** operation graphically creates an atomic node on the display device. Any node can be deleted with the **delete node** operation in Taskmaster.2. Taskmaster.1, however, does not allow this operation to be applied to super nodes. Any arcs associated with a node being deleted are also deleted. Creating an arc with the **create arc** operation allows the user to indicate that there is a data flow between two nodes. The user is prompted to select two nodes. The first node selected is the start, or left end, of the arc. The second node selected is the end, or

	Atomic Nodes		Super Nodes	
	Unspecified	Specified	Unspecified	Specified
delete node	•	•	•	•
create arc ¹	•	•	•	•
delete arc ¹	•	•	•	•
explode node			•	•
expand node	•		•	•
collapse ²	•	•	•	•
specify node	•	• ³		
specify arc ¹	•	•	•	•
move node	•	•	•	•
view node	•	•	•	•
view arc ¹	•	•	•	•
create network tool ²		•		•
attach network tool	• ⁴			

1. The indicated nodes can be at the ends of the arc having the operation applied to it.
2. The indicates nodes can be included in the subnetwork selected during the operation.
3. Respecify
4. The atomic node becomes a super node after the network tool is attached.

Figure 2.5
Operations Allowed on Nodes

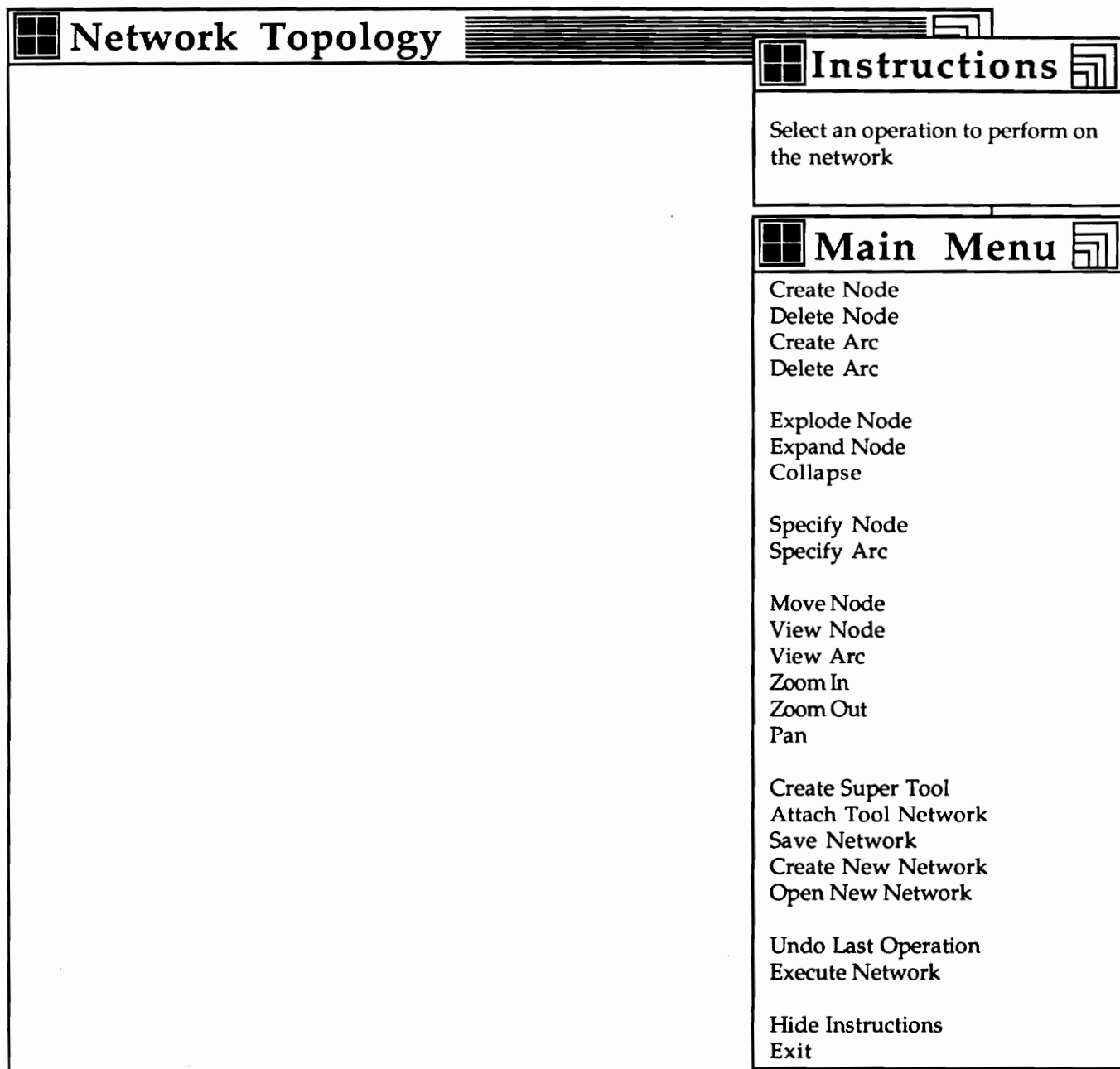


Figure 2.6
Initial Window Configuration

right end, of the arc. The data flow that the arc represents begins at the output of the first node and ends at the input of the second node. Creating an arc creates a *communication path* between the two nodes. Taskmaster.1 requires that the nodes at both ends of the arc be atomic nodes. This requirement has been removed in Taskmaster.2, and the nodes can be either atomic or super nodes. The **delete arc** operation can be applied to either specified or unspecified arcs.

Modification of a subnetwork represented by a super node can be achieved in one of two ways in Taskmaster.2: expanding the super node or exploding the super node. In Taskmaster.1, exploding a super node is the only way to modify its subnetwork. More specifically, the **explode node** operation results in the selected super node being replaced by the subnetwork that it represents and **expand node** results in creation or modification of a subnetwork (see below for further explanation). Figure 2.8 shows the resulting network after applying the explode node operation to the super node “super” in the network shown in Figure 2.7.

Abstraction in Taskmaster is achieved with the use of the collapse and expand operations. The **collapse** operation allows the user to create a super node by selecting nodes and arcs in the current topology to be in its subnetwork. The identified subnetwork is subsequently represented by a super node. Selection of the nodes and arcs is accomplished by drawing a polygon around the nodes and arcs that are to be in the subnetwork of the super node, as is shown in Figure 2.9. After the polygon is completed, a name for the super node is requested from the user, and the subnetwork is replaced by the super node (Figure 2.7).

The **expand** operation can be applied to both atomic and super nodes in Taskmaster.2. Expansion of a super node is not allowed in Taskmaster.1, requiring the user to explode a super node in order to modify its subnetwork. Expansion of either type of node in Taskmaster.2 results in a new network window being displayed, and allowing network modification in the new

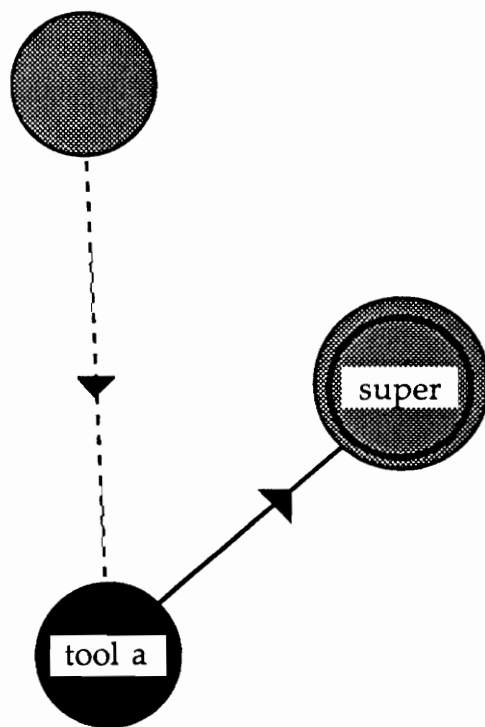


Figure 2.7
A Taskmaster Network

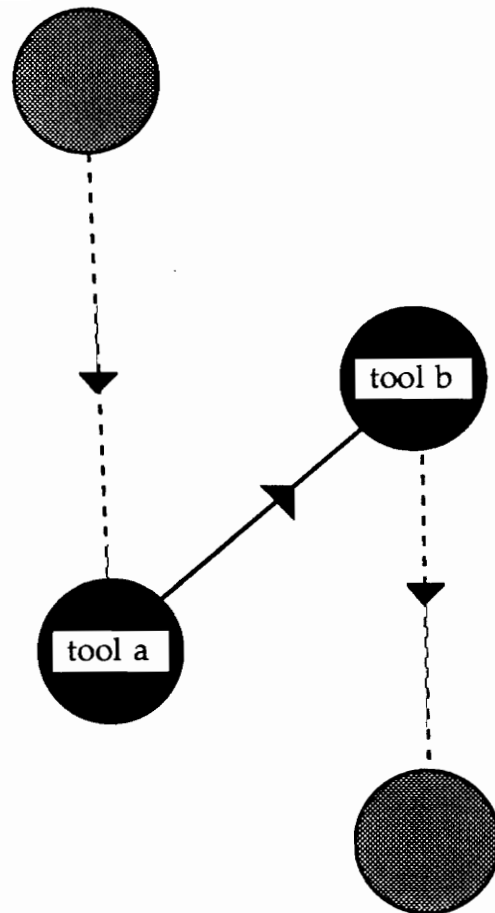


Figure 2.8
Explosion of a Super Node

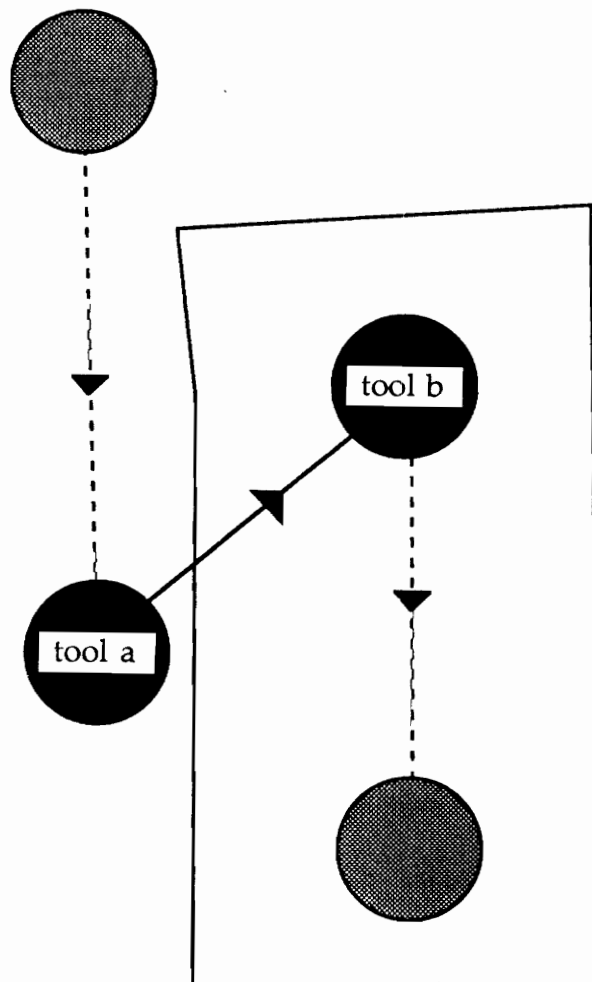


Figure 2.9
Collapse Operation

window. If the node being expanded is a super node, the subnetwork that the node represents is displayed for modification in the new network window, as is shown in Figure 2.10, which shows the display when expanding the super node “super” from the network in Figure 2.7. As can be seen in Figure 2.10, the operations menu presented during expansion is the same as the main operations menu (Figure 2.6), minus the operations: save network to disk, restore network, and execute network.

The expand and collapse operations both create super nodes. The subnetwork represented by the super node supplies the super node with ports. There are several alternatives for determining which ports from the subnetwork are considered the ports of the super node. All of the alternatives as well as the one chosen are discussed in Section 3.6.1.

A generic network by itself can display useful information about the final task solution. However, in order to execute the network and obtain the task solution, tools must be attached to the nodes, and the communication between the tools must be defined. Attaching a tool from the database to a node can be achieved through the **specify node** operation. The user is led through a series of menus until the specific tool is chosen. After the tool is chosen, the user is prompted to select optional arguments and to supply values required by arguments. The information required by the network editor in order to prompt for the arguments is retrieved from the tools database. These arguments are parameters that may be given to the tool (optional arguments) and must be given to the tool (required arguments). For example, the UNIX grep tool has optional arguments (-v, for example) and required arguments (the search string).

The user can apply the **specify arc** operation to any arc in the network. A specification window is displayed (Figure 2.11), and the user can select any output port from the node on the left end of the arc to connect to any input port from the node on the right end of the arc. After the user

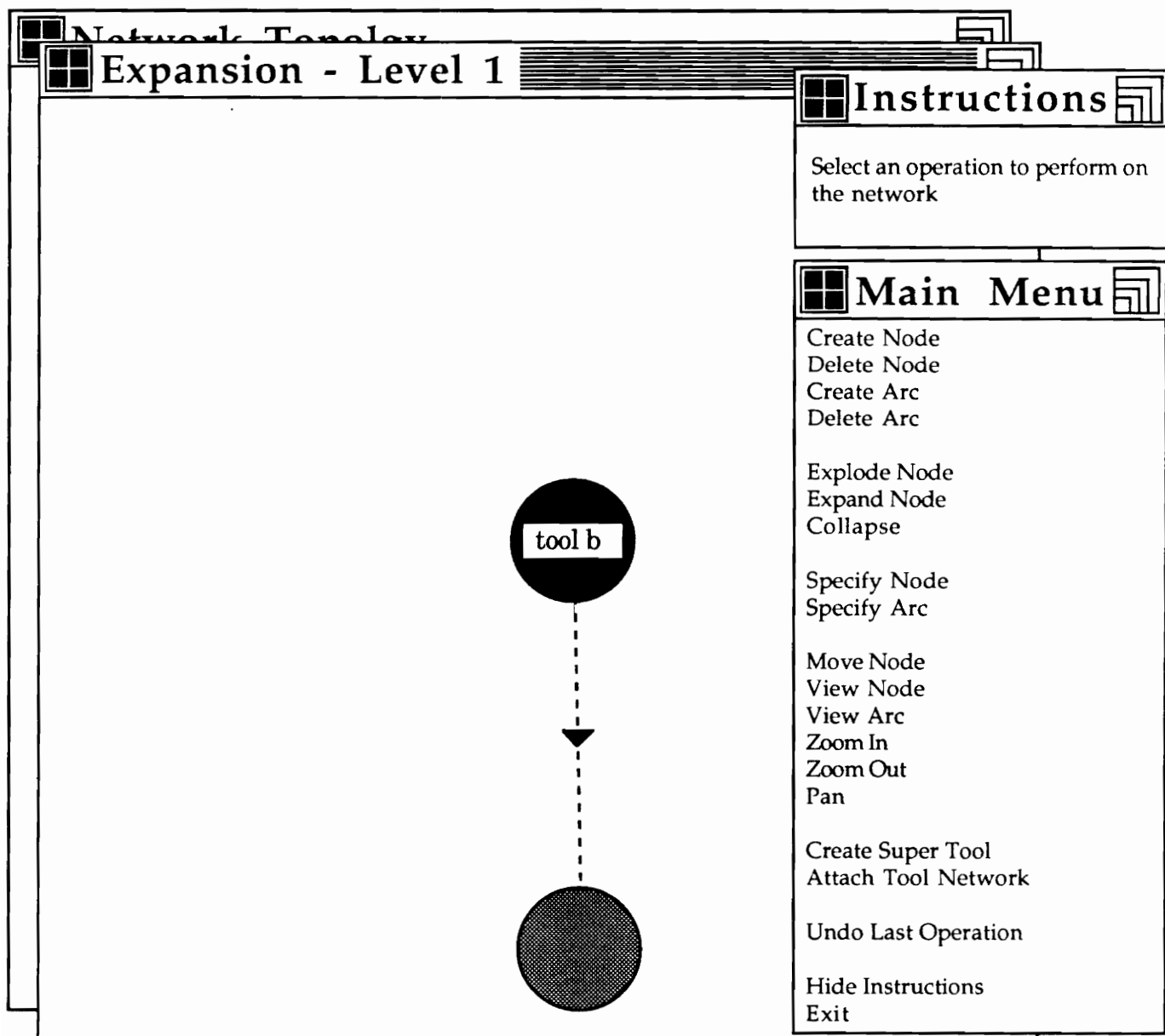


Figure 2.10
Node Expansion

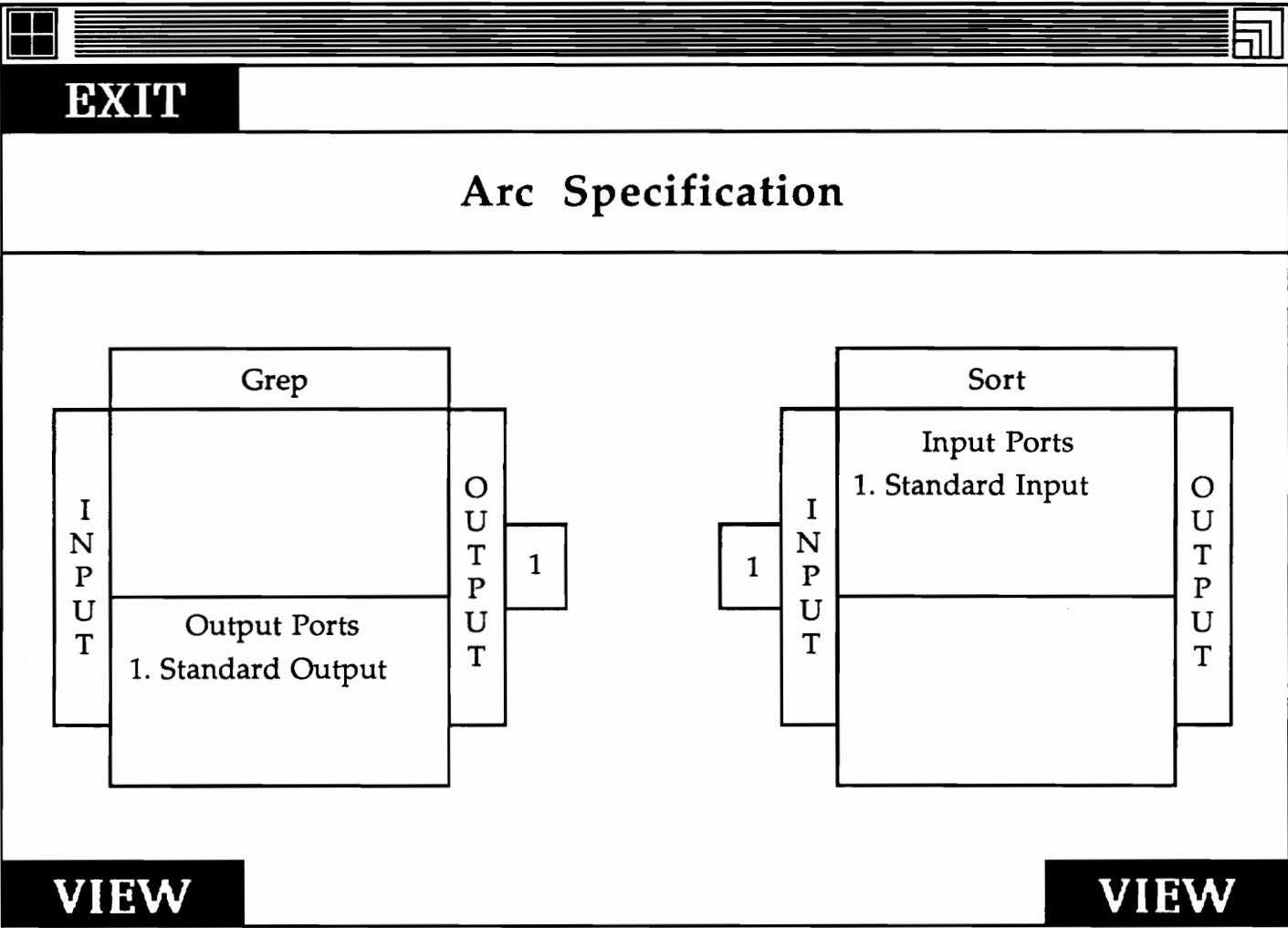


Figure 2.11
Arc Specification

indicates which output port is to be connected to which input port, the network editor checks if the type of the ports are compatible by retrieving type information from the tools database. If the type of the output port is not compatible with the type of the input port, an error message is issued, and the connection is not made.

The **move node** operation can be used on either atomic or super nodes. Any arcs associated with a node that is being moved will follow the node.

The default viewing of a network, a *level 1 view*, is as a network topology consisting of nodes and arcs. A *level 2 view* displays communication information about a given node; Figure 2.11 displays a level 2 view of two nodes. A detailed tool description is shown in a *level 3 view* of a node, as is shown in the right half of Figure 2.12. The user can get information specific to a node by selecting **view node**. A tool information window is presented to the user, displaying the level 2 and level 3 information pertaining to the selected node (Figure 2.12). The same tool information can be seen from the arc specification window by clicking on the view button associated with that node. A detailed description of a port can be seen by clicking on the port in the tool information window. The **view arc** operation displays the same window as the **specify arc** operation with the same level 2 view of both nodes, without the specification ability, but retaining the ability to view the node information.

To help the user manipulate complicated and large networks, the graphical view of the network can be changed with one of the zoom commands, or the pan command. The user has the option to **zoom in** and **zoom out** in order to make the relevant portion of the network topology fit in the topology window. The **pan** operation allows the user to indicate a new point to be centered in the window.

EXIT	
Communication Requirements	Tool Description
	<p>Tool Name: Grep</p> <p>Description:</p> <p>Grep is a filter that searches the input stream for lines containing a specified pattern. Lines containing the pattern are written to the output.</p> <p>Attributes:</p> <p>hello</p>
<p>Click the mouse on a port for a detailed description of the port</p>	

Figure 2.12
View Node

A network can be saved to disk and retrieved from disk using the **save network to disk** and **open network** operations respectively. The current network can be discarded and a new network created with the **create new network** operation. A subnetwork can be saved as a network tool for later reuse using **create network tool**. As defined earlier, a network tool is composed of many tools. Therefore, all of the nodes in a subnetwork being saved as a network tool must be specified previous to including them in the subnetwork. A feature of Taskmaster.2 is that once saved, a network tool can be attached to an atomic node in the same manner as an atomic tool; the type of the tool is transparent to the user. To support this feature, the network tool is stored in the tools database (see Section 3.5 for a detailed discussion on the creation of a network tool). The subnetwork of a network tool can be retrieved for modification with the **attach network tool** operation. The user is prompted to select a node to which the subnetwork is attached. Once the subnetwork is attached, the node becomes a super node.

Error recovery is supported by the **undo** operation and negates the effect of the most recent operation that modified the network topology.

After a network has been created and specified it can be sent to the execution monitor for instantiation using the **execute network** operation. Before sending the internal representation of the network topology to the execution monitor, the network editor

- connects ports for unspecified, yet unambiguous arcs,
- creates special duplication and merge nodes for ports with multiple outputs and inputs respectively, and
- performs consistency checking while looking for unconnected ports within the network.

If any unconnected ports are found, the editor allows the user to correct any oversights. After the network passes inspection, the internal representation is sent to the execution monitor on the

remote host.

2.1.4 The Execution Monitor

Upon invocation, the execution monitor establishes communication with the network editor. This communication across the network is achieved with UNIX sockets [AUX88], as discussed in Section 2.1.6. After communication is established, the network editor sends the network to the execution monitor via the socket, and the execution monitor

- reads the network,
- validates the network,
- spawns the network, and
- monitors execution of the network.

Validating the network includes checking the connectivity of the network and checking data path consistency. A breadth first search is used to traverse the network during the validation stage. After the network has been validated, it is instantiated. Each node in the network becomes a separate process and the communication between the processes is achieved with the use of UNIX pipes. Using UNIX pipes directly restricts the user to using character streams as data flow. This restriction has been surmounted with the use of a post-processor and a pre-processor at the opposite ends of the communication link. These processors massage the data and allow for structured data flow. The post-processor embeds structure descriptors within the data stream and the pre-processor decodes these descriptors. A breadth first search is used also for node instantiation order in an effort to try and satisfy the interprocess communication restrictions in the operating system.

The execution monitor provides feedback to the network editor in order to keep the user apprised of the network's status. Each node is in one of three states:

- pre-instantiation - the process is not yet spawned,
- execution - the process is spawned and is either active or idle, or
- terminated - the process has completed.

The different states of a node are reflected in the graphical representation of the node (Figure 2.13). Initially, all the nodes are in the pre-instantiation state and are grey. When node reaches the execution state, its color changes to black with a white triangle on the bottom of the node. Within this triangle is either an "A" or an "I", indicating that the node is active or idle respectively. At node termination, the node becomes white to blend in with the background.

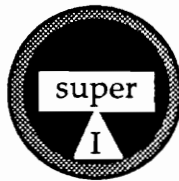
In addition to receiving execution status information, the user is able to change the view of the network topology during execution. Typing a user control interrupt sequence at the keyboard invokes the menu of operations. Although the user may not alter the functionality of the network, the view of the network may be modified using the following operations:

- move node, view node, explode node, expand super node,
- view arc,
- collapse,
- zoom in, zoom out, pan, and
- undo last operation.

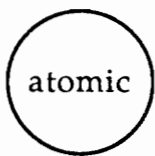
In Taskmaster.2 expansion of a node is allowed during execution, but on a super node only; this operation is not allowed in Taskmaster.1 during execution. Expanding an atomic node is prohibited because of the nature of the operation – it results in a change in the functionality of the network. Although expansion of a super node is allowed during execution, modification of the functionality of the super node's subnetwork is not permitted.



Pre-instantiation



During execution — active and idle



Terminated

Figure 2.13

Graphical Node Depiction During Execution

An additional operations allowed during network execution is **abort execution**. Applying this operation immediately stops network execution.

2.1.5 The Tools Database

The tools database is application independent, allowing the integration of different application domains within the Taskmaster environment with a minimum amount of effort. The database contains many tools, both atomic and network. For each atomic tool, the following information is stored:

- communication requirements - number and type of input and output ports,
- arguments, both required and optional,
- a brief and a detailed description of the tool,
- a brief and a detailed description of each port, and
- other information for use during node specification.

For each network tool, the database contains a brief description of the tool and a file name. The subnetwork of the network tool is stored in a file with the name given in the tools database. When retrieving a network tool, the network editor imports the subnetwork of the network tool from the file and attaches it to the selected node. This process is transparent to the user, whether retrieving a network tool or an atomic tool (see Section 3.5 for a detailed discussion on the retrieval of a network tool).

The information in the database drives the node and arc specification processes, as well as provides information used in the different views of the network. The node specification process uses the information from the database for menu creation and for requesting argument values from the user. Arc specification retrieves the port descriptions and the communication

requirements in order to assist the user in establishing data flow. The descriptive information from the database is presented to the user through level 2 and level 3 views of the network.

The master copy of the tools database resides on the same machine as the execution monitor, the VAXstation 2000. When invoking the network editor from the user workstation, the user is asked if he wants the local copy of the tools database to be updated by copying the master copy of the database from the host computer, the VAXstation 2000. During an editing session on the user workstation, the user can modify the local tools database by saving network tools. If such modifications are made to the tools database, the modified database can be saved both on the local workstation and to the remote host upon confirmation from the user at the end of an editing session.

2.1.6 The Interaction Among Taskmaster Components

As mentioned earlier, the network editor of Taskmaster.2 resides on a Macintosh II running A/UX, and the execution monitor on a VAXstation 2000 running Ultrix-32. The execution monitor and the network editor communicate via UNIX sockets in Taskmaster.2 (as opposed to the DECnet protocols used in Taskmaster.1). In the use of sockets, the network editor plays the role of the server and the execution monitor that of the client. On invocation, the network editor creates a socket and signals that it can accept communication on that socket. When a network is ready for instantiation, the network editor invokes the execution monitor with the use of the "remsh" command, and waits until the execution monitor is ready to receive the network. The execution monitor then creates its own socket and attempts to establish communication with the network editor. After communication is established, the editor sends the network to the execution monitor.

During execution of the network, the execution monitor sends information regarding the status of

the network back to the network editor via the socket. This information is sent only when there is a change in the network status, such as a node changing from one state to another: pre-instantiation to execution, active to idle (and vice-versa) within the execution state, or execution to termination.

The tools database is stored as a file, which can be read and modified by the network editor. The information stored in the tools database is used for driving the tool specification process, as mentioned above. Modification of the database occurs when the user saves a subnetwork as a network tool. The network tool's network is stored in a file, and a network tool record is created for the database, including the name of the file in which the network is stored. There are other alternatives for storing a network tool, and these are discussed in Section 3.5.

2.2 Achieving the Goals of this Research

Developing a portable Taskmaster environment and achieving conceptual consistency among operations supported by the network editor are the goals of this research. The approach taken to achieve these goals in Taskmaster are discussed briefly in the following sections. The attainment of user interface consistency is discussed in detail in Chapter 3; A detailed discussion on the achievement of portability can be found in Chapter 4.

2.2.1 Portability

The portability of software depends on the components of the software, namely

- the implementation language,
- any software packages used (such as math or graphics packages),
- the required hardware, and
- the required operating system.

Using standard components, i.e. standard languages, software packages, hardware and operating systems, contributes greatly to the portability of software. Not only must standard components be used, but they must be used in a standard manner; e.g. Assume the software must use standard operating system A and standard hardware B. If it is not common for B to support A, then the software does not use its components in a standard manner.

The portability of the Taskmaster environment also depends on its components. Using standard components for the implementation of Taskmaster in a standard manner will provide portability in the environment. The implementation language of Taskmaster.1 is C, the graphics package used is GKS, the hardware is a VAXstation I and a VAX 11/785, and the operating systems are MicroVMS and Ultrix. C is a common programming language, and therefore this component of the environment need not be changed to achieve portability. The hardware used does not reflect any requirements of the Taskmaster environment, but requirements of the graphics package and operating system, and therefore plays no direct role in the portability of Taskmaster. The graphics package and the operating systems used in the implementation of the Taskmaster.1 environment both contribute to the lack of portability of the environment, as discussed in detail in the following paragraphs.

Taskmaster.1 uses the GKS package, level 0b, to implement the graphics used by the network editor. To achieve a higher level of portability, the network editor of Taskmaster.2 uses the X Windowing System, version 11 (also referred to as X and X11 in this thesis). As mentioned in [JERN87], the GKS standard leaves too many important issues open to the developer of a GKS implementation, which results in lack of program portability, even when porting to another GKS implementation. Another factor affecting the portability of Taskmaster is the changing graphics standards and the emergence of new graphics packages. Although many graphics packages are touted as the standard, X is emerging as a de facto standard [ANDE88], [JERN87],

[KEEF88], [PATE88], [POUN89]. The X Windowing System can be implemented on any operating system, although it has been developed mainly under UNIX [POUN89]. This allows X to be portable, which in turn allows the network editor of Taskmaster.2 to be portable. As mentioned in [PATE88], often the functionalities of graphic systems and of windows management are developed in separated ways. The X Windowing System combines graphics and windowing capabilities, allowing the programmer to have complete control over the entire display in a consistent fashion. This is also in concert with one of the goals of this research effort.

Taskmaster.1 is physically located on two machines that have different operating systems, UNIX and VMS. Because of this, the environment is not easily ported. The first step in achieving portability with respect to the operating system is to implement Taskmaster on machines with the same operating system. The next step is to choose an operating system that is standard and can be used in a standard manner with the rest of the implementation components. Like the X Windowing System as a graphics package, UNIX is a de facto standard as an operating system [SCAN88], [PERL86a]. Combining the UNIX operating system with the X Windowing System and C is common. The portability and popularity of UNIX combined with the portability of X allows the entire Taskmaster package – both the network editor and the execution monitor – to be highly portable. Taskmaster.2's network editor and execution monitor both reside on machines that have UNIX as the operating system. Because of this, Taskmaster.2 can also run on a single UNIX machine. The communication between the network editor and execution monitor is also effected by both components residing on UNIX machines. This communication is achieved with the use of sockets, which can be used regardless of whether or not the two components reside on the same machine.

In summary, to achieve portability of the Taskmaster interface

- a standard graphics package must be used, and
- the environment must be supported by standard operating systems, between which communication is possible.

The graphics of Taskmaster.2 are supported by the X Windowing System. Both components of the Taskmaster.2 environment are implemented under the UNIX operating system.

2.2.2 User Interface Consistency

“The work of a user who interacts with multiple applications can be greatly simplified if the user interface across these applications is consistent.” [UHLI88]. This also holds true for the user of a single application – the work of a user who uses *any* application can be greatly simplified if the user interface *within* the application is consistent. When presented with a consistent interface the user can transfer his learning within the application, allowing him to spend more time problem solving [POLS88].

There is general agreement that consistency in a user interface is a goal for which to strive [SCHN87], [UHLI88], [LEWI89], [BERR88], [PERL86b], [PAYN86]. Although a precise definition for consistency is elusive, guidelines for achieving consistency do exist. These guidelines include

- presenting the display in the same manner in similar situations,
 - using the same terminology throughout the interface,
 - placing error messages and instructions in the same location each time they are displayed, and
 - requiring the use of the same command for the same results in similar situations
- [SCHN87].

The Taskmaster.1 interface follows the first of these guidelines. An example of this is the "Exit" button that appears during some operations. Whenever this button is displayed, it is in the upper right hand corner of the current window. The second guideline, using the same terminology throughout the interface, is not adhered to by the Taskmaster.1 interface. The names of some of the menu selections do not follow this guideline, as is discussed in Section 3.7.

The Taskmaster.1 interface does adhere to the third of the guidelines above; Error messages and instructions are displayed in the same location each time they are displayed. However, this can be a disadvantage. Messages are displayed in separate "pop-up" windows. Even if the user moves the window, the window appears in the original position the next time a message is displayed. A better expression of this guideline with respect to windows is

- placing error messages and instructions in the same location as they last appeared.

The three main areas in the Taskmaster.1 interface which lack in consistency reflect a lack of adherence to the fourth guideline; The same command cannot be used in similar situations with similar results. These areas are:

- arc operations,
- the relationship between network and atomic tools, and
- expansion and deletion of nodes.

Taskmaster.1 restricts arc operations by requiring atomic nodes at the ends of an arc at the time of arc operations. This is not a restriction in version 2. The user can perform these operations regardless of the types of the nodes at the ends of the arcs. Such an approach allows the user to specify communication paths to subnetworks within super nodes without exploding the super nodes. As well as achieving consistency within these operations, removing this restriction allows for easier mixing of top down and bottom up problem solving. The modifications to arc

operations are discussed further in Section 3.4.

Network tools and atomic tools are very different concepts in Taskmaster.1, requiring that the user master two distinct methods of attaching a tool to a node. Attaching an atomic tool is accomplished with the specify node operation in Taskmaster.1, whereas the attach network tool operation is required for retrieving a network tool. Attaching a network tool results in the network tool's subnetwork being retrieved from the file in which it was previously stored. This subnetwork is attached to a node selected by the user, which, in turn, becomes a super node. In Taskmaster.2, the specify node operation results in all of the atomic tools *and* network tools being presented to the user for selection. The ability to import the subnetwork of a network tool, as provided by the attach network tool operation in Taskmaster.1, is not lost and is furnished by the same operation in Taskmaster.2. The different methods for achieving this tool consistency, as well as the method used, are discussed in Section 3.5.

Atomic nodes and super nodes are similar in many ways; This similarity is reflected more completely in Taskmaster.2 than in Taskmaster.1. Both versions of Taskmaster allow both types of nodes to be moved, to be viewed and to be included in a subnetwork either for saving as a network tool, or for collapsing. The delete node and expand node operations are allowed only on atomic nodes in Taskmaster.1; This inconsistency is rectified in Taskmaster.2. The different approaches for achieving node consistency are discussed in Section 3.6.

In summary, to achieve consistency in the Taskmaster interface, adherence to the following guidelines must be enforced:

- using the same terminology throughout the interface,
- placing error messages and instructions in the same location as they last appeared, and
- requiring the use of the same command for the same results in similar situations.

Special attention needs to be paid to enforcing the last guideline with respect to

- arc operations,
- the relationship between network and atomic tools, and
- expansion and deletion of nodes.

2.2.3 Summary of Goals Achievement

To achieve portability of the Taskmaster interface

- a standard graphics package must be used, and
- the environment must be supported by standard operating systems, between which communication is possible.

To achieve consistency in the Taskmaster environment, the following guidelines must be enforced:

- using the same terminology throughout the interface,
- placing error messages and instructions in the same location as they last appeared, and
- requiring the use of the same command for the same results in similar situations.

Chapter 3

Achieving a Consistent User Interface

3.1 Introduction

This chapter discusses presentation and dialogue consistency issues in the Taskmaster user interface. In Section 3.2, the advantages of a uniform interface are discussed. The major areas currently lacking consistency in the Taskmaster interface — arcs, tools, and nodes, — are briefly outlined in Section 3.3, with each area discussed in detail in a separate sub-section. Resolving arc discrepancies is discussed in Section 3.4. The discussion about the forced inconsistent use of tools is approached from two viewpoints, that of the user and that of the network editor. Both of these viewpoints are discussed in Section 3.5. The discussion of the user's viewpoint is first, due to the user's viewpoint impacting the network editor's viewpoint. The lack of a uniform representation of nodes is discussed in Section 3.6. As well as rectifying the major inconsistencies in the Taskmaster interface, other areas in the Taskmaster interface have also been enhanced. These enhancements are discussed in Section 3.7.

3.2 Consistency

“The advantages of consistency lie in facilitating generalizations by the user, who having learned some parts of the system can then infer others.” [PAYN86]. Although Payne and Green are referring to the consistency of commands in a command language, this statement can be applied to commands in any user interface. An example of a consistent command in the Taskmaster environment is the view node operation. Once the user has used this operation to view the description of an atomic node, the operation necessary to view the description of a super node is easily inferred.

Consistency can be applied to the presentation of the interface, as well as the commands offered by the interface. For example, in Taskmaster the “Exit” button always appears in the upper left hand corner of each window. Another example is the graphical depiction of the nodes. An unspecified node, whether it is an atomic or super node, is grey while a specified node is black.

There is general agreement that consistency in a user interface is a goal for which to strive [SCHN87], [UHLI88], [LEWI89], [BERR88], [PERL86b], [PAYN86]. Although a precise definition for consistency is elusive, guidelines for achieving consistency do exist. These guidelines include using the same terminology throughout the interface, placing error messages and instructions in the same location each time they are displayed, and requiring the use of the same command for the same results in similar situations.

Grudin [GRUD89] cautions against the notion of consistency when applied without regard to other interface issues, such as ease of use. He claims that although consistency may help ease of learning, it may hinder ease of use. He is also concerned about creating a user interface that is consistent to the designers, but not the users – inappropriate consistency. An example he gives involves the decision made regarding the printing of folders (directories). When the print

command is executed for a folder, two choices are available: printing a hard copy of the listing of the folder; or printing a hard copy of all the documents in the folder. The first choice is preferred by the designers of the system, while the second is more appropriate to the users [GRUD89].

Both of the above issues – achieving consistency with a loss to ease of use and achieving inappropriate consistency – have been considered while deciding how to integrate enhancements into the Taskmaster interface. Ease of use has not been sacrificed. In fact, the system is more flexible and easier to use, as is explained below in the descriptions of the changes made. None of the changes involve inappropriate consistency. All changes are for the benefit of the user; the difficulty or ease of implementing the changes has played no role in deciding the best way to achieve this goal.

3.3 Inconsistency in the Taskmaster.1 Interface

3.3.1 Terminology

Recall the following terminology from Section 2.1.1. A tool is a program which performs a single high-level operation. There are two types of tools: atomic and network. Atomic tools, as the name implies, are single units and cannot be decomposed. A network tool is a tool composed of many other tools. These are created by the user and stored in the database for future use. There are also two types of nodes in the Taskmaster environment, atomic and super, both of which can be specified or unspecified. A specified atomic node represents a tool from the database. An unspecified atomic node is a place holder in the network topology. A super node represents a subnetwork. If all nodes in that subnetwork are specified, then the super node is specified. If at least one node in that subnetwork is unspecified, then the super node is

unspecified. A node that is not included in the subnetwork of a given super node is considered external to that super node and is referred to as an external node. A node in the subnetwork of a super node is correspondingly termed an internal node.

Note that network tools and super nodes are closely related. A super node represents a subnetwork of nodes, which may or may not have tools attached to them. A network tool, once created, is also a subnetwork of nodes, but in this case each node has a tool attached to it. The major difference between super nodes and network tools lies in their presentation to the user. The user views a super node as a subnetwork of nodes. The user can perform any operations on the super node that are applicable (see Figure 2.5). A network tool, on the other hand, is viewed as an atomic node. Only the operations allowed on specified atomic nodes are permitted. Note that the expand and explode operations allowed on super nodes are not allowed on network tools.

3.3.2 The Inconsistencies in the Taskmaster.1 User Interface

The three major areas of the Taskmaster.1 interface that are lacking in consistency are the three main objects presented to the user: arcs, tools, and nodes. First, arc operations cannot be performed on an arc that has a super node at an end. Both the input and output ends of the arc must be atomic nodes. This conflicts with the purpose of the arcs: indicating data flow. Second, atomic tools are found in the tools database and the user can attach these tools to nodes with the specify node operation. When the user creates a network tool, it is stored as a network. Retrieving the network tool is accomplished with a different operation than that of the one used to retrieve an atomic tool; once the retrieval is complete, the node representing the network tool is a super node and is treated as such. This contradicts the idea that a tool that can be retrieved by the user from the database without worrying about the tool's underlying

representation. Third, atomic and super nodes are treated differently, even in cases when it is natural to treat them the same (and makes sense to do so). This can be seen with the expand and delete node operations. Neither of these operations, however, can be applied to a super node in Taskmaster.1.

3.4 The Consistency of Arcs

As previously mentioned, Taskmaster.1 does not allow arc operations to be performed on arcs with a super node at either end. The primary purpose of arcs is to indicate data flow, and data flow can occur between super and atomic nodes. The ports of a super node are the ports from its subnetwork that are necessary for communication with the network external to the super node (see Section 3.6.1 for a more detailed discussion). Effectively, the super node does have ports, and communication can exist between the super node and its surrounding network.

In order to allow arc operations to be performed on arcs with a super node at either end, several issues must be resolved:

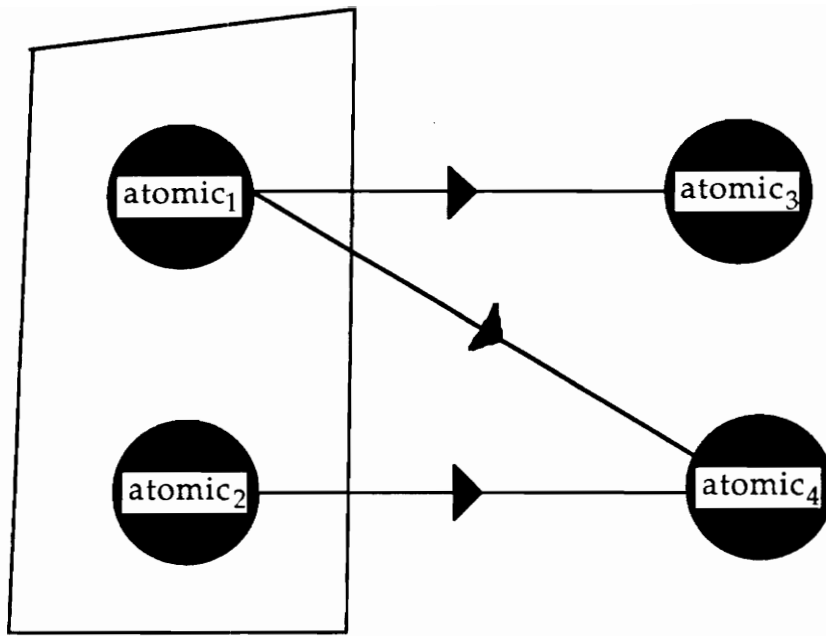
- When creating an arc with a super node at an end, to which internal node of the super node is the arc attached?
- When specifying a node that has a super node at an end, which port are available for establishing communication?
- When creating a super node with the collapse operation, what happens to the arcs that are connected such that one node is internal to the super node and the other is external to the super node?
- If a node, whether atomic or super, has arcs before expansion is applied, what happens to these arcs after expansion is complete?

- When exploding a super node, what happens to the arcs of the super node?

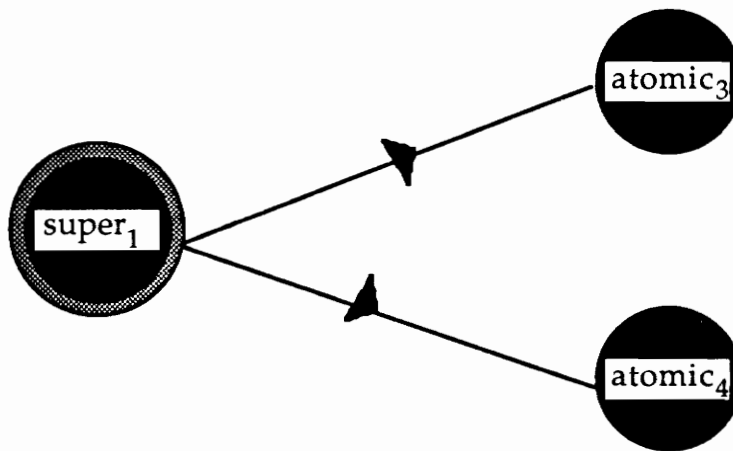
Further explanation of these questions and solutions are discussed in detail in the following sections.

3.4.1 Consistency in Arc Creation

Both super and atomic nodes have ports, and therefore both can have data flowing to or from them. Allowing arc creation involving super nodes raises some important issues and questions. When the user indicates that an arc has a super node at one end, to which internal node is the arc actually attached? One possibility is to all the internal nodes of the super node. Graphically, this is depicted as one arc between the external node and the super node. Unfortunately, this leads to ambiguity when selecting an arc for an arc operation. For example, the network shown in Figure 3.1b has a super node `super1` that represents atomic nodes `atomic1` and `atomic2`, both of which have arcs to `atomic4`, as is shown in Figure 3.1a. Graphically there is one arc between `super1` and `atomic4`, although `atomic4` actually has established communication paths with both `atomic1` and `atomic2`. When the user selects the arc between `super1` and `atomic4` in Figure 3.1b, which arc is really being selected? The one between `atomic1` and `atomic4`, or the one between `atomic2` and `atomic4`? Another possibility is attaching the arc to any internal node that supplies the super node with at least one port. Again, this leads to the same type ambiguity mentioned above when selecting an arc for manipulation. A third possibility is attaching the arc to the super node itself. This third option appears to be the more preferable because it does not have the ambiguity that is present in the other options and helps achieve the goal of consistency. In Taskmaster.2, when the user creates an arc with a super node at an end, the arc is attached to the super node.



(a)



(b)

Figure 3.1
Collapsing with Arc Preservation

3.4.2 Consistency in Arc Specification

Another important question to be answered concerns arc specification. Specifying an arc allows the user to define the communication between the nodes at the ends of the arc by identifying which ports from the output node supply data to the ports of the input node. When specifying an arc that has a super node at an end, which ports are made available for establishing communication with the super node? Two alternatives present themselves as solutions to this question: the ports of the super node as supplied by its subnetwork (see Section 3.6.1 for a complete discussion on the ports of a super node); or all of the ports from the internal nodes of the super node. When specifying an arc that has atomic nodes at both ends, the ports of the atomic nodes are the ports presented for specification. To support consistency, the first option is the one implemented in Taskmaster. Just as the ports of an atomic node are presented to the user during arc specification, so are the ports of a super node.

3.4.3 Maintaining Arc Consistency: Ramifications on the Collapse Operation

Allowing arc operations on arcs attached to super nodes impacts the collapse operation. In particular, when creating a super node with the collapse operation, what happens to the arcs that are connected such that one node is internal to the super node and the other is external to the super node? There are four alternatives:

- leave the arcs attached to their original nodes,
- delete the arcs between the original nodes,
- delete the arcs between the original nodes and for each arc, create a new arc between the external node and the new super node, or
- delete the arcs between the original nodes and for each arc, add the external node to a set

of nodes, N . For each node n in the set N , create an arc between n and the new super node.

Figure 3.1a shows a network with atomic nodes `atomic1`, `atomic2`, `atomic3`, and `atomic4` before applying the collapse operation. After applying the collapse operation to nodes `atomic1` and `atomic2`, the network is displayed as shown in Figure 3.1b, with super node `super1` representing the subnetwork containing nodes `atomic1` and `atomic2`. Graphically, there is one arc between `atomic4` and `super1` in Figure 3.1b. Leaving the arcs attached to their original nodes when a super node is created causes confusion when performing arc operations. In particular these operations are ambiguous when applied to the arc between `atomic4` and `super1` in Figure 3.1b. Which arc is the one being selected, the one that connects `atomic1` to `atomic4` or the one that connects `atomic2` to `atomic4`? Using the second alternative, deleting the arcs between the original nodes, forces the user to recreate any arcs required for communication between the super node's underlying subnetwork and the network external to the super node. The third alternative also creates ambiguity. If there are two nodes internal to the super node with arcs to the same external node, such as `atomic1` and `atomic2` to `atomic4` in Figure 3.1, two arcs will be created between the external node and the super node. These problems are avoided in the fourth alternative. Only one arc is created between the super node and any given external node. The created arcs are attached to the external nodes and the super node, not the internal nodes of the super node. The fourth alternative preserves the communication paths while removing any ambiguity during arc specification, and is the one employed by Taskmaster.2.

3.4.4 Maintaining Arc Consistency: Ramifications on the Expand Operation

Both atomic and super nodes can be expanded using Taskmaster.2. Recall that expanding an atomic node allows the user to create a subnetwork to be associated with that node. Expanding a super node allows the user to modify the subnetwork associated with the super node. Allowing arc operations to be applied to arcs attached to super nodes impacts the expand

operation. If a node, whether atomic or super, has arcs before expansion is applied, what happens to these arcs after expansion is complete? When expanding an atomic node in Taskmaster.1, the arcs are deleted. This option was one of three possibilities considered for the expansion of atomic nodes in Taskmaster.2:

- delete the arcs,
- delete the arcs and for every arc create an arc between each internal node of the new super node and the external node, or
- delete the arcs and for every arc create an arc between the new super node and the external node.

As mentioned above during the discussion of the ramifications of maintaining arc consistency on the collapse operation, deleting the arcs forces the user to recreate any arcs required for communication between the super node's underlying subnetwork and the network external to the super node. Using the second alternative creates the same ambiguity mentioned in the discussion of arc creation above. The third option maintains a uniform relationship between arcs and nodes while avoiding ambiguity, and is the one implemented in Taskmaster.2. On completion of the expand operation applied to an atomic node, all arcs attached to the pre-expanded atomic node are re-created with one end at the new super node and the other at the external node.

When expanding a super node, an operation not allowed by Taskmaster.1, the same question arises: what happens to the arcs between the super node and the surrounding network? In this case, the super node is already in existence, and its arcs can be preserved.

3.4.5 Maintaining Arc Consistency: Ramifications on the Explode Operation

Recall that applying the explode operation to a super node replaces the super node with its

underlying subnetwork. Allowing arc operations on arcs attached to super nodes impacts the explode operation. When exploding a super node, what happens to the arcs of the super node? Again, four options exist:

- delete the arcs of the super node,
- delete the arcs of the super node and for every arc, create an arc between each of the internal nodes of the super node and the external node,
- delete the arcs of the super node and recreate any arcs between the internal and external nodes of the super node that existed before the creation of the super node, or
- delete the arcs of the super node and for each specified arc of the super node, create an arc between the external node and the internal node that contains the port connected to the external node's port.

As mentioned earlier, simply deleting the arcs results in the the loss of established communication paths. Creating an arc between each of the internal nodes and the external node for each arc results in an excessive amount of arcs, many of which are not desired by the user. Recreating arcs that existed before the creation of the super node is appropriate, but doesn't go far enough. This is true of the fourth alternative also. Therefore, the solution employed by Taskmaster.2 is the combination of the third and fourth alternatives. That is, the arcs of the super node are deleted, any arcs existing before the creation of the super node are re-created, and for each specified arc of the super node, an arc is created between the external node and the internal node of the super node that contains the port connected to the external node's port.

3.4.6 Maintaining Arc Consistency: Ramifications on Tool Consistency

Maintaining arc consistency decreases the differences between atomic and super nodes, which simplifies maintaining tool consistency. As is discussed below in Section 3.5, a node representing

a network tool is presented to the user as a node representing an atomic tool: as an atomic node. To the network editor, the node represents an underlying subnetwork, and is treated as such. By allowing arc operations on arcs attached to super nodes (which also represent underlying subnetworks), the implementation of a uniform user interface with respect to tools is simplified.

3.5 The Consistency of Tools

As mentioned previously, network tools and atomic tools are treated differently in Taskmaster.1. To access a network tool, the user must select the “attach network tool” operation, instead of the “specify node” operation used for retrieving an atomic tool. Once the subnetwork of the network tool has been retrieved and attached to a node, the node is a super node and is treated as such; the node has no special status as a network tool once the attach network tool operation is complete. This presentation of a network tool differs from the presentation of an atomic tool and can cause confusion. To support consistency in the tools of Taskmaster.2, network tools are treated the same as atomic tools and the nodes to which they are attached are treated the same as the nodes to which atomic tools are attached. Several questions must be answered in achieving this uniformity with regard to both the user interface and the implementation:

- How is a network tool retrieved by the user?
- Once a network tool has been retrieved and attached to a node, how is that node seen by the user?
- How are the descriptions for the network tool acquired?
- How are the ports of the network tool determined?
- Where in the database should the network tool be stored?
- How is the network tool stored in the database?

The solutions to these questions are discussed in Section 3.5.1. These solutions impact the implementation of tool consistency. This impact, as well as the issues it raises, is discussed in Section 3.5.2.

Recall from Section 3.3.1 that super nodes and network tools are closely related. Although they both represent subnetworks, their presentation to the user is different. Although the internal representation of a network tool is the same as that of a super node, a node with a network tool attached is presented to the user as an atomic node.

3.5.1 The User Interface: Tools

In order for the user to retrieve a network tool, it must first be created. The method used to retrieve a network tool impacts the method used to create a network tool. Section 3.5.1.1 discusses the method used to retrieve a network tool in Taskmaster.2, as well as the status of a node with a network tool attached. The creation of a network tool from the user's viewpoint is discussed in Section 3.5.1.2.

3.5.1.1 The User Interface: Network Tool Retrieval

How does the user retrieve a network tool? The attach network tool operation is used in Taskmaster.1. To achieve consistency in the user interface among tools, in Taskmaster.2 retrieving a network tool is accomplished with the same operation as that used to retrieve an atomic tool: the specify node operation. Allowing the user to retrieve a network tool with the specify node operation has an impact on the storing of a network tool, both from the viewpoint of the user, as discussed in Section 3.5.1.2, and from an implementation standpoint, as discussed in Section 3.5.2.1.

Once a network tool has been retrieved and attached to a node, how is that node seen by the

user? Using Taskmaster.1, the user sees the node as a super node, and can perform operations on the node as a super node. In Taskmaster.2 a node representing a tool, regardless of whether the tool is an atomic tool or a network tool, is graphically depicted as an atomic node. The the user can perform operations on the node as an atomic node, i.e. any operations allowed on an atomic node can be applied to this node. The operations allowed on nodes depends on whether the node is atomic or super, and whether the node is specified or unspecified, as can be seen Figure 2.5.

In both versions of Taskmaster, retrieving the subnetwork of a network tool is accomplished with the attach network tool operation. The user selects a node to which the subnetwork is attached, and that node consequently becomes a super node.

3.5.1.2 The User Interface: Network Tool Creation ---

Indicating the subnetwork to be designated as a network tool is accomplished in the same manner in both Taskmaster.1 and Taskmaster.2: a polygon is drawn around the nodes that the user wishes to include in the subnetwork. Acquiring the display name for the network tool is also the same in both versions of Taskmaster: the user is prompted for the name.

Network Tool Descriptions: Taskmaster.1 does not allow a network tool to have a description associated with it. In an effort to minimize the differences between network and atomic tools, Taskmaster.2 permits a network tool to have a description. This necessitates the acquiring of a description and the subsequent recognition of two alternatives: the user can supply the description; or the network editor somehow infers a description from the descriptions of the tools in the network tool's subnetwork. Because the second alternative can result in a "hacked" description, the user is prompted to supply the network tool's description. Another factor to consider is the short description of the tool that appears in the tools menu during node specification. Obtaining this description can be achieved in one of two ways. Either the

network editor creates the short description from the long description given by the user, or the user supplies a short one line description of the tool in addition to the long description. There is no general algorithm that can be used by the network editor to create a short description from the long description, therefore the user is also prompted to supply a short, one line description of the network tool that will be used in the menus during node specification.

Network Tool Ports: Another set of issues surrounding network tools is the handling of communication ports. In Taskmaster.1, the ports of a network tool are determined by the network editor at the time of attaching a network tool. Allowing the user to treat a network tool as an atomic tool requires the determination of the network tool's port during network tool creation. Determining the network tools' ports can be achieved in one of three ways:

- the network editor automatically determines the ports,
- the user indicates the ports, or
- the network editor determines default ports and the user can modify these defaults.

The first alternative is very strict and takes the control of network tool creation out of the hands of the user. The second alternative provides no guidance to the user as to which ports are appropriate for inclusion as ports of the network tool. The third alternative provides guidance to the user without being restrictive or taking control away from the user, and is the one implemented in Taskmaster.2. Using this alternative, however, leads to another question. Which ports constitute the external interface of a newly created network tool? The choices here are the same as for determining which ports are the ports of a super node, discussed in Section 3.6.1, and the same solution is used. The default ports are the ports from the selected subnetwork that are necessary to maintain functionality.

Network Tools in the Tools Database: In order to allow the user to retrieve a network tool using

the specify node operation as if it were a conventional tool, the network tool must be stored in the tools database. How this is accomplished is (and should be) transparent to the user and is discussed in Section 3.5.2.1. However, the location in the tools database for the network tool is not transparent to the user. When using the specify node operation, the user is led through a series of menus that reflect a hierarchy of tools in the database (Figure 3.2). A network tool must be stored somewhere within this hierarchy. Does the network editor determine that location or does the user? In an effort to leave control of network tool creation in the hands of the user Taskmaster.2 allows the user to determine the hierarchical location of the network tool in the tools database hierarchy. However, allowing the user to determine the hierarchical location of the network tool raises another question. How does the user indicate that location? One alternative makes use of windows and menus used during the specify node operation. The user proceeds through the menu until reaching the menu that represents the desired location for the network tool. In Figure 3.2, the user selects "File Operations" to indicate that the network tool belongs in the same group as the "save file", "open file", "sort file" and "delete file" tools. Another alternative makes use of the graphical abilities of the network editor. A "tree" is drawn of the tools database using the nodes and arcs of the network editor, where atomic nodes represent the tools in the database and super nodes represent the possible locations in the database for storing the network tool (Figure 3.3). The tools from the database are shown as examples in order to assist the user in selecting the appropriate location for the network tool. The user indicates the location for the network tool by selecting the super node that represents the desired location for the network tool. Both the menu and tree approach allow the user to see the hierarchical structure of the database. The tree method, however, allows the user to see the entire structure at a glance, whereas the menu method requires the user to make a selection from the menu in order to see the next level of the database. The tree method allows the user to more easily select the appropriate location for

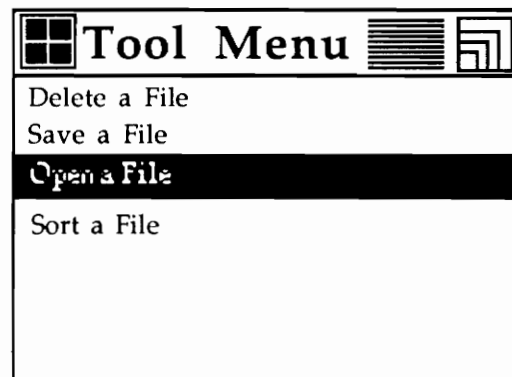
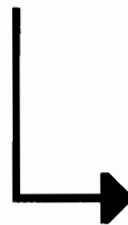
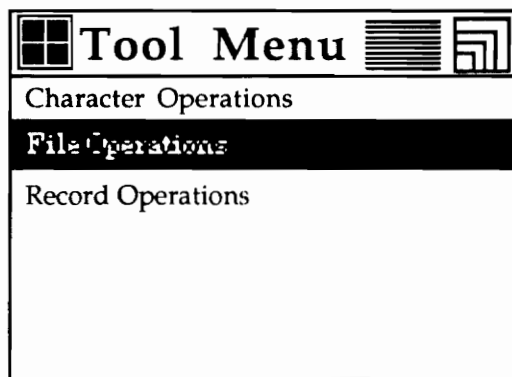


Figure 3.2
Specifying a Node

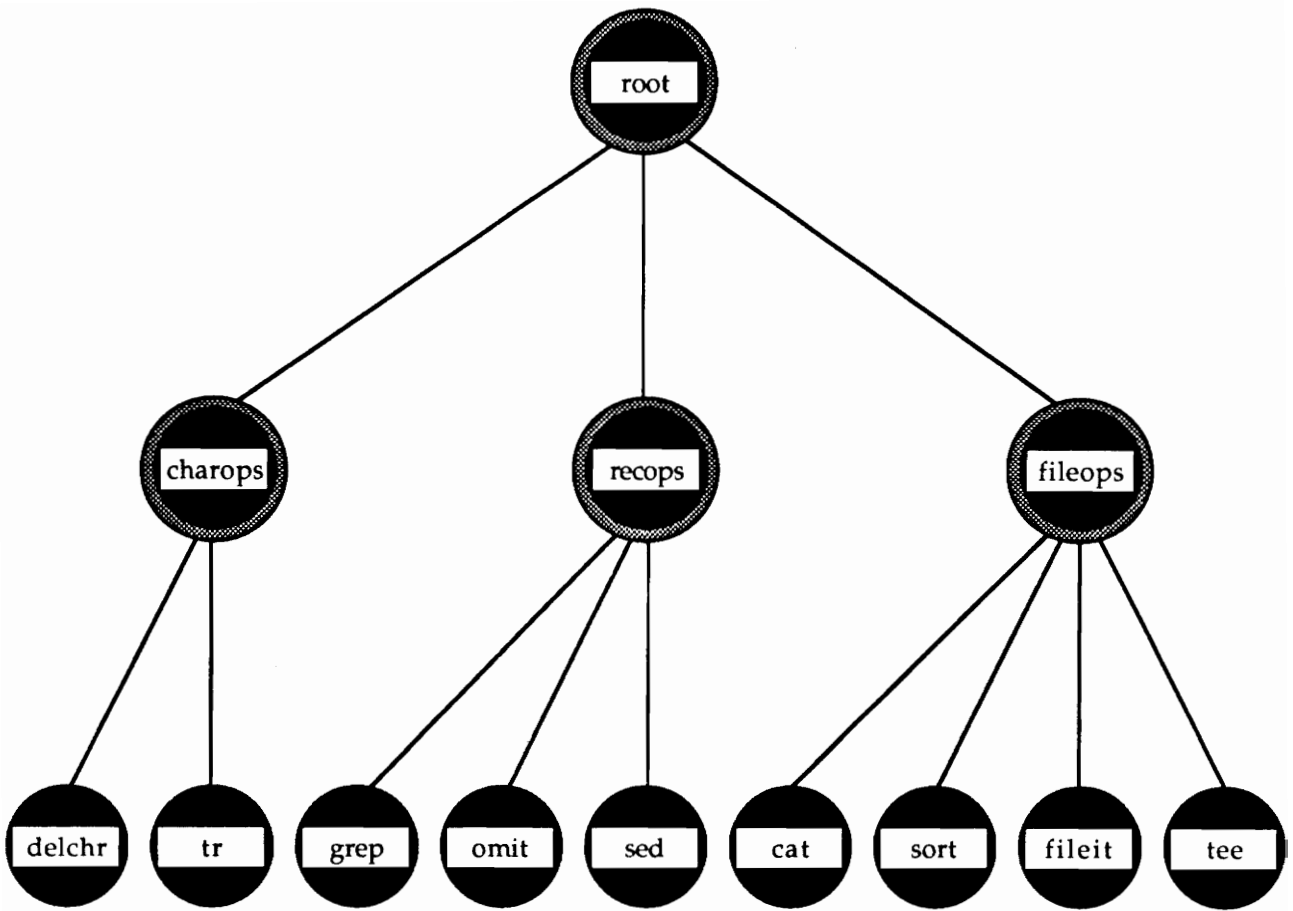


Figure 3.3
Network Display of Tools Database Hierarchy

the network tool. In addition to this advantage, the tree method presents insertion choices in a graphical manner consistent with the overall design of the Taskmaster interface. The tree method is the method employed by Taskmaster.2 for indicating the desired location in the database for the network tool.

3.5.2 The Implementation: Tool Consistency

3.5.2.1 The Implementation: Network Tool Creation

Creating the Network Tool: As in Taskmaster.1, when selecting the create network tool operation the user is prompted to draw a polygon to indicate the subnetwork of the network tool and to supply the display name of the network tool. As mentioned earlier, Taskmaster.2 allows the user to supply a long and short description for the network tool; the network editor prompts the user to supply these descriptions. The ports of the network tool must also be determined. The network editor displays the complete set of ports from the nodes in the subnetwork to the user, with the default ports highlighted. The user can modify these default ports by deleting any of the defaults and by selecting others.

Storing the Network Tool: The network tool is fully defined after the subnetwork, long and short descriptions, display name and ports have been determined. The only task left is that of inserting the network tool into the tools database. As mentioned earlier, the user indicates where in the tools database hierarchy the network tool belongs. The network editor draws a tree of the hierarchy by creating a network that represents the hierarchy (Figure 3.3). By presenting the hierarchy to the user in this manner, other network operations can be applied. Although no functional modification to the hierarchy is allowed, the user can use the operations that alter the view of a network: i.e. the zoom in, zoom out, pan, view node, and move node. The leaves of the tree represent the tools that are already in the tools database

and are depicted graphically as atomic nodes. These are shown as examples in order to assist the user in determining the appropriate location for the network tool. The internal nodes of the tree represent the higher levels of the tools database and are depicted graphically as super nodes. The arcs of the network do not represent communication paths and are shown simply to indicate the connection between higher level nodes and the leaves and lower level nodes. The user selects the super node that represents the desired location for the network tool.

After the user has indicated the location for the network tool in the tools database hierarchy, the network editor must store the network tool in the database. How is the network tool stored in the database? One option is storing the entire subnetwork in the database along with the information necessary for the node specification process. A second option stores the information necessary for the node specification process in the database, along with a file name. The file name is the name of the file where the network tool's subnetwork has been stored. The option chosen does not have an impact on the user's viewpoint in any way. Because of this, the second option is the one employed by Taskmaster.2 due to the simplicity of the implementation. Another advantage to storing the subnetwork in a file involves the size of the database. If the entire subnetwork of a network tool is stored in the database every time a network tool is created, the database becomes large and unwieldy [FAN88].

3.5.2.2 The Implementation: Network Tool Retrieval

When the user selects the specify node operation, the tool chosen may be an atomic tool or a network tool, although the type of the tool is transparent to the user. If an atomic tool is chosen, the network editor proceeds as in Taskmaster.1. If a network tool is chosen, the network editor must retrieve the subnetwork from the file given in the database. The subnetwork is attached to the indicated node and the node appears to the user as an atomic node. Allowing the network editor to present the network tool to the user as an atomic tool is simplified by

maintaining arc consistency. A node with a network tool attached to it appears as an atomic node to the user, and therefore the user expects that arcs can be attached to this node. To the network editor, this node acts as a network node in many ways, and Taskmaster.1 does not allow the user to perform arc operations on arcs attached to super nodes.

As mentioned earlier, once a network tool is retrieved and attached to a node, the network editor considers this node both an atomic node and a super node. It is an atomic node both graphically and operationally. That is, the node appears to the user as an atomic node and only those operations that can be applied to atomic nodes may be applied to it (see Figure 2.5). When applying these operations to this node, the network editor treats the node as a super node. Similar to a super node, the node represents a subnetwork. The port representation of the node within the network editor is the same as the representation for super nodes. Recall, however, that the ports of a network tool are determined in a different manner than those of a super node. The user determines the ports of a network tool at the time of network tool creation and the network editor determines the ports of a super node at the time of its creation, (via either the expand or collapse operation).

The retrieval of the subnetwork of a network tool is done with the same operation in Taskmaster.2 as in Taskmaster.1, attach network tool. Once retrieved, the node to which the subnetwork is attached is a super node both graphically and operationally.

3.5.3 A Ramification of Maintaining Network Tool Consistency

If the user saves network tools during an editing session, the database is modified. At the end of the editing session the user may save the modified database. The user can also save the modified database on the remote host by supplying a file name as a response to the appropriate prompt. Allowing the user to save the modified database on the user workstation can lead to

several local databases. Because of this, at network editor invocation, the user is prompted to select a database for use during the editing session.

3.6 The Consistency of Nodes

Despite their similarities the nodes of Taskmaster.1, atomic and super, are not consistent with each other. In both versions of Taskmaster, super nodes have ports supplied using the method described below in Section 3.6.1. However, the expand and delete node operations can be applied only to atomic nodes in Taskmaster.1. Reconciling these differences is discussed in Sections 3.6.2 and 3.6.3.

3.6.1 The Ports of a Super Node

The underlying subnetwork of a super node supplies the super node with ports. Three possible alternatives are given in [RAGH88] for determining which ports of the underlying network become the ports of the representative super node:

- no external context preservation (Figure 3.5a),
- external functional context preservation (Figure 3.5b), or
- external relational context preservation (Figure 3.5c).

Figure 3.4 shows a network with a polygon drawn to indicate the underlying subnetwork of a super node. Although a network drawn in Taskmaster does not show the ports of the nodes, the ports are drawn here to simplify the following discussion. The ports are numbered to facilitate the comparison between Figure 3.4 and Figure 3.5.

Preserving no external context results in a super node with ports as shown in Figure 3.5a. These are the ports within the subnetwork that are unconnected within the subnetwork. Figure 3.5b

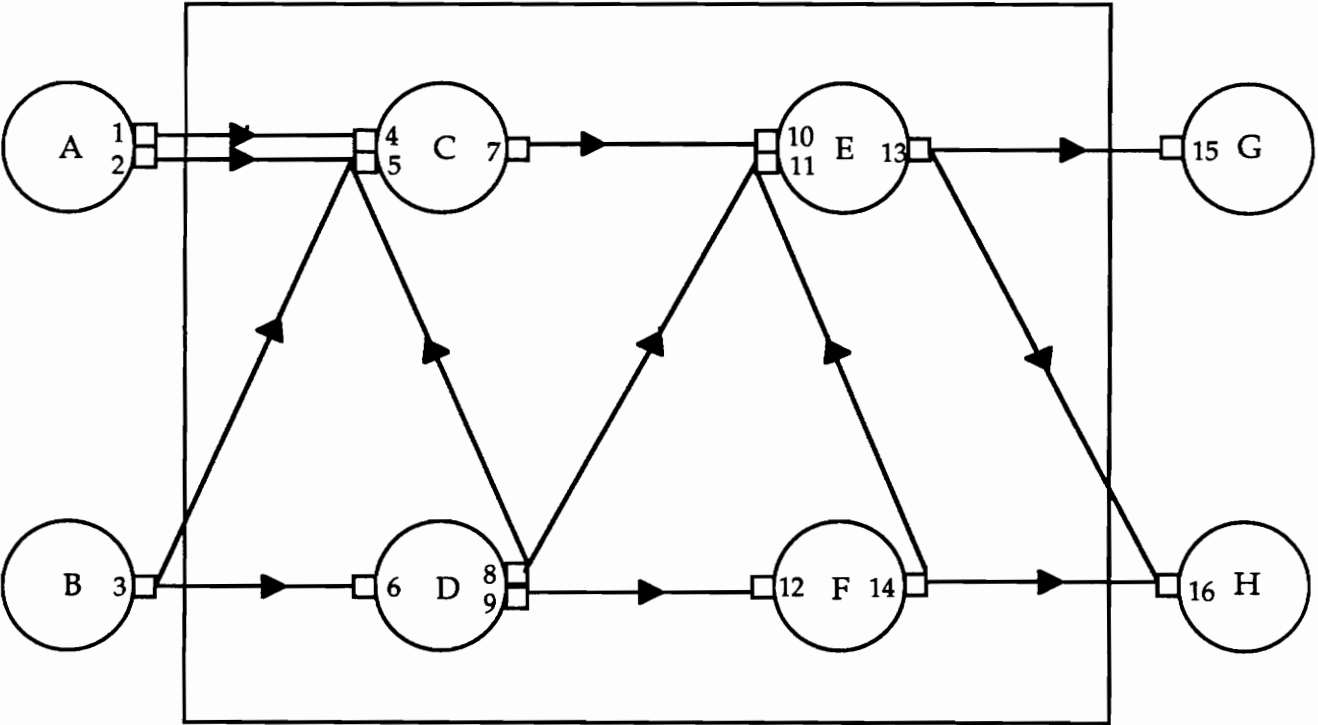


Figure 3.4
Network with the Ports Shown

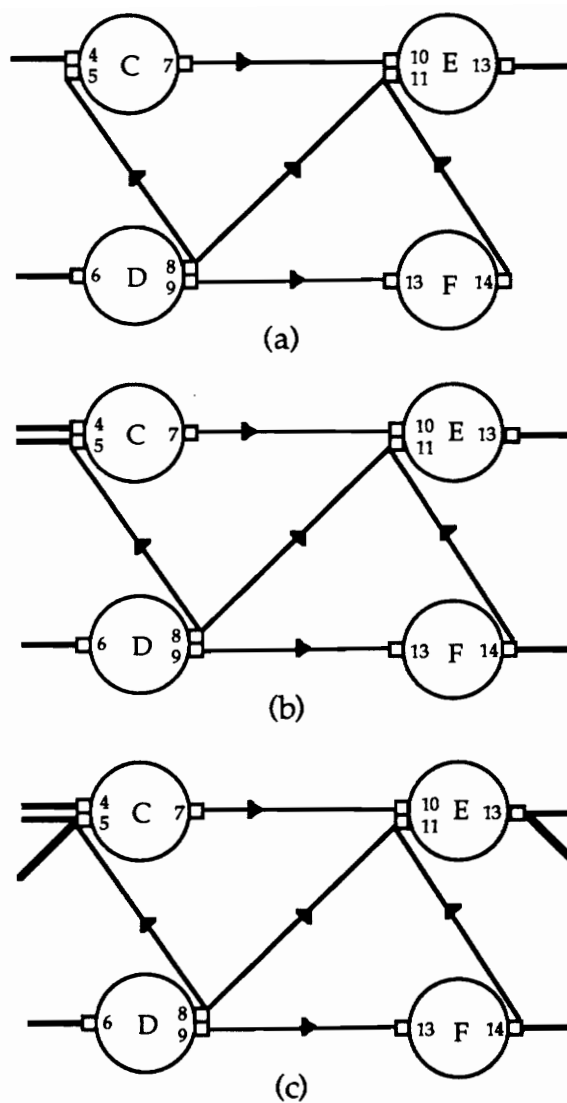


Figure 3.5
The Three Port Models

shows the results of applying external functional context preservation. These are the ports within the subnetwork that are either unconnected within the subnetwork or are connected to an external node or both. The third model, external relational context preservation, records all external context information and is very context-specific, as is shown in Figure 3.5c. The ports for this model are found by creating a port for every external link.

The model utilizing external functional context preservation is the one used in both Taskmaster.1 and Taskmaster.2. This model supports reusability at the functional level without relying on the surrounding network for this reusability.

3.6.2 Node Consistency and the Expand Operation

In Taskmaster.1 the expand operation can be applied to atomic nodes. This allows the creation of a subnetwork, but not the modification. In order to modify the subnetwork represented by a super node, the super node must be exploded. The subnetwork then becomes part of the surrounding network. This indicates that complete top-down and bottom-up task specification [ARTH88] is not allowed in Taskmaster.1. In Taskmaster.2, the functionality of the expand operation is completed. Whether expanding an atomic node or a super node, invoking the expand operation causes a new window to be presented to the user. If the node being expanded is a super node, the underlying subnetwork of the super node is displayed in the window for modification. Expanding an atomic node results in an empty window being presented to the user for network creation and modification. After exiting the expansion window, the newly created or modified subnetwork is represented by a super node.

From an implementation standpoint, adding the ability to expand a super node creates new concerns. In Taskmaster.1, the nodes in the subnetwork are not added to the complete network

until the expansion window is exited. When exiting the expansion of a super node in Taskmaster.2, the network editor must not duplicate the nodes from the subnetwork existing pre-expansion and must delete any nodes from the subnetwork deleted during modification in the expansion window.

3.6.3 Node Consistency and the Delete Node Operation

Taskmaster.1 does not allow the deletion of a super node. Taskmaster.2 has remedied this inconsistency by allowing the user to delete a super node. As when deleting an atomic node, a confirmation is requested from the user before deletion occurs. The arcs associated with the super node are deleted, all of the nodes in the underlying subnetwork are deleted, and any established communication paths are deleted as well.

3.7 Additional Enhancements to the User Interface

Several other areas of the Taskmaster.1 user interface are inconsistent. For example, upon invocation of the network editor, the user is asked whether or not to update the database via a prompt at the terminal screen. It is natural to ask this question with the same window that is used at all other times for getting a yes/no response, as is done in Taskmaster.2. As mentioned earlier, one guideline to achieving a consistent interface involves using the same terminology throughout the interface. This guideline is not followed in the Taskmaster.1 interface. Nowhere in this thesis or in the user interface is the subnetwork of a network tool referred to as a tool composite. In view of this, the attach tool composite and save tool composite operations of Taskmaster.1 are now named attach network tool and create network tool respectively. Although the node operations in the main menu of Taskmaster.1 are create node, delete node, specify node and view node, the arc operations are create arc, delete arc, specify communication

path and view communication path. These last two operations are now named specify arc and view arc respectively. Another confusing operation is the create network operation of Taskmaster.1. Selecting this operation results in prompting the user to either create a new network or open a previously saved network. Since these two operations are different from each other, they are both in the main menu of Taskmaster.2 as create new network and open network respectively. Another inconsistency in the main menu of the Taskmaster.1 is the organization of the commands. Taskmaster.2 uses the grouping presented in Section 2.1.3, as can be seen in Figure 2.8, while Taskmaster.1 uses the following grouping (the names of the operations are the names used by Taskmaster.2 to allow for easy comparison):

- Node Operations

- Create Node, Delete Node, Move Node

- Specify Node, View Node, Expand Node

- Arc Operations

- Create Arc, Delete Arc

- Specify Arc, View Arc

- Abstraction Operations

- Collapse, Explode, Save Network Tool, Attach Network Tool

- Topology Viewing Operations

- Zoom In, Zoom Out, Pan

- Network Storage and Retrieval

- Save Network on Disk, Create New Network, Open Network

- Error Recovery

- Undo Last Operation

- Network Execution

Execute Network.

Taskmaster.2 allows the user to exit any operation at any step of the operation, and also allows the user to exit any operation by typing ctrl-e. Taskmaster.1 does not supply the user with the ability to exit any operation at any step, and typing ctrl-e works in some cases but not others. In Taskmaster.2, exiting any menu, including those shown during node specification, can be achieved by either typing ctrl-e or by selecting "Exit" from the menu. When the user is prompted to type in a string, exiting can be accomplished by entering a null string or by typing ctrl-e. When presented with a window, such as the network topology window or the node viewing window, the user can exit by either clicking the mouse on the exit button or by typing ctrl-e. When supplying a description for a super node or network tool, the user can exit the operation by typing ctrl-e. In all these cases, typing ctrl-e will allow the user to exit the operation.

Once a user is familiar with the Taskmaster interface, the user may not need the detailed instructions that are provided at every step. The displaying of the instructions takes time, and an experienced user may wish to dispense with the instructions. At the invocation of Taskmaster.2's network editor the user is prompted to select either expert mode, which provides no instructions, or novice mode, which provides the detailed instructions at every step.

Another enhancement provided to assist the user is the use of menus during network saving and retrieval. Taskmaster.1 requires the user to type in a name when saving or retrieving a network. When retrieving a network in Taskmaster.2, the user is presented with a menu containing a list of the previously saved networks, thus lifting the requirement that the user remember the exact names of all previously saved networks. When storing a network in Taskmaster.2, a user is shown a list of the previously saved networks and prompted to type in a

network name in order to prevent overwriting a previously saved network. These menus are used during network tool creation and network tool network retrieval as well.

Chapter 4

The Port

4.1 Introduction

To achieve the goal of portability, Taskmaster.2 employs a different graphics package and operating system than those of Taskmaster.1. As a side effect of changing the graphics package and operating system, the hardware that Taskmaster.2 uses is different than that of Taskmaster.1. The implementation of Taskmaster.2's network editor uses the X Windowing System Version 11 (X11) on a Macintosh II running A/UX, while Taskmaster.1 uses GKS level 0b (GKS0b) on a VAXstation I running MicroVMS. The execution monitor of both versions is implemented on a machine with the UNIX operating system Ultrix-32: Taskmaster.2 on a VAXstation 2000 and Taskmaster.1 on a VAX 11/785. These variations lead to many differences in the implementation of the two versions of Taskmaster. The change in graphics package has the greatest impact, due to the dissimilarity of the two packages. The two packages used are discussed in Section 4.2 with emphasis on the differing elements of the packages that affect the implementation of Taskmaster. As discussed in Section 4.3 the

variation in respective graphics packages leads to differences in the Taskmaster implementation, some of which are transparent to the user, and others visible to the user. Changing the operating system of the network editor has many advantages; many of them stem from the fact that both executable components of Taskmaster.2 (the network editor and the execution monitor) will run under the UNIX operating system. The differences in implementation due to the change in operating system are discussed in Section 4.4. The difference in hardware supporting the network editor associated with the two versions has caused variations in the implementations, all of which are visible to the user. These variations are discussed in Section 4.5.

4.2 GKS0b versus X11

The portability of applications using GKS is greatly affected by the different levels of GKS. Within a given version the levels in GKS, 0a to 2c, represent levels of functionality, with a given level incorporating all the functionality of the levels below it and then adding some of its own. This means that an application implemented at a given level of GKS can be implemented at any higher level, i.e. GKS has upwards level compatibility. This is not the case with the X Windowing System. An application using a given version of X is portable to any system hosting the same version of the X Windowing System. For both GKS and X, there is no guarantee of version compatibility, either from an older version to a newer version (upwards compatibility) or vice-versa (downwards compatibility). However, X11 provides a set of routines to make it easier to port applications from X10 to X11, while in GKS the upwards version compatibility depends on the implementation. Version compatibility is not to be confused with the level compatibility of GKS mentioned above. The packages used by Taskmaster.1 and Taskmaster.2, i.e. level 0b of GKS (also referred to as GKS0b in this thesis)

and X11 respectively, are discussed in the following sections. See [ENDE84] and [VAXG84] for more detailed discussions about GKS and [NYE88a] and [NYE88b] for more detailed discussions about the X Windowing System. Section 4.2.1 discusses some background for both packages. Differences concerning the control of windows and their contents, are discussed in Section 4.2.2. The greatest differences between the two packages concern the input and output capabilities, discussed in Sections 4.2.3 and 4.2.4 respectively.

4.2.1 General Background

4.2.1.1 The X Windowing System

The X Windowing System can run on multiple machines (connected via a communication network) due to its client-server implementation. The server software controls the display and the input devices, while the application programs are the clients. The two terms, application program and client, are used interchangeably in this thesis. The server and clients communicate by sending packets of information across the network; these packets are interpreted by Xlib, the C-language interface to X. One type of packet, an event, "... is a packet of information that is generated by the server when certain actions occur, and is queued for later use by one or more clients." [NYE88a, p35]. Events are the principle method by which clients get information. There are different types of events, including types containing information about input, windows and other clients. Event management is handled with event loops within the application program. These loops check for key presses, mouse movement and clicking, and window events. On receiving an event, the action appropriate to that event is taken.

4.2.1.2 GKS0b

During the execution of a program using the GKS0b graphics package, GKS0b is in a precisely defined state whose elements consist of the operating state and the values of variables in state

lists present in a GKS0b system. The state can be changed by either directly calling certain functions or through side effects of function calls. Depending on the operating state, calling a specific GKS0b function may or may not be allowed [ENDE84, p45].

4.2.2 Windows and Their Contents

The GKS0b graphics package manages the windows and events for the programmer, while the X Window System gives the control to the programmer. Switching this control from the graphics package to the application program hindered the port of Taskmaster, yet helped with achieving the goal of consistency. How the control switch aided in achieving the goal of consistency is explained in section 4.2.3. The control switch hinders the port, however, by requiring that the application program handle window management. In turn, this requirement adds additional complexity to the application software. Window management includes refreshing the contents of windows whenever necessary. For example, when a window is mapped to the screen, i.e becomes visible, it can obscure other windows that are already mapped. Unmapping this window causes the other windows to become visible, and thereby, requires the contents of these other windows to be refreshed by the programmer.

The term “window” has different meanings in X11 and GKS0b. These different meaning do not themselves lead to difficulty in porting Taskmaster from GKS0b to X11, but they are explained here to clear up any confusion caused by these different meanings. In X11, “a window is a rectangular area that works in several ways like a miniature screen.” [NYE88b, p19] Windows appear on the screen and can be manipulated by the user. The different windows visible at one time can each be reflecting a different activity. In GKS0b, a window is “a predefined part of a virtual space.” [ENDE84, p27]. It is a rectangular region used by the application program for specifying coordinates for various operations. An X11 type window can be simulated in GKS0b

by treating one screen as multiple workstations. In this thesis, the term “window” refers to the concept of an X11 window, unless specifically noted otherwise.

4.2.3 Input Capabilities

Input in X11 is controlled through the use of events, while in GKS0b getting input from the user is achieved through the use of several input functions. Achieving the goal of consistency is aided by the events of X11. These events allow the application program to get input from several input devices at one time, while in GKS0b only one input device can supply input at a time. The advantages of this are explained in more detail in Section 4.3.3.

The user of an application program can generate input events in X11 by moving the mouse, clicking the mouse button, or pressing a key. These events are queued with any other generated events, and the client can then access them from the queue. In this way, the client can access any kind of the input at any time. Some input is not appropriate at certain times, e.g. a mouse click when the client has requested a string from the user. The client can choose to ignore the events generated by such input or issue an error message. Note, however, that all events are added to the queue and the *client* can use or ignore them as appropriate for the application.

As mentioned above, GKS0b handles input in a very different manner. Input is requested by the application from a specific logical input device. A logical input device is an abstraction of one or more physical input devices, e.g. a keyboard and a number pad; each logical input device belongs to an input class. An input class groups logical input devices together based on their functions. The different input classes of GKS0b provide different forms of input:

- Locator – provides the position given by the user positioning a locator input device, such as a mouse;

- Stroke – provides a sequence of positions given by the user positioning a locator input device at several different locations;
- Valuator – provides a real number supplied by the user with the use of a valuator input device, such as a number pad;
- Choice – provides a non-negative integer which represents a selection from a number of choices supplied by the user choosing one possibility from a choice input device, such as a menu; and
- String – provides a character string entered by the user from a string input device, such as a alphanumeric keyboard [ENDE84, p34].

Only one specific logical input device can supply information at a time. This prohibits the application from allowing the user to supply input in more than one way. For example, if the user is presented with a menu, a choice must be made from the menu before any other processing can take place.

When requesting input using GKS0b, the application can require that the input supplied the user be immediately echoed back to the user. This capability does not exist in X11, although it can be simulated by the application program. The lack of echo capability in X11 has a significant impact on the port, because the simulation of echoing is slow. The alternatives and the method used in Taskmaster.2 are discussed in Section 4.3.2.

4.2.4 Output Capabilities

4.2.4.1 Coordinates

X11 and GKS0b have different coordinate systems, which also impacts the port of Taskmaster by requiring a revision of the calculations to support graphical output. In X11, each window has its own xy axis with the origin in the upper left hand corner of the window; additionally, the

coordinate units are represented by pixels. An xy axis is used in GKS0b also, but with the origin in the lower left hand corner of each window. The application program determines the units with the use of world coordinates in GKS0b. When requesting graphical output, the application program indicates in which window the output is to be displayed, and the coordinates given are relative to the origin of that window. For a complete discussion and explanation of the different coordinate sets in GKS, see [ENDE84]. Recall from Section 4.2.2 that GKS0b can simulate X11 type windows by treating one screen as multiple workstations.

4.2.4.2 Setting Graphics Attributes

When creating a picture on the screen, a graphics package requires values for certain attributes in order to determine how the picture is to be drawn. For example, to draw a line, the graphics package need to know the line width, and the line style, e.g. solid or dashed. The values for such attributes can be supplied by the application program, or provided through initial default values.

X11 allows the client to set attribute values through the use of a data record called a graphics context. In GKS0b, each graphics function has its own set of attributes. These attributes can be set calling a specific function for an attribute, such as changing the line width attribute via a "set line width" function. This difference mandates a revision in the setting of the graphics attributes in Taskmaster. Instead of calling a function every time an attribute must be changed, as in Taskmaster.1, many different graphics contexts are created in Taskmaster.2, each with different attribute settings.

A graphics context (GC) contains the values for all attributes that X11 requires in order to correctly draw an image on the screen. When invoking a graphics function, the application program indicates which attribute values are to be used by passing a GC to the function. The

application program can have several GCs, each with different attribute values. This allows the application program to quickly change attribute values by passing a different GC.

GKS0b has the notion of bundles and bundle tables for setting graphics attributes, as well as changing the attribute values through function calls. A bundle contains attribute values for a specific function, and a bundle table stores many bundles. Using these bundle tables allows the application program to change attribute values by changing the index that the graphics package used to access the table. In GKS0b, the bundle tables are predefined and cannot be changed by the application program. If none of the bundles in the table contains the needed attribute values, the application program can indicate that the bundle table is not to be used, and then set the attribute values using the previous method mentioned above – calling functions to set the values.

4.2.4.3 Text

Although the two different packages have very different text capabilities, only the capability to set the size and location of the text has an impact on the implementation of Taskmaster. In X11, each graphics context has a fixed size font associated with it, while in GKS0b a font is simply the shape of the characters in the text and the size is set independently. The fixed size fonts used in X11 have an impact on the magnification capabilities of Taskmaster, further discussed in Section 4.3.1. GKS0b allows the application program to indicate whether the text is to be drawn centered, or left or right justified at the given location. X11 does not have this capability, and the application program must perform the necessary calculations to achieve this affect.

Whenever an X function for drawing text called, the font used in drawing the text is the font in the passed graphics context (GC). The size of the text is determined by the font; if a different

size is desired, a different font is used. The size of text drawn in GKS0b is determined by two factors: the value of a text size attribute, and the world window boundaries. The world window boundaries can be changed to create a magnification effect, as is explained in Section 4.2.4.5.

4.2.4.4 Fill

In both X11 and GKS0b areas on the screen can be filled with either a pattern or one color by specifying a “fill style”, patterned or solid, and the pattern to use. The difference, which has an impact on the port, is in pattern creation. In GKS0b patterns are retrieved from a pre-defined pattern table, while X11 allows the application program to create its own patterns. For information on pattern creation in X11, see [NYE88a, p128-33].

In X11, the graphics context (GC) has a “fill style” attribute that determines whether the area is to be filled with a pattern or a solid color. The GC also contains information about the pattern to be used. The GC contains a “fill style” attribute that can be set to indicate whether the area is to be filled with a solid color or a pattern. The pattern used is created by the application program, and the default pattern is a solid color. In GKS0b, an attribute similar to X11’s “fill style” attribute is set to either solid or pattern. If the “fill style” attribute is set to pattern, GKS0b uses the pattern indicated from a pre-defined pattern table. Some levels of GKS higher than 0b allow the application program to create patterns for the pattern table, but level 0b does not.

4.2.4.5 Magnification

In GKS0b, enlarging or shrinking the size of a picture generated on the screen is done by changing the window boundaries, while in X11 the size of the picture, or the objects in the picture, must be recalculated by the client and the screen redrawn. Another factor impacting the magnification capability is the text capabilities of X11, as mentioned in Section 4.2.4.3.

The different magnification capabilities of the two packages have an impact on the zoom capabilities of the two versions of Taskmaster, as discussed in Section 4.3.1.

As an example of magnification in GKS0b, if the original window boundaries are (0,0), (1,1), and a 4 times magnification of the picture is desired, the boundaries must be changed to (0,0), (.5,.5). The graphics calculations remain the same and only the portion of the generated graphics within the new boundaries is drawn. No such capability exists in X11, requiring the application program to enlarge or shrink the picture itself.

4.3 Changes due to Graphics Package

The difference between the X Window System and GKS are many, and some of the differences cause Taskmaster.2 to be different from Taskmaster.1 in ways that are visible to the user. These differences can be seen when the user chooses either of the zoom operations or any operations that use rubberbanding in Taskmaster.1. Other differences between the two graphics packages cause differences between Taskmaster.2 and Taskmaster.1 which are transparent to the user. Both the visible and transparent modifications are discussed further in the following sections.

4.3.1 Zoom Operations

As mentioned earlier, no capability exists in X11 for magnification of the picture appearing on the screen. This is due in part to the fact that text is drawn using fonts in the X Window System, requiring a particular font size to be chosen. Another contributing factor to the lack of magnification capability is the line width attribute. Either this attribute must be reset in the GC used for network drawing, or a separate GC must be created for each level of magnification. When the view of the network topology is magnified, the default font to be used during normal

network drawing is set to a font one size larger than the current default size and the line width is reset in the GC used for network drawing. Both versions of Taskmaster of the zoom capability, although this capability is implemented differently in the two versions. Taskmaster.2 has an incremental zoom , while Taskmaster.1 has a proportional zoom; Taskmaster.2 also has a limit as to how far in and out the network can be zoomed, while Taskmaster.1 has no such limit.

4.3.2 Rubberbanding

In Taskmaster.1, on selecting the **move node** operation the user is requested to select a node to move. Once the node is selected, a line connects the node to the cursor, wherever the cursor is, and this line follows the cursor. The line following the cursor is known as a rubberband. The X Window System does not have this capability, due to the window and event management being controlled by the client. A rubberbanding procedure must be written by the programmer to simulate the rubberbanding capabilities of GKS0b. Such a procedure is very slow to follow the cursor; a possible explanation for this lies in the implementation of X11. When complying with a graphics request, X11 calls the underlying graphics package of A/UX, QuickDraw. A rubberbanding procedure using QuickDraw may increase the speed of such a procedure, but decreases the portability of Taskmaster. Since this conflicts with one of the goals of this thesis, an alternate method is used. Using the **move node** operation as an example, once a node is selected, that node is highlighted and the cursor changes from the default arrow to a circle. Effectively, the system is now in "rubberbanding" mode. This method is used in Taskmaster.2 to replace any rubberbanding used in Taskmaster.1.

4.3.3 Menus and Other Input

In GKS0b, a menu can be used via a call to a GKS0b menu function. The application program

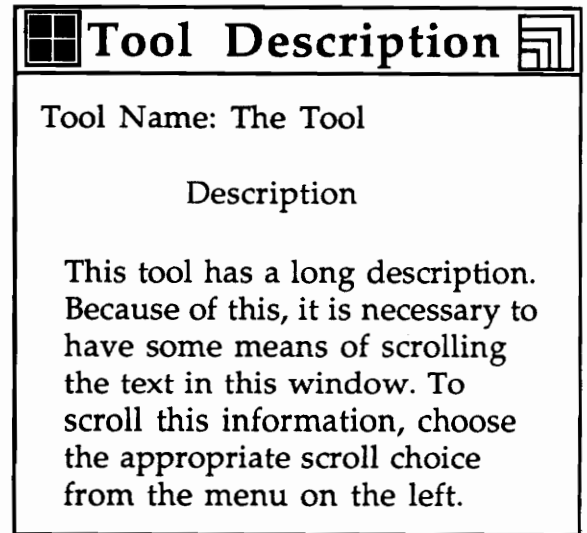
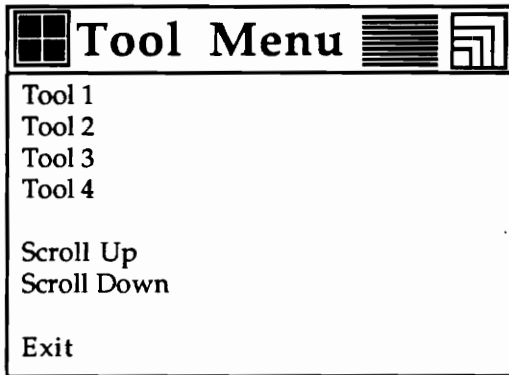
specifies the choices in the menu by creating an array with all of the menu options in it. This array is passed to a GKS0b routine, which displays a menu with the options in it and returns to the programmer a value indicating which of the options was selected by the user. A menu in X11 is simply a window with many subwindows. It must be created and handled by the programmer. An event loop is used with menus to

- map the subwindows,
- display the text for the options,
- monitor mouse movement in the menu, and
- check for menu option selection.

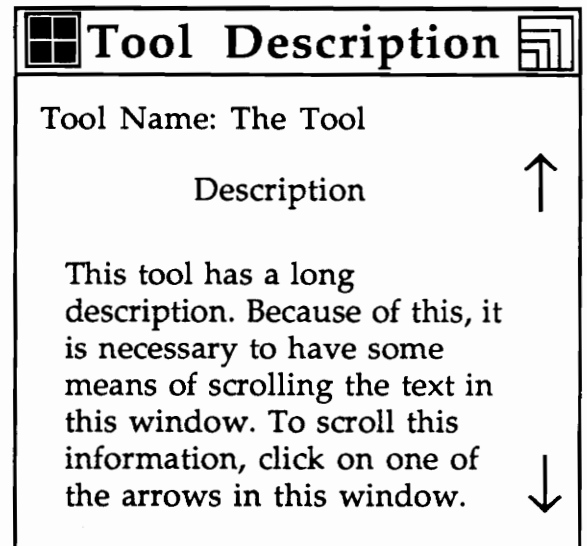
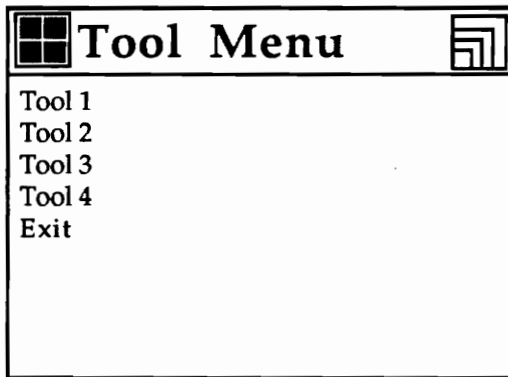
During the node specification process, a menu is presented to the user for tool selection (a tool menu). Also presented to the user is a window displaying any information known about the tool, i.e. a level 3 view. There may be more information about the tool than can fit into the window, and therefore the user must be allowed to scroll the information. The menu system of GKS0b requires that a menu selection be made before any other events are processed. Because of this, the tool menu of Taskmaster.1 contains the choices "Scroll Up" and "Scroll-Down" (Figure 4.1a). These choices refer to the information in the window with the level 3 view, not the menu itself. It is more natural and consistent to have scroll arrows in the tool information window, and allow the user to use those arrows, even when the menu is displayed (Figure 4.1b). X11 allows the client to enforce the consistent use of scrolling by handing the control of the windows and events to the programmer.

4.3.4 The Display

As mentioned in Section 4.2.2, refreshing the display in X11 is handled by the client. This means that the application program must redraw the contents of windows that become visible



(a)



(b)

Figure 4.1
Tool Menu Examples

after other windows are moved, resized or unmapped. A given window can have a "backing store" attribute set such that the contents of the window are saved, and whenever the window becomes visible, the contents are refreshed without the intervention of the application program. This ability is very useful, but must be used sparingly, due to the amount of memory required. Because of this, only the topology window has the "backing store" attribute set to save the contents of the window. All other windows must be refreshed by the client.

4.4 System

The network editor, the execution monitor and the tools database of Taskmaster.2 all reside on UNIX machines. This simplifies many of the operations of the network editor, and allows the communication between the network editor and the execution monitor to be achieved with UNIX sockets. Copying the master copy of the tools database from the host machine is done through the UNIX "rcp" [AUX88] command; invoking the execution monitor is achieved through the UNIX "remsh" [AUX88] command. Using "remsh" allows the execution monitor to simply be a compiled object, instead of the special DECnet object required in Taskmaster.1.

4.5 Hardware

The screen of a Macintosh II is smaller than the screen of a VAXstation I. This necessitates the re-design of Taskmaster's windows. The size of the windows in Taskmaster.2 are proportionally smaller than the windows in Taskmaster.1, and their location is different to allow for the easy viewing of all important information simultaneously. In the network topology window of Taskmaster.1, the "Exit" button is visible in the upper right hand corner of the window, the current operation mode is shown in the upper left hand corner, and the legend

explaining the different marking for execution time nodes is displayed in the lower left hand corner. Taskmaster.2's topology window is slightly different due to the window size differences. In particular, the "Exit" button is in the upper left hand corner, and the current operational mode is directly to the right of the "Exit" button. The legend is not shown unless a network is being executed.

Chapter 5

Conclusions

This thesis presents Taskmaster.2, an interactive graphical environment for high-level task specification, execution and monitoring. The primary goals of this research are to increase the portability of Taskmaster and to provide consistency in the user interface of Taskmaster. The development of Taskmaster.2 is the culmination of the research towards achieving these goals. The remainder of this chapter discusses the contributions of this research, its limitations, and possible future research work.

5.1 Contributions of this Research

5.1.1 Portability

By porting the network editor of Taskmaster from a machine using the VMS operating system to a machine using the UNIX operating system, the portability of Taskmaster.2 is much higher than that of Taskmaster.1. This is true not only because of the wide acceptance of the UNIX operating system, but because of the implementation of the underlying communication mechanism used between the network editor and the execution monitor. With both components

residing on UNIX machines, the communication between them can be achieved with the use of sockets. This approach allows both components to reside on the same machine, should that be desired, or allowing them to reside on different machines. Another factor contributing to the portability of Taskmaster.2 is the use of the X Windowing System as the underlying graphics package. The X Windowing System is itself portable, and thereby contributes significantly to the portability of Taskmaster.2.

As mentioned in Section 2.2, to achieve portability of the Taskmaster interface

- the environment must be supported by standard operating systems, between which communication is possible, and
- a standard graphics package must be used.

As is explained in the previous paragraph the environment is supported by standard operating systems. Actually, it is supported by one operating system, namely UNIX. The graphics are supplied by a standard graphics package, the X Windowing System. The goal of portability has been achieved.

5.1.2 Consistency in the User Interface

To achieve consistency in the Taskmaster environment, the following guidelines must be enforced:

- using the same terminology throughout the interface,
- placing error messages and instructions in the same location as they last appeared, and
- requiring the use of the same command for the same results in similar situations.

By changing the names of some of the commands in the operations menu, the first guideline has been followed. The second guideline has also been adhered to, with the assistance of the X

Windowing System.

The Taskmaster.1 interface exhibits many inconsistencies with respect to the third guideline. This research not only identifies the areas that do not adhere to this guideline, but determines whether or not reconciliation is needed in that area. If reconciliation is needed, the methods for achieving consistency are outlined, and the appropriate option is determined. Significant results in presenting a consistent user interface are outlined in the next three paragraphs.

The tools of Taskmaster.1, atomic and network, are treated very differently. By treating them the same in Taskmaster.2, the user can apply what is known about atomic tools to network tools and vice-versa. This allows the user to use the same operations in similar situations for similar results.

The atomic and super nodes of Taskmaster.1 are treated differently with respect to the expand and delete operations. By treating them the same in Taskmaster.2, both the top-down and bottom-up approaches to user task specification are succinctly supported. Again, this permits the use of the same operation in similar situations.

Arcs in Taskmaster.1 cannot be specified or created with super nodes. By allowing this capability in Taskmaster.2, the user can indicate data flow between any two nodes, whether or not the nodes are part of a super node's subnetwork.

All of the guidelines for maintaining consistency are followed in the Taskmaster.2 interface, and the goal of consistency has been achieved.

5.2 Limitations

Although Taskmaster.2 does achieve the goals stated in this thesis, it is still not without

limitations. Of the limitations discussed below, many have a variety of solutions, some of which are outlined.

1. Users cannot delete network tools from the database. This can be remedied by supplying an additional delete network tool operation to the user. The network tools in the database can be displayed to the user for selection.
2. Network tools cannot have optional or solicited arguments. When the user is specifying a node by attaching an atomic tool, values for the arguments of the tool are supplied by the user at the time of node specification. Recall that when a network tool is created, all of the nodes in its subnetwork must be specified. This means that all of the tools attached to these nodes have values for all of their arguments. Therefore, there is no need for the network tool to have arguments.
3. Taskmaster.2 does not perform any data path type checking until an arc is specified. A limited form of type checking can be performed at the time of arc creation. Assume an arc is created starting at node A and ending at node B. If none of A's output ports correspond to any of B's input ports, an error message can be issued.
4. During execution, the nodes in the network topology change their graphical representation based on their execution status. The arcs, on the other hand, do not change at all, whether or not there is data flowing along the arcs between two tools. This can be modified by changing the graphical representation of the arc when data is flowing along the arc, e.g. thickening the arc. Another possible modification is by having a shape, such as a small circle, travel the arc when data is flowing.
5. Taskmaster.2 allows the user to have only one network opened at a time. Allowing

multiple networks to be open provides the user with the flexibility to create and modify task networks in a natural manner. For example, during creation of a task network for a one task specification, the user may realize an intermediate network that solves a different task. Allowing the user to duplicate and maintain this new network without saving and closing the original network can be supported by allowing concurrent development of the two networks.

Other limitations that are present in Taskmaster.1 as well as Taskmaster.2 are discussed in [RAGH88].

5.3 Future Research Possibilities

The previous section outlines the limitations of Taskmaster.2 as well as their possible solutions. This sections addresses a broader scope of research possibilities. Following is a discussion of these possibilities.

1. One area needing extensive research is the creation of the tools database. Currently, creating a tools database requires the creation of a file that follows a certain form. A friendly user interface needs to be developed for not only creating a tools database, but adding to one.
2. Research needs to be pursued addressing the use of Taskmaster as a learning tool. For example, with the use of a UNIX tools database a novice can use Taskmaster to learn the UNIX commands without searching through vast amounts of documentation.
3. It is possible that Taskmaster can be used as a debugging tool for programs written using tools for which a Taskmaster tools database exists. The programs, written in a

conventional programming style, can be interpreted by Taskmaster, and a network topology which represents the program can be created by the network editor. Such an approach allows the programmer to view a graphical representation of the "program" which, in turn, can assist in spotting errors.

4. Another area of new research concerns the graphical representation of the tools. In Taskmaster.2, all of the tools have the same graphical representation: a node. Allowing different graphical representations for different types of tools provides a more powerful graphical interface. This area has many complications, not the least of which involves the decision concerning the appropriate graphical representations.

Several other research possibilities are also suggested in [RAGH88].

In summary, this research has provided portability and consistency for the Taskmaster environment. Taskmaster can be easily ported to any machines running the UNIX operating system, either with the network editor and the execution monitor residing on the same machine, or different ones. In the opinion of the author, the network editor of Taskmaster.2 now presents arcs, tools and nodes to the user in a uniform manner.

References

- [ANDE88] Ray Anderson "The X Window - A standard has arrived", *.EXE Magazine*, Vol. 2 Issue 9, March 1988, pp. 52-55.
- [ARTH83] James D. Arthur, "An Interactive Environment for Tool Selection, Specification and Composition", Ph.D. Dissertation, Purdue University, Indiana, 1983.
- [ARTH87] James D. Arthur, and Douglas E. Comer, "An interactive environment for tool selection, specification and composition", *International Journal of Man-Machine Studies*, Vol. 26 No. 5, May 1987, pp. 581-595.
- [ARTH88] James D. Arthur and K.S. Raghu, "Abstraction Mechanisms in Support of Top-Down and Bottom-Up Task Specification", *TR88-15*, Virginia Polytechnic Institute and State University, Virginia, 1988.
- [AUX88] A/UX Programmers Reference - Section 2, System Calls.
- [BERR88] R. E. Berry, "Common User Access - A Consistent and Usable Human-Computer Interface for the SAA Environment", *IBM Systems Journal*, Vol. 27, No. 3, 1988, pp. 284-293.
- [BOTT87] T. M. Bottegai and D. J. Quammen, "Graphical Development of Software", *IEEE Proceedings of the 1987 International Conference on Systems, Man & Cybernetics*, October 1987, pp. 334-338.
- [BROW85] Gretchen P. Brown, Richard T. Carling, Christopher F. Herot, David A. Kramlich, and Paul Souza, "Program Visualization: Graphical Support for Software Development", *IEEE Computer*, August 1985, pp. 27-35.
- [CHAN87] Shi-Kuo Chang, "Visual Languages: A Tutorial and Survey", *IEEE Software*, January 1987, pp. 29-39.

- [DIAZ80] J. L. Diaz-Herrera and R. C. Flude, "Pascal/HSD: A Graphical Programming System", *IEEE Proceedings of the Computer Software and Applications Conference*, October 1980, pp. 723-728.
- [EDEL88] Mark Edel, "The Tinkertoy 'Graphical Programming Environment'", *IEEE Transactions on Software Engineering*, Vol. 14 No 8, August 1988, pp. 1110-1115.
- [ENDE84] G. Enderle, K. Kansy, and G. Pfaff, *Computer Graphics Programming: GKS - The Graphics Standard*, Springer-Verlag, 1984.
- [KING80] Stephen King, *The Stand*, New American Library, New York New York, 1980.
- [FAN88] Wen Li Fan, "Partitioned Frame Networks: A Study of Structured Program Specification", M.S. Thesis, Virginia Polytechnic Institute and State University, Virginia, 1988.
- [GLIN84] Ephraim P. Glinert and Steven L. Tanimoto, "Pict: An Interactive Graphical Programming Environment", *IEEE Computer*, November 1984, pp. 7-25.
- [GLIN86] Ephraim P. Glinert, "Towards 'Second Generation' Interactive, Graphical Programming Environments", *IEEE Workshop on Visual Languages*, June 1986, pp. 61-70.
- [GRAF85] Robert B. Grafton and Tadao Ichikawa, "Visual Programming", *IEEE Computer*, August 1985, pp. 6-9.
- [GROT88] Michael G. Grottola, "UNIX and OS/2 not UNIX vs. OS/2", *Mini-Micro Systems*, Vol.-21 No.-9, September 1988, pp. 92-100.
- [GRUD89] Jonathan Grudin, "The Case Against User Interface Consistency", *CACM*, Vol. 32 No. 10, October 1989, pp. 1164-1173.
- [JACO85] Robert J. K. Jacob, "A State Transition Diagram Language for Visual Programming", *IEEE Computer*, August 1985, pp. 51-59.
- [JERN87] Mikael Jern, "Unravelling Graphics Standards", *Datamation*, Vol. 33 No. 23, December 1987, pp. 56/13, 18, 19.
- [KEEF88] Sarah Keefe, "PC's To Acquire UNIX Windowing Capability", *.EXE Magazine*, Vol. 3 Issue 1 suppl., June 1988, pp. 4-5.
- [LAKI80] Fred H. Lakin, "A Structure from Manipulation for Text-Graphic Objects", *ACM SIGGRAPH Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques* 1980, pp. 100-107.
- [LEWI89] Clayton Lewis, D. Charles Hair, and Victor Schoenberg, "Generalization, Consistency and Control", *SICGHI Conference Proceedings '89*, May 1989, pp. 1-5.

- [MA89] Pai-Chun Ma, Frederic H. Murphy, and Edward A. Stohr, "A Graphics Interface for Linear Programming", *CACM*, Vol.-32 No.-8, August 1989, pp. 996-1012.
- [MATW85] S. Matwin and T Pietrzykowski, "PROGRAPH: A Preliminary Report", *Computer Languages*, Vol. 10 No. 2, pp. 91-126.
- [MOND84] N. Monden, Y Yoshino, M Hirakawa, M Tanaka, and T. Ichikawa, "HI-VISUAL: A Language Supporting Visual Interaction in Programming", *IEEE Workshop on Visual Languages*, 1984, pp. 199-205.
- [MORI85a] Mark Moriconi and Dwight F. Hare, "Pegasys: A System for Graphical Explanation of Program Designs", *Proceedings of the ACM SIGPLAN 85 Symposium on Language Issues in Programming Environments - SIGPLAN Notices*, Vol. 20 No. 7 July 1985, pp. 148-160.
- [MORI85b] Mark Moriconi and Dwight F. Hare, "Visualizing Program Designs Through Pegasys", *IEEE Computer*, August 1985, pp. 72-85.
- [MULH86] Kelly. C. Mulheren, "An Interactive, Visual-based Environment for Task Specification", M.S. Project Report, Virginia Polytechnic Institute and State University, Virginia, 1986.
- [MULH87] Kelly C. Mulheren, James D. Arthur, and Roger W. Ehrich, "A Network Specification and Execution Environment", *International Journal of Mini and Microcomputers*, Vol. 9 No. 3, 1987, pp. 57-62.
- [MYER88] Brad A. Myers, "Window Interfaces: A Taxonomy of Window Manager User Interfaces", *IEEE Computer Graphics and Applications*, No. 9, September 1988, pp. 65-84.
- [NYE88a] Adrian Nye, *The Definitive Guides to the X Window System, Volume 1:Xlib Programming Manual for Version 11*, O'Reilly & Associates, Inc., 1988.
- [NYE88b] Adrian Nye, ed., *The Definitive Guides to the X Window System, Volume 2:Xlib Reference Manual for Version 11*, O'Reilly & Associates, Inc., 1988.
- [PATE88] F. Paterno, "The Integration Between Functions of Graphics Systems and Windows Management", *Pixel, Computer Graphics CAD/CAM Image Process*, Vol. 8 No. 10, 1988, pp. 49-54.
- [PAYN86] Stephen J. Payne and T. R. G. Green, "Task-Action Grammars: A Model of the Mental Representation of Task Languages", *Human-Computer Interaction*, Lawrence Erlbaum Associates, Inc., Vol. 2, 1986, pp. 93-133.
- [PERL86a] Gary Perlman, "Coding Quality and Tools in Programming Methods", *SIGSOFT Engineering Notes*, Vol. 11 No. 3, July 1986, pp. 44-50.

- [PERL86b] Gary Perlman and Frederick L. Horan, "Report on UNIX|STAT release 5.1: Data analysis programs for UNIX and MSDOS", *Behavior Research Methods, Instruments, & Computers*, Vol.-18 No.-2, 1986, pp. 168-176.
- [POLS88] P. Polson, "The Consequences of Consistent and Inconsistent User Interfaces", *Cognitive Science and Its Applications for Human-Computer Interaction*, ed. Raymonde Guindon, Laurence Erlbaum Assoc., 1988.
- [POUN89] Dick Pountain, "The X Window System", *BYTE*, January 1989, pp. 353-360.
- [POWE83] Michael L. Powell and Mark A. Linton, "Visual Abstraction in an Interactive Programming Environment", *SIGPLAN Notices*, Vol. 18 No. 6, June 1983, pp. 14-21.
- [RAED85] Georg Raeder, "A Survey of Current Graphical Programming Techniques", *IEEE Computer*, August 1985, pp. 11-25.
- [RAGH88] K. S. Raghu, "Taskmaster: An Interactive, Graphical Environment for Task Specification, Execution and Monitoring", M.S. Thesis, Virginia Polytechnic Institute and State University, Virginia, 1988.
- [RAUC88] Wendy Rauch-Hinden, "No rupture in the UNIX community", *Mini-Micro Systems*, Vol.-21 No.-9, September 1988, pp. 57-58.
- [REIS86] Steven P. Reiss, "GARDEN Tools: Support for Graphical Programming", *Advanced Programming Environments: Proceedings of an International Workshop*, June 1986, pp. 59-72.
- [SCAN88] Tim Scannell, "What's all the fuss about UNIX?", *Mini-Micro Systems*, Vol.-21 No.-8, August 1988, p. 56.
- [SCHN87] Ben Schneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley Publishing Co., Inc., 1987.
- [STAS87] Susan Stash, "A Multiple Standards Approach", *Computers and Graphics*, Vol. 11 No. 4, pp. 479-481.
- [UHLI88] S. Uhler, "Enabling the user interface", *IBM Systems Journal*, Vol. 27 No. 3, 1988, pp. 306-313.
- [VAXG84] VAX GKS0b, Software Reference Guide, Digital Equipment Corporation, November 1984.

Vita

Brenda Jean Jackels was born in Minneapolis, Minnesota January 5, 1966. She attended many grammar schools (her family moved often), junior high, and two high schools (her first one was closed and she graduated from her rival high school). She then spent four years in Urbana-Champaign at the University of Illinois. After getting her bachelor's degree in Computer Science (and getting the nickname Annie), she decided that the real world wasn't ready for her. In order to give the real world time to prepare for her arrival, she spent another two years in school. After getting her Master's in Computer Science and Applications from the Virginia Polytechnic Institute and State University (known commonly as Tech, Virginia Tech and VPI), she accepted a job offer from IBM in Manassas Virginia. It is still unknown if she found the real world ready for her entrance. The combination of Illini and Hokie can be a formidable one.

To be continued ...