(2)

RL-TR-93-203
In-House Report
December 1993

GUIDE TO THE ENVIRONMENTAL-LANGUAGE

Jason Hamshar

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

RL-TR-93-203 has been reviewed and is approved for publication.

APPROVED: [illegible]

SAMUEL A. DINITTO, JR., Chief
SW Technology Division

FOR THE COMMANDER: [illegible]

JOHN A. GRANIERO
Chief Scientist
Command, Control, & Communications Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE December 1993	3. REPORT TYPE AND DATES COVERED In-House Jun 93 - Aug 93

4. TITLE AND SUBTITLE GUIDE TO THE ENVIRONMENTAL-LANGUAGE	5. FUNDING NUMBERS PE - 62702F PR - 5581 TA - 27 WU - 65
6. AUTHOR(S) Jason Hamshar	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Rome Laboratory (C3CA) 525 Brooks Rd Griffiss AFB NY 13441-4505	8. PERFORMING ORGANIZATION REPORT NUMBER RL-TR-93-203
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (C3CA) 525 Brooks Rd Griffiss AFB NY 13441-4505	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Joseph A. Carozzoni/C3CA (315) 330-3564

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Environmental-Language (E-L) was designed and developed by Dr. Mike Karr of Software Options Inc., and Harvard University. The software performs most of the functions of a Configuration Management system and an Activity Coordination system. It is constructed on top of the Emacs text editor (Emacs is part of GNU and copyrighted 1985, 1986 by Richard M. Stallman) using the ELSP programming language embedded in EMACS.

Configuration Management is the process which monitors multiple programmers working on a project and provides the capability to document each others work without over-writing others. All changes are tracked and recorded. In lieu of working with files, E-L operates on artifacts, which differ in that they are:

- In theory, never deleted. Alterations are handled by creating new artifacts. By saving previous versions, the user can rollback in the event of problems.
- Typed, such as Unix-programs, C-modules, C-source, or Lisp-modules, rather than by extension conventions.

Activity Coordination manages activations, which are formalized by transaction graphs. With E-L servers running on multiple hosts, an activity (known as a process) will execute in parallel across these member sites distributed according to each system's

(Continued)

14. SUBJECT TERMS Configuration Management, Activity Coordination	15. NUMBER OF PAGES 48
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT U/L

Block 13 (Continued)

utilization and uninhibited by the host's architecture and operating system. An Activity Description, also known as a transaction graph, is a graph of inter-related symbols representing a program. Each symbol or node has special functions, capabilities, and features inherent within it. Activity description programs are constructed by ordering these symbols and their links with each other in specific ways. Used in conjunction with the activity coordinator, a single activity may be executing over several networks.

Contents

1 Introduction

1.1 What is the Environment Language?

The **Environmental Language** [1] (also referred to as E-L) is an operating environment, constructed over Emacs [2], that integrates the power and functionality of a Configuration Manager, an Activity Coordinator, and an Activity Description language into one package. Built on this Emacs foundation, E-L comes equipped with a versatile editor and a powerful environment on which to base its operating structure. In fact, many of the operations are either prebuilt Emacs commands or developed using Emacs' formalisms.

1.2 Who is this manual meant for?

At the time of this document's release, E-L is suitable mainly for researcher. Lacking the robustness of commercial products, E-L should not be used by naive users! Intended as a supplemental guide to navigating through E-L, this manual will not provide an indepth theoretical analysis of all of E-L's features and capabilities, but rather it will try to clarify areas, which may have been confusing or inadequately documented. However, the average user, meeting the given criteria, should be able to gain insight, run demonstrations, and even write simple activity description programs after reading this manual. In addition, it's strongly recommended that the user be familiar with Emacs!

1.3 What is Configuration Management?

Configuration Management is a process, which monitors users working on a project and provides the capability for several people, such as programmers, to work on documents without inadvertently over writing each others work (although users can create multiple up-to-date versions) ; all changes are tracked and recorded. In lieu of operating on files, the user works on artifacts, which differ in that they are:

[1] Environment Language is copyrighted by Dr. Mike Karr of Software Options Inc. and Harvard University.

[2] Emacs is part of GNU and copyrighted 1985, 1986 by Richard M. Stallman.

- In theory, never deleted. Any alterations are handled by creating a new artifact. By saving previous versions, the user can now rollback in the event of problems.
- Typed, such as Unix-program, C-module, C-source, Lisp-module, rather than by extension conventions.

For instance, given a group of C programmers tackling a particular project, C-module artifacts would incorporate not only the changes made by each and every team member, but also the header file knowledge within the same artifact. Furthermore, having a type assigned to it, the E-L operating environment would also be able to provide more automated services, such as automated compiling and linking.

1.4 What is the Activity Coordinator?

The *Activity Coordinator* manages activations, which are formalized by transaction graphs. Eventually, having E-L servers running on a multitude of host sites, an activity (known as process) will execute in parallel across these member sites portioned according to each system's utilization and uninhibited by the host's architecture and operating system. For example, in an E-L environment, if a process or an activity requires a large amount of computing resources and is executing on a busy site, the E-L server will

1. Communicate with E-L servers on remote sites.
2. Find the server with the lowest system load.
3. Transmit the activity to this site, where it will continue its execution.
4. Repeat the steps above as necessary and upon program completion, it will transmit the results back to the site of origin

The intention behind this global unification at a more "friendly" and open level is to maximize performance by optimizing the overall resource utilization of machines on the network. Ideally, machines running E-L would have a consistent level of CPU utilization throughout the day, but with a generally much lower magnitude in utilization peaks.

For example, lets say a United States international company executes a significant number of jobs during standard business hours (8am - 5pm EST) each day. Member branches, located in Japan and Sweden, also execute execute a significant number of jobs during their standard business hours. Using traditional methodologies, each branch's computer would have heavy system utilizations during the day and almost none at night. In addition, while one branch computer is running at a high utilization level, the other branch computers, situated around the world, are running idle. Machines governed by the E-L paradigm, however, would search for idle machines over the network and use the combined resources to reduce the overall work of each computer. If nothing else, this is a significant shift in the computer paradigm.

1.5 What is an Activity Description?

An *Activity Description*, also known as a transaction graph, is a graph of inter-related symbols representing a program. Each symbol or node has special functions, capabilities, and features inherent within it. Activity description programs are constructed by ordering these symbols and their links with each other in specific ways. Used in conjunction with the activity coordinator, a single activity may be executing over several networks. Later, in this manual, an example of such a distribution of an executing program into smaller fragments will be shown.

2 Notion of Plexes

Depicting a viewport to the vast E-L knowledge base (all information available in the mini-world), a plex describes the related attributes for the given object. For instance, in the instantiation plex, each entry contains the name of an instantiation, the date and time it was created, and the identity of the user, who created it. On the other hand, for the artifact plex, each entry starts with a plus symbol, the date and time when the artifact was last updated, its type and name, and the identity of the user, who last modified it. Most plexes, regardless of the plex type, share the following characteristics:

- Each entry for a plex is on a separate line.
- Any changes to the E-L environment, either local or remote, that are related in some way to the plex being viewed, take *immediate* effect on

the display. For example, if a user modifies artifact A on computer system one and commits it, while viewing the artifacts plex on computer system two, then the plex on my screen will display the new features of A (new time and date, and user).

- Where attributes exist to be manipulated, the plex permits refiltering of the buffer. Refiltering is made available to create smaller views on a potentially large buffer.
- They allow certain prompts to be automatically answered by placing the cursor on an entry in a related plex. For example, if the user issues view-instantiation, while the cursor resides on an entry within the instantiation plex, then the entry will be the default provided when the command requests an instantiation name.

To call up a plex, the user places the cursor in an Emacs' buffer and types *meta-x view-plex.*

View plex:

Type in the name of the plex (or part of the name and press the space bar for the completion list), and then press return. The minibuffer will report done and the plex is shown in the buffer by the name of the plex type.

3 Operations on Artifacts/Drafts

In E-L, artifacts are the backbone behind configuration management operations. We are going to examine the tasks available to the user and programmer. However, when starting a new project or editing an artifact, the programmer is actually creating a draft, which resembles an artifact in appearance, except that it's temporary and can't be executed.

3.1 Creating

To begin a draft, type *meta-x create-draft* and in response to the prompt **Type:**, type in the full name (or a partial name and press the space bar to get the completion). Once the programmer has selected a type from the list, available by pressing the space bar, he/she must respond to the system prompt **Name:** (something that identifies the artifact to the user). After providing a name, a buffer labeled by this name will appear in one of the Emacs' windows. The programmer can now manipulate the draft.

3.2 Editing

To edit an artifact, type *meta-x view-plex* and respond

View plex:*artifacts*

Move the cursor to the artifact in the plex to edit and issue the *meta-x edit-artifact* command. This line should appear in blue highlight after the prompt **Edit artifact:**. Except for the situation described below, E-L will bring up the new draft containing the contents of the now outdated artifact in the Emacs' buffer labeled by this name. The programmer can now manipulate the draft.

The problematic situation arises, if the requested artifact or an artifact that references this artifact is currently being modified by another user, E-L will display the warning

This artifact is almost-out-of-date. Edit anyway? (yes or no).

The user **should** always respond with *no*, as a yes reply will result in two artifacts with the same name being created. This scenario cascades into a multitude of problems. Since the artifact is "almost-out-of-date" and entails the existence of a draft, E-L will prompt

This artifact is almost-out-of-date. Take over draft? (yes or no).

Indicating *yes* will enable draft editing, unless the draft is currently being edited by another user in which case E-L will apprise the user of the situation (file locks).

When editing an artifact, which is now a new draft, any artifacts, which referenced this artifact, have now also become drafts, provided conversion hasn't already taken place.

3.3 Manipulating

Once either of the two operations above have been done, the Emacs' buffer may be modified using any of the standard Emacs' editing commands with the exception of the highlighted buttons. The actual buttons shown depend on the artifact type chosen and have special meaning to the artifact. They can not be removed and may require certain information prior to being saved.

To reference another artifact from the current artifact, move the cursor to the position in the buffer where the reference is to be made, and type *meta-x insert-artifact-into-draft.* Supply the name of the artifact to call in response to the prompt **Artifact to insert:**. When reading the Language Users Manual and looking at the examples, anything within [] means to call upon another artifact. For example, the C compiler is the artifact *cc*!

When working with the C artifacts, it's recommended that the function headers be highlighted. In addition to making these headers stand out from the code, the highlight also contains special information that E-L can use during processing. The function header is marked using standard Emacs. Highlighting this marked block is accomplished by issuing the *highlight-region* command. Depending on the exact nature of the artifact, additional information may be added to the highlighted header. Highlighting is disengaged by placing the cursor anywhere over the highlighted region and typing *highlight-region.*

3.4 Saving

To update the E-L knowledge base and associated plexes, the programmer should place the cursor on the draft to be saved and type *commit-one-draft* or *commit-draft-and-references-and-referencers.* While the former commits only the currently edited draft, the latter will commit all drafts, which reference this draft. Prior to committing the draft(s), the computer will request verification; **Commit one draft (default {buffer name} C-module):**. The user can either proceed by pressing return or cancel via *control-g.*

Except for the situation where the draft to be saved references a draft being concurrently edited by a user running another E-L process, the draft will be committed. If this situation occurs, E-L will report the name and type of both the draft being committed and the draft that it references. Essentially, the user must wait until the base artifact has been committed.

Success or failure to commit will be reported in the minibuffer and in the event of success, the buffer will change to read only as indicated by the two %% signs, which now appear in the bottom left hand corner of the buffer. Depending on the artifact type, the programmer may be asked if E-L should send a notice after making some derivation.

3.5 Deleting

When removing information from the knowledge base, former versions move from either out-of-date (if deleting an artifact) or almost-out-of-date (if deleting a draft) to up-to-date (ROLLBACK). In addition, those files which referenced only former versions are also reinstated. However, deleting an artifact is permissible, when references to this artifact do not exist (maintenance, consistency). Several ways to eliminate information in E-L exist.

delete-artifact-and-references Removes the artifact specified in the artifact plex.

delete-artifact-in-region Deletes a range of artifacts from the artifacts plex.

delete-draft Deletes a draft from the draft plex.

contro\ı-x k Emacs command for kill buffer. If this is an active draft (currently assigned to an editable buffer), E-L will ask the programmer to verify whether to abandon or delete the draft.

3.6 Locating Problems

If the programmer received errors in the E-L Notices window when committing a programming language artifact, the problems may be located and an error message reported by applying the *examine-problems* command to the artifact in question. A fairly descriptive message will be given inside of the commit-error buffer.

3.7 Viewing

To view an artifact without changing it into a draft, move the cursor to its location in the artifact plex, and employ the *examine-artifact* command. The contents will be shown in a read only buffer.

3.8 Executing

Only the operating system artifact available, such as the Unix-program artifact, may be executed. In order to run a C program (or any high level language program) on a UNIX machine, the programmer has to also create an Unix-program artifact, which indicates what to compile and how to compile it. Assuming that no errors exist and a derivation was possible, the executable is located in the *.../e-l/as/repository/artifacts/derivatives /Executable%sun4-dbg/* and given the name of the next integer available.

Running a program is accomplished by moving the cursor to the appropriate Unix-program artifact, and typing *execute.* A shell is started, an environment variable linking the variable name $ executable and the actual filename is created, and this label resides after the prompt. Type in any command line parameters desired, press return, and the executable begins processing.

4 Running your own Artifact

Having provided a background into artifacts and the instructions available to both the programmer and the user, we will now illustrate these tools through an example. For this example, the programmer is going to write a C program, which will:

1. Prompt for the user's name.

2. Print hello on the terminal.

3. Capitalize and print the user's name.

Throughout the remainder of this section, some box figures will be drawn. Anything appearing in **bold** facetype indicates the non-removable button (determined by the artifact type) placed into the buffer by E-L. All text shown as *slanted* represents calls to other artifacts. The remaining text shown in normal must be typed in by the user.

1. To start a new file, move the cursor to an emacs buffer, and type *meta-x create-draft.*

2. Answer the prompts requested in the mini-buffer in the following manner. Note, that the user may provide any legal name to the name prompt.

   Type: *C-source*
   Name: *Capital*

3. A buffer named Capital should appear with a button labeled Caption as shown below.

   Caption:

4. The programmer should move the cursor to the blank line immediately under the **Caption** and type *meta-x insert-artifact-into-draft.*

   Artifact to insert: *malloc*

5. A blue highlighted button appears containing the name of the artifact, its type, and the date and time it was last modified. After this button, type the phrase <*spec*>.

6. Move the cursor to the next line, and repeating the last two steps, insert an artifact called *ctype.*

7. Utilizing the Emacs text editor, finish manipulating the buffer by typing in all the text displayed in normal style below until they are identical.

Caption:
[malloc universal 1993/06/27 13:37:55] < spec >
[ctype universal 1993/06/27 13:37:55] < spec >

```
char *Capitalize(orig)
   char *orig;
{
   int i;
   char *dest;

   dest = (char *) malloc(sizeof(char)*(strlen(orig)+1));
   for (i = 0; i ¡ strlen(orig); i++)
      dest[i] = toupper(orig[i]);
   return(dest);
}
```

8. To commit this draft (save it as an artifact), move the cursor to this Emacs' window and type *meta-x commit-one-draft.* Press return at the first prompt, and type *no* for the second prompt;

   Send notice after deriving C-text from Capital? (yes or no)

   A notice is now displayed as to the whether the draft was committed. If unsuccessful, the user can try to discover the problem by typing *meta-x examine-problem* followed by the name of the artifact. For more information, please refer to saving artifacts in the previous section.

9. The programmer may want to dispose of this read-only buffer, which is accomplished via *control-x k.* If the buffer is still in edit mode (not committed) and the user types this command, E-L will prompt

   Killing the buffer Capital C-source. Abandon or delete the draft?.

   abandon The draft will remain in the draft plex, but be flagged as inactive. By inactive, this implies that another user may now take-over the draft.

delete The draft along with any changes made is deleted from the draft plex. An artifact by this name, if one exists, is now transformed from a status of almost-out-of-date to up-to-date.

10. Assuming that the programmer has successfully committed the previous draft, he/she will start another by typing *meta-x create-draft* and responding to the set of prompts as described below.

    Type: *C-module*
    Name: *hello*

11. A buffer named hello should appear with three buttons. The slanted items appearing in the buffer are artifacts the programmer will insert using the same procedure as shown earlier. The text appearing in normal style represents the information the programmer will type into the buffer.

```
Caption:
—— Targets ——————————————————————————————
—— Source ———————————————————————————————
[stdio universal 1993/06/27 13:37:55] < spec >
[Capital universal 1993/06/27 13:37:55]
void main()
{
   char name[30];

   puts("Please, Enter your name?");
   gets(name);
   puts(Capitalize(name));
}
```

12. Once this has been accomplished via Emacs editing, commit the artifact as demonstrated earlier.

13. Create an artifact with type Unix-program, name *Hello*, and a buffer that resembles that shown below.

```
Caption:
—— Targets ————————————————————————
sun4 [cc universal 1993/06/27 13:49:37] -g
 C [cc universal 1993/06/27 13:49:37]
—— Modules/Instructions ————————————
 [hello C-module 1993/08/10 10:23:32]
```

14. When committing the artifact, respond *yes* to the EL prompt

    Send notice after deriving Executable%cc from Hello? (yes or no)

    After a short period of time, the programmer should see a message in the E-L Notices buffer similar to

    kbsa-tmp 29 Job done. (derive-from 'Hello' at 1993/08/10 09:38:00)

15. To execute this program, view the artifact plex, place the cursor on the line shown below and type *meta-x execute.*

    + 1993/08/10 09:38:00 Unix-program Hello kbsa-tmp,

16. A shell is started and the command **$executable** appears after the prompt. After executing the program, the user may opt to kill the shell buffer.

5 Survey of Activity Description Nodes

Much the same way that the building blocks of a C program are its functions and commands, these nodes make up the syntax/grammar for the E-L activity description language. Each node aids by accomplishing some relatively simple task, such as read user's input or adding to numbers, and communicating this fact to other nodes, which perform their own specific actions.

Prior to examining/writing a program, a semi-understanding of these nodes and the particular features and capabilities is in order.

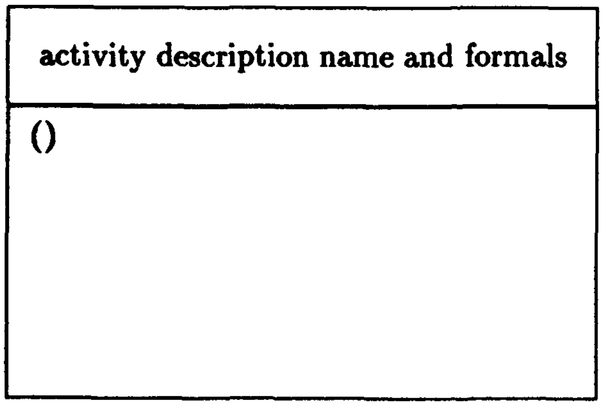
After each node name, I have listed its internal node window, where one exists, as it appears initially. I have also provided a list indicating the number of arcs both input and output, which may be connected, to a specific node.

5.1 Name and formals

Found in the top left corner marked by (), this node contains a one-line buffer, which will be supplied by the programmer with the name of the activity description file minus the extension *. ad* before the (). Inside the (), the programmer lists the parameter variables and their types necessary for this module's execution; each variable name is immediately followed by a colon and then a type. The types supported by the current version of E-L include

1. Boolean
2. Integer
3. String
4. None
5. Any

Values for these variables is requested upon execution, ***meta-x instantiate-ad.*** These variables are similar to command-line arguments in that is, they are both requested and initialized upon program initiation.



5.2 Exclusion node

In addition to having no contents and requiring all attached arcs to be exclusion arcs, an exclusion node will not initiate a grab, but rather waits until one arc is grabbed, before grabbing everything [3]. This node can be reasoned as being similar in functionality to that of a C function, which issues the wait system call in UNIX, and becomes a homicidal parent once any one of its children acknowledges (signals) it.

- Input Capabilities – *N* arcs connected to any part of the node.
- Output Capabilities – None Available.

5.3 Sequencer node

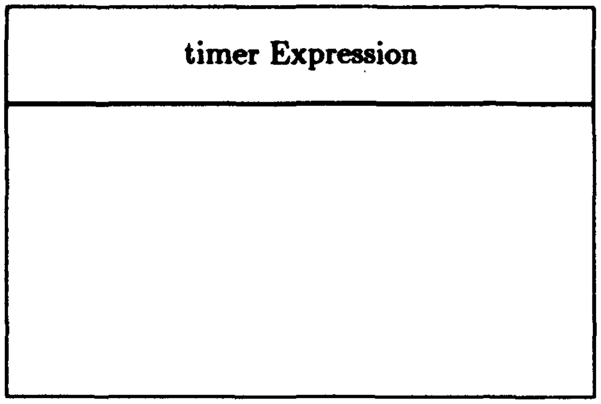
Similar to a tee joint in plumbing, this node takes *all* its input data flow line(s), and sends each value received down *all* its output line(s). This is useful, when two different graph routes become similar or when a need arises to share the same data line among different parts of the graph.

- Input Capabilities *N* arcs connected to the top half of the node.
- Output Capabilities *N* arcs connected to the bottom half of the node.

5.4 Timer node

Used to interface an activity description program with the system clock, the timer node permits a programmer to set up time requirements that allow actions to be carried out after a certain amount of time. Normally, the time value to track is stored inside the buffer, however, in the absence of this inherent information, the time value is arrived at via the input arc connected to the top center of the node. The top left and right regions on the node supply supplemental information, that when connected to arcs, *must* be provided prior to the time value expiration. At this time, current implementation requires a timer node to have a least one input arc.

[3]This description is paraphrased from Mike Karr's GED on-line documentation



5.5 User-interaction node

The User-interaction node enables the activity description program to acquire input from the user during the actual execution of one instantiation. When the instantiated program is executing and this node is lit, the program is waiting for input from the user. To respond, the user places the cursor over the node and clicks the right mouse button. A pop-up request response window will appear and the mini-buffer will prompt the user for information. Place the cursor inside the mini-buffer window, and respond appropriately for the type of data requested. The physical node itself can be broken into three parts

top third Enables connections via arcs for input purposes.

Inside the framed box is the user or a string variable defined as some user. *N* arcs may be attached.

Only the operator whose user id matches this value may provide input; it's secured from unauthorized access. However, an unauthorized user can see what responses are being made by either or both parties.

middle third Permits communication between two or more User-interaction nodes as to who gets a response. This segment permits *N* arcs to be connected.

Once data is obtained from a node, it shuts down all nodes connected to this middle third. Transitivity does *not* hold when connecting arcs among a group of nodes; Every node, that wants to be disabled once this node is activated or vice-verse, must have a direct arc with every other node. Essentially, when a value is provided to one of these nodes, all User-interaction nodes attached to it are disabled. On the screen, this section of the node is represented by a string revealing some relevant message as determined by the programmer.

bottom third Port where output arc(s) are linked to the node. On screen, displays the type expected on input and returned down the N output arc(s).

User-interface node contents
User: **Display:** **ResponseType: any**

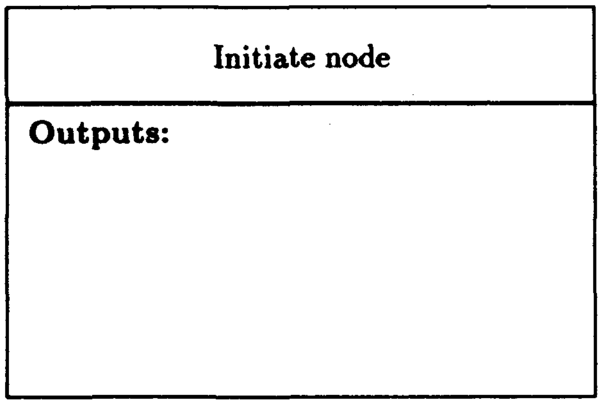
5.6 Terminate node

The Terminate node signifies the end of the program. In the absence of inputs, only one arc may be attached, while n arcs may be linked to n input variables.

terminate node contents
Inputs:

5.7 Initiate node

The initiate node is the beginning of the activity description program. As the beginning of the activity description, only one should exist. *N* arcs may sprout from this node.



5.8 Synchronizer node

Having no contents, it ignores all input arcs and yields none on all output arcs. Its purpose is to wait for every input, before continuing onward, thus the name synchronizer. Think of it like a sleep call that waits until being signaled by all of its cooperating processes.

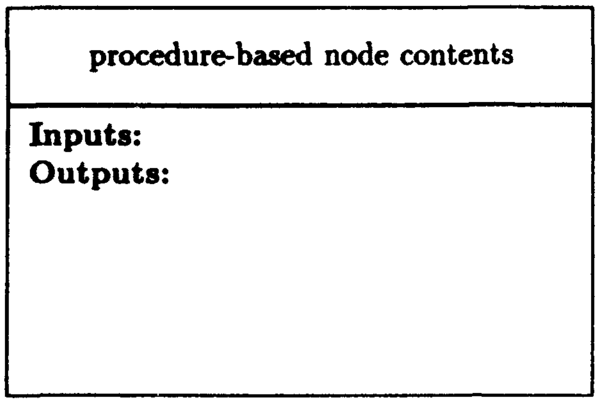
5.9 Procedure-based node

Theoretically, the procedure-based node incorporates the ability to take a set of inputs, perform some operation(s), and then send out some outputs. The idea is to expand the capabilities of the activity description, by allowing tasks not supported directly to be done by calling a program written in a more productive language like C. Currently, the only tasks carried out by this node are simple assignment statements; a $\leftarrow +(b, c)$. This physical node can be broken into three parts.

Input Area Located in the top half of the physical node, this segment, in the absence of a parameter list, permits only one connection. With *N* parameters comes the possibility for *N* arcs.

Data Area Current implementation allows only expressions like that listed above to be written. This section does not accept input or output arcs.

Output Area Taking up the bottom half of the physical node, this section permits *N* arcs to be attached.



5.10 Condensation node

A condensation node allows module activity descriptions to be called and defines the linkage between the dangling arcs found inside the module and its references to the outside environment. This is like the familiar "box it" idea in that all the details are encapsulated inside the ellipse, and thus, a complex set of operations can be thought of as a single call. Well designed condensation nodes facilitate code reusability.

All dangling arc references are placed around the ellipse according to the number of degrees specified in the node window. For the activity description name, the programmer *must* supply the name of this file as referenced on the disk minus the extension *.ad*. All variables to be passed must be provided after the actual parameters (do not include the variable type as it's already known).

condensation node contents
Activity description name: **Actual parameters:**

5.11 Dangling arc node

Dangling arcs enable the execution flow to progress outside of the current activity description. For instance, they are most oftenly used when flow of execution is transmitted from one Condensation node (module) to another. The name of the dangling arc is given by the value on the first line of the node window. A dangling arc may have only one linked arc, regardless of whether it's input or output.

dangling arc label

5.12 Site Region

Allows the region it bounds on the activity description to be run on a site other than the one in which it was started.

6 Commands of the Activity Coordinator

To use the activity coordination system described by E-L, the following commands are available. As these instructions are being described alphabetically, the systems output will be displayed in **bold** print, and the user will be told the type of information, that should be provided.

Note, that unless stated otherwise, when E-L asks for the name of an instantiation, the user must type in the full name of the instantiation, that is its name, date, and creator's user name. Pressing the space bar, provides the list of possible completions for the data pattern entered by the user (the entire list will be shown if no pattern is provided). In addition, prior to completion of the last prompt, all commands may be aborted via *control-g*.

6.1 delete-instantiation

Removes an instantiation from the plex.

Delete instantiation at site: rome-lab
Delete instantiation at rome-lab with name:

After providing the name, a couple of messages will flash inside the minibuffer and the instantiation will be gone.

6.2 instantiate-ad

Makes an executable instance of an activity description. It also gathers information as to the values for its command line parameters.

Activity description: /home/KBSA2/kbsa/e-l/bin/

The user must provide the pathname as well as the name of the activity description for the response. For convenience, Emacs pathname/filename completion is available.

Name for instantiation:

Unlike the general case in which an existing instantiation is being referenced, E-L is requesting that the user provide a name for a new instantiation.

Instantiate at site: rome-lab

The system will now inquire as to the values for its parameter variables. Once this is done, **Garbage collecting...done** will be shown in the mini-buffer, and the instantiation will be created.

6.3 view-instantiation

Brings the executing instantiation into view. The authorized user will be able to respond to prompts and make decisions.

View instantiation at site: rome-lab
View instantiation at rome-lab with name:

Having satisfied the last prompt, some messages will be displayed. A window entitled PGE will pop up and the mini-buffer will display the message **Wrote /tmp/gsa***DDDD* (*D* represents a digit). Finally after some more processing, the system will plot the transaction graph. Information provided during the instantiate-ad phase will be disclosed in blue letters at the top of this window.

7 Running an Example Activity Description

Distributed with the E-L system is an example activity description with the name *editor-reviewer.ad* found in the directory *../ad* from inside the *bin* directory. The editor-reviewer demonstrates the ease with which a single instantiation without real networking programming effort can be made to run off two machines, acquire data from two isolated users simultaneously, and notify the other user of any responses made. Essentially, the scenario simulates a correspondence between two people, an editor and a reviewer. The reviewer will tell the editor, whether or not he/she will review the document. The editor in turn acknowledging the reviewer, will perform some action, such as cancel the job, accept the cancellation by the reviewer, or wait until after the reviewer is finished reviewing. Any sudden decision by either party becomes immediately known to both.

For the remainder of this section, **Bold** faced items indicate what is shown in the mini-buffer, while *emphasis* will be placed on items the user types. In addition, after every sub-section number, or command entry, don't forget to press return.

1. To start E-L on a UNIX system, the user would move to the directory where E-L was stored and execute the program *e-l*. On our machine KBSA1, the user would enter the following

   (a) *cd /home/KBSA2/kbsa/e-l/bin/.*

   (b) *e-l &*

2. To run the demo or any other activity description for that matter, type *meta-x instantiate-ad.*

3. **Activity description: /home/KBSA2/kbsa/e-l/bin/**

   Press the delete (or backspace on some systems) key four times to remove the *bin/* and type in the changes described below and press return.

   Activity description: /home/KBSA2/kbsa/e-l/ *ad/editor-reviewer.ad*

4. **Name for instantiation:** *el-intro*

   A new buffer called *scratch has been created and the user can type in any name within reason. For this demonstration, I'll assume that the user will respond to this prompt with *el-intro*. At this point, EL wants to know the site at which to run instantiation.

5. **Instantiate at site: rome-lab**

6. After displaying a couple of message via the mini-buffer in rapid succession, EL makes inquiries as to the values for its list of global variables. For this example, EL asks a series of questions as to which users will be performing what roles. Although any legitimate user will suffice, I

will assume the user is ***kbsa-tmp***. If the user fails to put in his/her user name, he/she will be unable to answer the prompts later.

   (a) **editor:*kbsa-tmp***

   (b) **reviewer:*kbsa-tmp***

   (c) **reviewer-site:**

   (d) **due-date:**

   The date format must be entered in the form of *year/month/day time* corresponding to the syntax pattern *DDDD/DD/DD DD:DD:DD*, where **D** is defined as a digit. For example, the string *1993/08/26 14:00:00* translates into 1pm August 26, 1993. If the user selects a time that has already expired, then the execution flow will pass through the timer node, otherwise, execution flow will stop in the timer node until either a situation arises which disarms the node, or the specified time setting is reached by the system clock.

7. After providing this last piece of information, the system should begin processing and eventually create an instantiation. During this phase a multitude of messages will quickly flash inside the Emacs minibuffer. *Sometimes*, messages indicating success will appear in the E-L Notices window.

8. In any event, success or failure can be seen by typing *view-instantiation*, pressing the space bar to see the list of all existing instantiations, and seeing if the instantiation name the user provided above, *el-intro*, is listed. In addition, providing the instantiation was created, one will also see following the instantiation name in the list, a date format string followed by the user name. The date format shown is NOT the same as the one stipulated by the user, but rather indicates when a particular instantiation was created. The user name distinguishes ownership as to the creator of a particular instantiation.

9. **View instantiation at site: rome-lab.**

10. **View instantiation at rome-lab with name:**
    el-intro DDDD/DD/DD DD:DD:DD kbsa-tmp

    Note that the entire instantiation name, date format, and user name are included. Remember, the D's shown here will appear as actual digits on the terminal.

11. If already viewing this instantiation, EL will retort **Already viewing Rome lab/DDD** and stop. When viewing an instantiation, do not attempt to close the window by a) destroying the window at the X environment level or b) using ged-quit as both leave E-L thinking the window is still there. If something goes wrong and the user wishes to see this instantiation again, then either

    (a) Exit of E-L via *control-x control-c* and then re-enter, or

    (b) View a different instantiation and then view the desired instantiation once again.

12. Assuming that the above is not the case, a window labeled PGE will be displayed along with two ellipses and a bunch of arcs connecting them. If a different instantiation was currently being viewed E-L will automatically cancel it prior to starting a new view. To expand a module:

    (a) Move the mouse cursor to the center of the ellipse to expand.

    (b) Press and hold down the right mouse button. An L shape should appear pointing north-west.

    (c) Move the L shape to the top left corner of the region where the user wishes to have the box expanded, and release the button.

13. The user may opt (and it's strongly suggested) to have this instantiation viewed on two machines. Of course, the user can expand the other ellipse on the same workspace, although this is often very unacceptable as the screen quickly becomes cluttered. To view this instantiation on two machines,

(a) Start E-L on another machine, under the same user name provided in the previous step. Remember, the prompts must be answered by the user matching the user name provided during instantiation.

(b) *meta-x view-instantiation*

(c) Follow the same steps as stated above after view-instantiation, except expand the other ellipse.

14. For this step, it's assumed that two machines are running E-L, have typed view-instantiation, provided the same instantiation name, and each machine has expanded a unique ellipse. At this point, the user with the expanded reviewer ellipse should notice that the two User-interaction nodes right below the Initiate node are green (Green signifies node is currently active).

15. To select one of the nodes, move the cursor over it and press the right button. A query window will appear and the user can either select it by providing the proper input or *control-]* to abort. Once input has been provided, the user(s) should witness the execution of the activity description. **Note: If the node expects an integer, the user must supply it in the mini-buffer or risk errors!**

16. Having provided input, the user should notice how the other User-node, linked to this node, was deactivated and the icon(s) connected to the output are now activated.

8 Working with the GED

GED, or Graphic Editor, is used to create new and modify existing E-L activity descriptions. Much the same way a programmer uses a text editor to write C source code, an E-L programmer uses the GED to write activity descriptions. Although it appears to be nothing more than a flow chart of symbols, closer inspection at the E-L buffer shows that in reality an activity description is essentially a document of graphics commands. We shall now embark upon examing basic GED operations.

8.1 Creating/Modifying an Activity Description

An activity description is read into the editor by the following:

- Issue the *meta-x ged-file* command and provide the the path name and filename of the activity description, when E-L requests it. Depending on whether or not the activity description the user supplied exists, E-L will either edit or create a new one.
- An activity description already occupying an Emacs' buffer may be called by *meta-x ged-buffer*.

8.2 Saving an Activity Description

Saving the activity description is accomplished by moving the mouse cursor to the emacs buffer with the same name as your activity description and typing *control-x-s*.

8.3 Operating on Nodes

Located in the upper right hand corner, eleven icons are available for the programmer to handle a number of tasks. Having already dissected the particular features and capabilities of these icons in depth, we are now going to look at how these icons can be manipulated by the programmer.

8.3.1 Placing

Before placing an icon on the workspace, the activity description programmer must select an icon by placing the mouse cursor over one of the eleven icons located in the upper right hand corner and clicking the left mouse button. To actually drop the now highlighted icon on the workspace, move the cursor to the desired location on the workspace and again click the left mouse button. If the active icon has a node window, then it will now be displayed and the programmer can modify its contents. This window can be saved via *meta-control-c* or aborted *control-]*. Except for the situation arising where an abort was done, the location referenced on the workspace should now display the active icon along with any modifications made to its internal window, where appropriate. However, not all internal window contents will be shown on their physical icons; the condensation node for example stores within its window the actual degrees where dangling arcs are to be located, while on screen, only the names themselves are displayed.

8.3.2 Editing

After a node has been placed, its internal node window may be modified, provided of course that it exists, by moving the mouse cursor over the node that needs to be modified, and clicking the right button. After making the desired changes to the internal node, commit them by issuing a *meta-control-c.* Changes may be aborted via the standard Emacs command *control-].*

8.3.3 Moving

Moving a node to another physical location on screen is accomplished in the following manner:

1. Position the mouse cursor over the node to be relocated.
2. Hold down the middle mouse button.
3. Move the mouse cursor to the destination and release the button.

Note the similarities between this and moving windows in X.

8.3.4 Resizing

The procedure described below enables the programmer to resize a node.

1. Move the mouse cursor to a location very close to the nodes border.
2. Hold down the right mouse button, and the cursor should appear as an inverted L shape with a cursor pointing to its vertex. If this is not the case, repeat this procedure, until the L shape is displayed.
3. Move this L shape into the node itself and then move it to its destination. Like X, the resizer must enter the window itself, prior to actual resizing!
4. After moving the cursor to its target position, release the button. The node should now be resized.

8.3.5 Deleting

Deleting a node and all its connecting arcs is achieved by entering delete mode and then clicking left over any and all nodes to be destroyed. To enter delete node, move the mouse cursor to the bottom right hand corner icon. Clicking on these cross-hairs, the cursor will transform into a skull and bones, signifying a dangerous operation. To return to edit mode, click on this icon again, and the cursor will resume its normal shape.

8.3.6 Printing

Printing an activity description chart is accomplished by typing *meta-x ged-lpr-file*. At the E-L prompt, provide the pathname and the name of the file (include the ".ad" extension).

8.3.7 Getting Help

Help is available on all the eleven nodes. To get help on a node, move the cursor to its symbol in the upper right hand corner and click the right button. A help window will appear.

8.4 Drawing Arcs

Careful observation of the GED screen would reveal, that throughout the entire time we've been editing our activity description, one of the two nodes in the bottom left hand corner has been highlighted. In addition, during this entire edit session, the programmer could have drawn arcs at any time. To draw a straight line arc, highlight the proper icon in the bottom left hand corner, and move the mouse cursor to the source node. Here, the programmer will click the left button and if drawing an arc is presently legal [4] from this node, the cursor will change into a dark circle, indicating that the source node has been set. Move the circle mouse cursor to the destination node and click the left mouse button. If legal, a line will be drawn between these two nodes. The curved line icon in the bottom left hand corner is for drawing bezier splines, which function exactly the same as the straight line, but allow

[4]For more information on the legality of drawing arcs from certain nodes, see the section entitled **Survey of Activity Description Nodes**.

visually appealing lines to be drawn. Drawing a bezier spline differs from drawing a straight line in that the designer must also select "control points", indicated by the cursor shaped as a finger pointing.

9 Running your Own Activity Description

By now, the reader should be comfortable with the editor-reviewer, and in particular what exactly occurs between the nodes. Before writing an activity description program, the programmer should model the situation being investigated. Some of the key questions a designer might ask include:

- How are things done normally?
- What are the alternatives a user may wish to select?
- How should I respond to these alternatives?
- Where might tasks, beyond the capabilities of this language, be required?

For this simple example, we are going to create an extremely simple financial program (equivalent of 5-10 lines of C code). The specifications are as follows

1. Program will request three integers, Principle, Rate, and Time, upon *execution.*
2. Once these numbers have been given, Interest and Yield will be calculated.
3. The results will be displayed and no further operations will be done.

Once we have laid out the specifications as such, it's fairly easy to construct an activity description matching this criteria.

1. Start E-L (see previous section) and issue *ged-file.*
2. Ged file: /home/KBSA2/kbsa/e-l/bin/*../ad/finance.ad.* Don't forget the . ad extension or the editor will not operate properly .

3. Right click the node labeled () in the upper left hand corner, and edit the field to look like:

activity description name and formals
finance(Banker:string)

   Finance, as we stated above. is the name of the activity description program, while banker is a string representing the user, who will be permitted to interact with the instantiation, supplied during *instantiation-ad.*

4. Since this is a simple program, we won't have to worry about condensation nodes or dangling arcs. However, since every program has to begin somewhere, select the **initiate node** and place it somewhere at the top of the screen. Leave it's node window "as is", that is save edits via *control-meta-c.*

5. Select **User-interaction node** and place three of them below the Initiate node. Fill the nodes in according to the following table, remembering to save edits before going on to the next one.

user-interface node contents
User: *Banker*
Display: *"Principle"*
Response type: *integer*

user-interface node contents
User: *Banker*
Display: *"Rate"*
Response type: *integer*

user-interface node contents
User: *Banker*
Display: *"Time"*
Response type: *integer*

6. Now that we've constructed the mechanism for obtaining the three integers, we need to perform some arithmetic operations. This is accomplished by using a procedure-based node. Place this node below the three defined above and update its internal window node to look like:

procedure-based node contents
Inputs: *Principle, Rate, Time*
Interest ← **(Rate, Time)*
Interest ← **(Principle, Interest)*
Interest ← */(Interest, 100)*
Yield ← *+(Principle, Interest)*
Outputs: *Principle, Interest, Yield*

7. Having done the calculations, all that is required is to print the results and exit. This task is accomplished by the terminate-node. Position this below the procedure node and edit its internal window to look like:

terminate node contents
Inputs:*Principle, Interest, Yield*

8. At this point, all the critical foundation work is complete, but the designer must still indicate the pathway for the execution flow. Make sure the straight line indicator on the bottom left has been selected prior to continuing. We are now ready to indicate the sequence on how these nodes are to be executed.

9. Move the mouse cursor to the Initiate-node and click left. The cursor should now look like a filled-in circle. Move the cursor to the top of one of the user interaction nodes and click the left button. An arc should have been drawn with an arrow pointing at the User-node.

10. Repeat this general procedure to link the other user-nodes to the Initiate-node. When drawing the arcs, it doesn't matter which end is the initial point, and which is the destination point. The points of connection on each node determine the actual direction the execution flow will take.

11. For each bottom of the user-interaction node, draw a line to the parameter name in the Procedure-node, which matches the name inside the user-interaction string.

12. Draw lines matching the names from the bottom of the Procedure-node to the terminate node. Once this has been accomplished, you are finished writing the program.

13. To run the program, use *meta-x instantiate-ad.*
    Banker:*kbsa-tmp*

14. To interact with this executing program, issue the *meta-x view-instantiation* command. Note that execution of the procedure node does not begin until all its input parameters have been supplied.

15. To clean up after running this instantiation, type *meta-x delete-instantiation* and answer the prompts as previously stated.

Hints Recovery File

This appendix compliments of Mike Karr of Software Options Inc. It details the step-by-step procedure for recovering E-L, should the activity coordinator and/or activity descriptions develop problems.

1 The system hangs with "Locking abc:x ..." in the minibuffer. This means that the lock server is not running. From a shell, type "e-l -do e-l-start-lock-server".

2 You try to instantiate something and nothing appears in the instantiations plex.

A View the activations plex. The first line should be of the form "host:port", e.g., KBSA2:5001. If it is, go to B. If not, you will want to start an activation server, but before doing so, you should check whether there is one still existing. (When an activation server is shut down, it immediately removes itself from the activations plex, but does not exit until all its children have exited. This may take up to a minute when everything is working, but it _may_ be the case that children get stuck, in which case, they, and the activation server too, never exit.) The most reliable way to tell if an activation server is running anywhere on your network is to go to a shell and type:

```
e-l -do e-l-show-locks -s
```

The "-s" means status; this lists all the clients of the lock server, and any activation server will be among these. There is one line per client, with a host:port the first item on the line, the user who started the process, and, if the process is alive, the name of the process, its pid in parentheses, and ending with "alive". If e-l-activation-server is the name of a process, don't try to start an activation server right away, because your new one will be stillborn. Wait 60 seconds and repeat the above command. If the process is still around, then on the indicated host, issue "kill pid" (with no signal). The process should go away. If it doesn't, get really brutal and issue "kill -9 pid". If the process still doesn't go away, consult your local operating system expert.

Once you get rid of all activation server processes, you can start a new activation server. In Emacs, issue the command:

```
  M-x start-activation-server
```

You will be prompted for a host, the default being the one you are running on. Supply a host and follow it with RET. You should see a hostport become the first line in a view on the activations plex. Go to step C.

B This step assumes the first line in the activations plex is of the form "host:port". To determine if there really is a running activation server, put the Emacs cursor at the beginning of the buffer, and issue:

```
  M-x stop-activation-server
```

If the activation server is alive, you will be prompted with its host:port; since you don't really want to stop it, type C-g rather than RET. If the activation server is dead, it will say so, and ask if you want it removed from the plex. Indicate the affirmative, after which it should disappear. Then go to step A.

C This step assumes a running activation server. The next step is to make sure that there is a "supervisor activation". There should be some line in the view on the activations plex that looks like:

```
  17 kbsa-tmp supervisor ...
```

If so, go to the next step. Otherwise, issue the command:

```
M-x start-supervisor
```

You should see a line for the supervisor appear at the bottom of the view.

D If the supervisor is asleep, its line in the plex will look like:

```
17 nancyr supervisor e-l-activation 17
```

If this is the case, things really should work. If they don't, call me [Dr. Karr]. Otherwise, continue to the next step.

E If the supervisor is awake, its line in the plex will look like:

```
17 nancyr supervisor KBSA2:5001     541    e-l-activation  17
```

Sometimes, a supervisor will die with its eyes open, i.e., it will look awake when it is really dead. To test if this has happened, on KBSA2 (i.e., the characters preceding ":5001"), do a ps 541. If there is no such process, the supervisor somehow died. (This shouldn't happen, but it seems to.) Put your cursor on the line in the view on the activations plex, and issue the command:

```
M-x orphan-activation
```

You will sometimes see the old pid (here, 541) replaced by a new pid, in which case the instantiation will sometimes proceed without further ado. In other cases, the KBSA2:5001 and the 541 will both be become blank, which means it has gone to sleep. (The term "orphan" is inconsistent with other terminology; it should be "somnabulize" or something like that.)

F Sometimes a supervisor will go into a terminal coma, in which case its line in the plex will look like:

```
17 nancyr supervisor      error-dummy     error-state
```

If so, put the cursor on this line and issue the command:

```
M-x kill-activation
```

The line should go away. You can then start a new supervisor as in step C.

G If the supervisor is in some other state, call me.

H If the supervisor seems ok and things still don't work, try restarting the activation server. Issue the command in step B, but respond with a RET instead of a C-g to proceed with the command. Use the techniques in step A to determine that the activation server actually exits. If it doesn't exit, use the techniques in step A to kill it from a shell and restart it.

Appendix B

This was the list of reference papers available at the time this document was being produced. Information contained in this document was gathered from my own experiences with E-L and these papers, which I strongly recommend are closely examined.

References

[1] *Users Guide to the Activity Description Editor (ade).* Software Options Inc., May 27, 1993.

[2] Townley, J., *E-L User's Manual*, Technical Report, Software Options, Inc., September 29, 1992.

[3] Karr, Michael *Beyond the Read-Eval Loop: Architecture of the Artifacts System*, Software Options, Inc., February 28, 1991.

[4] Morris, Robert *Implementation of an Activity Coordination System*, 1991.

[5] Karr, Michael & Cheatham, Thomas E. *An Activity Coordination System*, May 21, 1993.

[6] Townley J. *Language Users' Manual*, April 5, 1993.

[7] Kellner, Marc I., Feiler, Peter H., Finkelstein, Anthony, Katayama, Takuya, Osterweil, Leon J., Penedo, Maria H., & Rombach, H. Dieter *Software Process Modeling Example Problem.*

[8] Karr, Michael and Cheatham, Thomas *A Solution to the ISPW-6 Software Process Modeling Example*, Proceedings ISPW-6, September 22, 1990.

[9] Karr, Mike *Software Design Documentation for the KBSA CM*, September 17, 1992.

[10] Karr, Mike *Permissions*, September 17, 1992.

[11] Karr, Mike *Attribute Domains*, September 17, 1992.

[12] Karr, Mike *Boolean Algebras and Regular Expressions*, September 17, 1992.