The Trillium User Interface Design Environment

D. Austin Henderson, Jr.

Intelligent Systems Laboratory
Xerox Palo Alto Research Center
Palo Alto, California 94304

Abstract

Trillium is a computer-based environment for simulating and experimenting with interfaces for simple machines. For the past four years it has been use by Xerox designers for fast prototyping and testing of interfaces for copiers and printers. This paper defines the class of "functioning frame" interfaces which Trillium is used to design, discusses the major concerns that have driven the design of Trillium, and describes the Trillium mechanisms chosen to satisfy them.

Introduction

As machines become more complex, the design of their user[1] interfaces becomes more difficult. A good methodology for improving these machine interfaces is "cut and try": build the interface, try it out on all interested parties (most particularly the end users), discover what its difficulties are, modify the design, and repeat. This empirical approach works best if the total time around this implementation cycle is short — the shorter the better. The goal is so-called "fast prototyping." At Xerox, we wanted to explore speeding up the design cycle by using a computer-based design environment for prototyping and simulating interfaces. The approach to be taken was to build a computer-based "construction set" populated with pieces appropriate for rapidly assembling our class of "functioning frame" interfaces.

 ACM 0-89791-180-6/86/0400 - 0221 $00.75

Historical Background

Five years ago at Xerox, the cycle time for interface design and experiment was, for the most part, so long that one or two cycles was the norm for our products. These products were "simple" machines, copiers and printers, with "simple" interfaces: mechanically-based control panels with lights, buttons and mimic presentations of the machine and its parts; typewriter-based recursive-decent menu systems; and display-based graphic interfaces. However, the functionality of the products was growing, and the interfaces (including display-based graphic interfaces) were becoming more complex. Given this increasing complexity, the production of good interfaces was becoming more and more difficult to guarantee.

As a result, a few design sites had included programmers in their design teams, and testing was done on simulations of the interfaces, rather than on final product itself. This reduced the design cycle time from months to weeks and was clearly improving the quality of product interfaces. However, it was observed that, although formal passes around the cycle would involve operability testing, often it was the case that the designers themselves, on first seeing and using their interface would immediately know how to improve it. The resulting desire to "fix" it before spending the resources to test it would shorten the cycle still more. Most of the effort in this shortened cycle was in the programming necessary to create the interface simulations.

To tighten the design loop any further, we clearly needed to remove the programming from the cycle. Fortunately, the availability of personal computers capable of running modern symbolic programming environments provided the opportunity to experiment

1. In this paper, the word "user" refers to the person who interacts with the interface designed using Trillium. The word "designer" refers to the person who interacts with Trillium itself. The word "operator" is avoided because it carries the connotation that the role of the person interacting with the machine is the "machine's operator," when interfaces for a broader class of users (repair persons, installers, sales persons) may all be appropriate for design using Trillium.

with doing just that. In response to this opportunity, a user interface design environment called Trillium was built [Henderson, 1983]. The rest of this paper discusses Trillium, examining a few of the key issues which influenced its design[2].

"Functioning Frame" Interfaces.

It will be helpful to ground the discussion in an example of the class of interfaces that Trillium is designed to design. Figure 1 shows a single frame from an interface for a simple copier. This frame is one of many making up the interface. It is composed of a decorative border, a collection of controls, and an area summarizing the description of the job. The number pad controls the count of the number of copies that should be made when the Start button is pressed. The buttons in each of the columns control other features of the job. Buttons are "pushed" by using the mouse. The buttons are back-lit (simulated by inverting the screen), with the lights "coming on" to indicate which particular choice has been made. There are also some restrictions on the settings of the features. For example, no more than 99 copies can be made in any given job and the copier does not make one-sided copies from two-sided originals. When these restrictions are violated, the interface takes some appropriate action to notify the user (in one case by refusing to take the expected action).

The frame in Figure 1 is one of many in this interface, through which the operator must move while using the machine. It shares a number of common features with the other frames, such as the artwork defining the border, the graphics defining the controls, and the displays comprising the summary area.

The term "Functioning Frame" will be used to describe interfaces like the one just described — the interfaces that Trillium can simulate. The key features of this class are:

- frame based control panels
- active controls and displays presented in two-dimensional
- concurrently active controls
- controlled movement among multiple frames

Other interfaces of this class include keyboard-driven menu systems and simple window-based editors.

2. There are many of these design issues, arising out of the task which it addresses, the technology on which it is built, and the skills and interests of users who employ it. This short paper can focus on only a few of the most important. Also, there are other design environments which address different sets of these issues. A longer paper (in preparation) will explore these in much greater detail and will discuss the relationship of Trillium of other design environment.

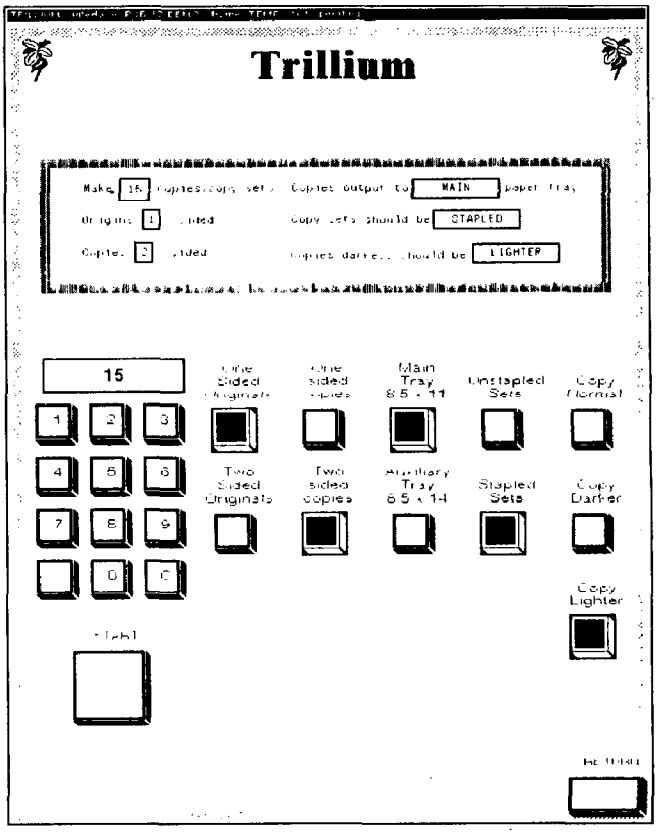

Figure 1: A single frame from an interface for a multi-functional office machine.

Design environments for Functioning Frame interfaces.

A fast prototyping environment for functioning frame interfaces should have the follow characteristics:

From the context of Trillium's conception (see Historical Background above):

1. Design should not require programming: Programming admits specifying much more than interfaces, and is therefore much more fine-grained than is needed for specifying just interfaces. Some formalism for specifying the design which is more specialized to design and reflects design abstractions is needed.

2. Design should be fast: The time required to go around the design/try cycle should be kept as short as possible. Design changes should be immediately affectable, and the ramifications of those changes immediately experienced. A short cycle maintains the designer's focus, and permits more exploration of the design space.

3. The environment should support the design of the interface's behavior as well as its presentation: Interfaces are active entities. Fast prototyping must enable experimenting with the behavior of the interface as well as with its visible presentation (looks).

From the nature of the interfaces to be designed[3]:

4. The environment should support two-dimensional controls and displays: The interfaces being designed are based on both physical and electronic displays and controls. These controls must be presented simultaneously and be concurrently active. This will require good graphic support.

5. The environment should support multiple views (presentation and behavior) of the machine: Interfaces break down the presentation of the functionality of the machine into parts, not all of which may be visible at once. For example, on a display-based multi-functional office machine, separate sections may be provided for copying and sending mail. Also, the interface may have different modes: similar presentations of the machine that behave in different ways. For example, when the user is entering an identification code, the number pad buttons control change significance and the other buttons cease to function.

6. The environment should support moving among the views of the machine: The interface may have to supply its user with some way to move around amongst these different views: going deeper into a more detailed description of the job, backing out to a higher level, moving over to another view.

7. Designs should make a clear distinction between machine state and presentation (controls and displays): That the machine behaves implies that it has some sort of internal state. Presentating these states to the user is one of the important functions of the interface. An interface may choose to present this state in many different ways. Also, many different controls may affect the same part of the state. It is therefore important to make a distinction between the state of the machine and its presentation.

8. Designs should support restrictions on, and interactions between, parts of the state of the machine: Certain states may be illegal. Certain changes may occasion other changes. Means for expressing these, and the actions that should be taken when these are detected must be provided. Certain changes may occasion other changes. Means for expressing these dependencies must also be provided.

From the design process:

9. The environment should recognize similarity among the pieces of the interfaces: An interface is made up of many individual pieces for describing controls, presentation and the interactions between parts of the state. Segments are similar to one another, but are different along distinguishable dimensions. (In Figure 1, the buttons are all similar, but each has its own label and effect). The environment should support and make full use of this concept of "similarity through controllable diversity."

10. The environment should support the construction and use of new design abstractions, particularly composition and specialization: The designers at a site develop their own local ways for describing these segments of interfaces. In addition, these design abstractions evolve. A design environment with a fixed set of design abstractions is a conservative force, restricting the designer to describe interfaces with unchangeable terminology. In contrast, an evolving set of abstractions supports a design community in extending its own language of, and thinking about, designs. Two of the simplest mechanisms for creating new abstractions are composition and specialization. Particular configurations when used repeatedly take on a life of their own, with the whole having its own characteristics that determine the characteristics of its parts. Specialization comes from the recognition of a particular configuration of a more general abstraction as having a separate importance of its own.

11. The environment should support incomplete specifications: Given a changing set of design abstractions, it is important for a designer to be able to create an instance of a design abstraction without knowing all the details of its definition. Then those details should be available from that instance for modification as the design progresses.

12. The environment should support sharing of common parts of the interface: There is much that is shared between different parts of any interface of any size. (In the example, the border and the Return button are the same in all frames. Also, the graphic used in all the buttons is the same.) While copying parts provides easy construction, it does not support easy modification of these shared parts.

3. Not all interfaces need all of these capabilities. But the environment must enable designers to create interfaces with these alternative characteristics, if for no other reason than to establish that the chosen interface is preferable.

Describing an interface in Trillium

A Trillium interface is composed of a collection of frames. Figure 1 portrays one frame of many in the interface of a multi-functional office machine, one of which is presented at any given time while the interface is operating. Each frame is composed of a collection of items, each of which is of some itemtype. In Figure 1, the collection of controls and displays which supports entering the number of copies is a single item -- a NumberPad. Each itemtype has a set of characteristics. The number pad has a *Placement*, a *Cell*, an *InitialValue*, and so on - 13 in all. Each characteristic has a value type from which Trillium determines how to manipulate values of that characteristic. The value type of *PrintBackground* is "grayshade" which indicates that the designer will want a shade editor to manipulate values of this characterisitc. Specific items are defined by supplying values for some of these characteristics; values not supplied by the designer are filled in with the characteristic's default value. The graphic which is the value of the *Picture* characteristic of the NumberPad is defaulted to *SimpleButtonBitmap*.

The style is that of a child's construction set; the set has pieces (items) of the same kind (itemtype). However, unlike the pieces in a construction set each of which it unchangeable, Trillium items are variable along certain dimensions (the characteristics). It is as though the rods in the set, for example, instead of coming in fixed lengths (and color, etc.), were adjustable in length (and color, etc.). Items are assembled within a frame to create both presentation and behavior. A designer uses the editing tools in Trillium to create, modify and experiment with operator interfaces. Thus the copier frame of Figure 1 is created by laying out a NumberPad, five SetOfVerticalButton's, and then adding some other pieces which constraint the values set by those items. This involves shifting rapidly between editing the interface (designing) and trying it out (operating) to evaluate the effect achieved. This process is supported by interactive window-based editors, as shown in Figure 2.

Itemtypes are either primitive or composite. An item of a composite itemtype defines a sub-assembly of other items; the composite item is expanded into these sub-items. A NumberPad has as subitems twelve buttons of various itemtypes, a PrintRegion, and a NumericInitializer; a button has as subitems a Picture depicting the button, a LineOfText for its label, and a sensor detecting when that area of the screen is "touched" with the mouse The substance of the

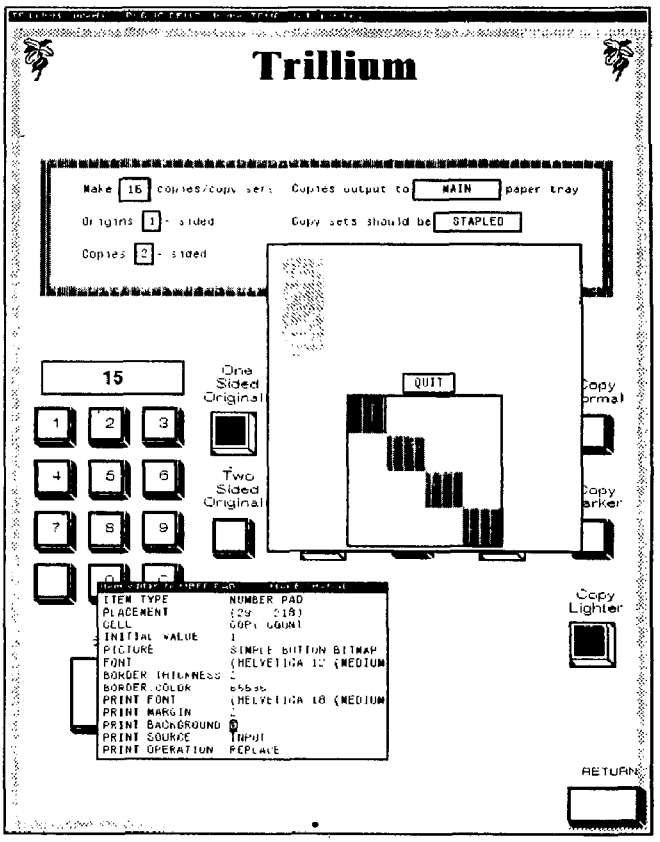

Figure 2: A portion of the Trillium screen containing an editor for a NumberPad showing its characteristics and their values, and a shade editor being used to modify one of those values.

composite itemtype, therefore, is an explicit description of how the characteristics of the composite item determine the number and individual characteristics of the sub-items of which it is composed. A (little) language of composition is used in making the descriptions. A NumberPad is described in this language as having a characteristic named LabelFont (among others) for controlling the font used in labelling the buttons, and as being composed of (among the other subitems) ten NumberButtons, each of which has, as the value of its characteristic for controlling the label, just exactly the value of the LabelFont characteristic of the NumberPad. Figure 3 shows part of the cascade of items resulting from the expansion of a NumberPad. Trillium also includes editors for manipulating the descriptions of itemtypes, as weel as some simple tools for infering the description of a composite itemtype from an example - a set of items which together would be the expansion of an item of the new composite itemtype.

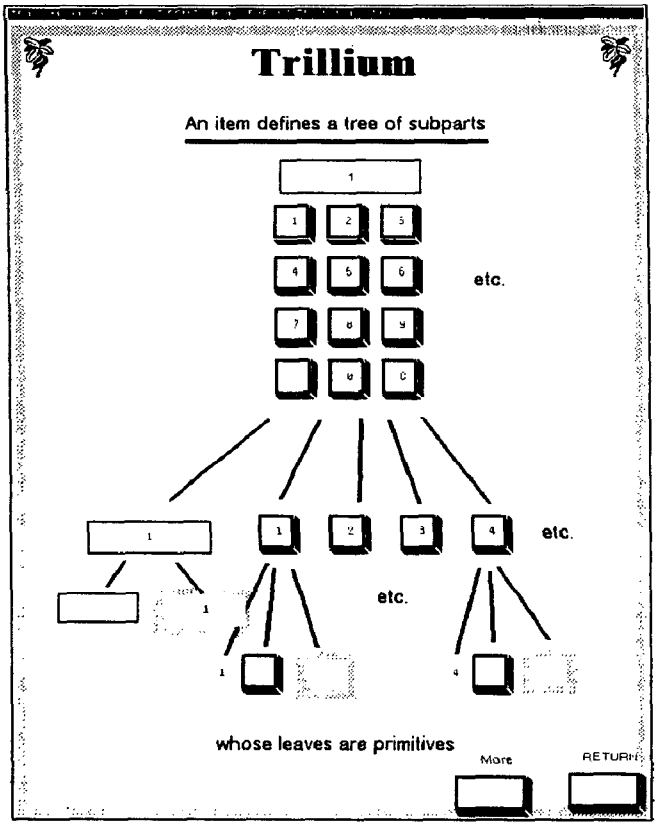

Figure 3: Part of the cascade of items resulting from the expansion of a NumberPad.

The Trillium Machine

The presentation and behavior of a Trillium interface is defined by the behavior of the items that make up its frames. A composite item derives all its presentation and behavior from its sub-items; primitive items have their presentation and behavior built in. The primitive itemtypes have built into them (as functions in the underlying programming language, Interlisp-D) the visible and behavioral description of the interface. This presentation and behavior is actually realized by an interpreter of the interface, the "Trillium machine." As it "runs" the interface, the Trillium machine need only look at the primitive sub-items resulting from the expansion of all the items in the frame currently being presented.

While running the interface, the Trillium machine creates and modifies a data structure representing the state of the machine being simulated. The state of the machine is given by two things: the frame stack - a push-down stack of frames which have been traversed to get to the current (top of stack) frame; and a collection of cells, each of which has a name and holds a value (a state variable).

The primitive items are divided into six kinds: each kind has a different protocol to which it must respond. The behavior of the interface is determined by the responses of the primitive items to the messages in these protocols.

The first three kinds of primitive item deal with the presentation of the machine:

artwork: the static graphics of the interface. Borders are artwork, as are the pictures which give the illusion of a button.

sensors: items which sense activity in the "world" around the interface, and take some action as a result; actions change the state of the machine, either by changing the frame stack (push - go deeper, pop - go back, or move - go sideways to another frame) or by attempting to change the value of cell. There are sensors which detect the mouse button being pushed within a certain rectangular region of the display; and others which detect the passage of time.

displayers: items which reflect the values of cells onto the display. The Lights "behind" buttons are displayers, as is the PrintRegion in the NumberPad and the PrintRegions making up the summary area.

The second three kinds deal with the behavior of the machine:

initializations: items defining actions to be taken on entry to a frame. The cell referenced by the NumberPad is initialized by a SetInitialValue which is a sub-item of that NumberPad.

inhibitors: items defining restrictions on the values which a cell may take. The cell referenced by the NumberPad has a NumericChecker on it which refuses values over 99; the Sidedness.of.Output cell has an Interaction inhibitor on it which prevents it from taking the value 1 when the Sidedness.of.Input cell has value 2.

implications: items defining actions which must be carried out to propagate effects. The Sidedness.of.Input cell has a ConditionalPropogator implication on it which sets the value of the Sidedness.of.Output cell to 2 whenever the value of Sidedness.of.Input cell is changed to 2.

The Trillium Machine runs a frame by looking at all the primitive items in the frame — the items which are the leaves of the tree of the expansion of composites (see Figure 3).

The Trillium Machine works as follows:

- on entry to a frame:
  - run each initialization (initializes the machine state for that frame).
  - run each artwork (sets up the static graphic background for the frame).
  - run each displayer (presents the values of the cells).
  - *until the state indicates that it is time to* change frames, repeatedly:
    - run each sensor (senses the world) and actions are taken (responds to the sensors).
- to take an action, either:
  - call for a frame change — push (go deeper), pop (go back), or move (go sideways); this is done by setting the indicator in the state which is tested in the sensor loop (see above), or
  - attempt to change the value of a cell:
    - do nothing if the new value is the same as the value that is there (prevents useless changes, and breaks loops in propagating values among mutually constrained cells).
    - run each inhibitor associated with the cell (check that the value is acceptable).
    - set the new value into the cell (the change is finally made).
    - run each displayer associated with the cell (presents the new value of the cell).
    - run each implication associated with the cell (propagates effects of the change by taking further actions).

Trillium's Language of Design

Because presentation and behavior are determined entirely by the assembly of items of particular itemtypes, the language of design within Trillium is determined by the collection of itemtypes. One advantage of this is that the language can evolve by creating and modifying itemtypes. To support such evolution, designers are given tools to create new composite itemtypes. Also, new primitive itemtypes can be added by the supporters (programming in Interlisp-D), thus extending the very nature of the interfaces being described (eg. to handle new input devices - dials, or output devices - color display). This evolution of the terminology of design reflects conceptual development within the design community. For example, over time, the notion of NumberPad might change to reflect new requirements on design, such as that the layout of the buttons be variable to permit the NumberPad embedding in physical spaces of differing shape and size.

Sharing parts of a design

To share information, any frame can be given other frames, called superframes, which act as its backdrop. Superframes can have items of all kinds in them, so they impart behavior as well as presentation. Thus the frame of Figure 1 contains the buttons and the number pad, and has two superframes: the Border frame which contains the artwork around the edge, and the ChangeFrameButton label Return (this insures that it is consistently in the same place in all frames); and the Summary frame, containing the PrintRegions and artwork of the summary.

The sharing of graphical information (such as bitmaps) and reference infomration (such as colors) is accomplished with the use of service frames. These frames are used solely for storing information about shared items, and are not seen by the user during the operation of the interface. By changing these referenced items, all the items which refer to them are indirectly changed in concert, thereby maintaining consistency.

Trillium as a design environment for Functioning Frame interfaces

The criteria set out earlier for a design environment for Function Frame interfaces can now be matched against Trillium's structure and capabilities:

1. Design should not require programming: In most cases, it doesn't. Items are just placed appropriately, and they interact through the cells they reference. The behavior is built into the primitive items into which they expand.

2. Design should be fast: Creating new items takes at most minutes. The elapsed time from completion of a bit of design to experimenting with the effect is the time taken to move the finger on the mouse from the design button to the one that requests that the interface be run.

3. The environment should support the design of the interface's behavior as well as its presentation: Five of the six kinds of primitive item have active behavior built in. Composites have whatever behavior their sub-items have. Interacting behaviors (eg. the interactions of the various functions of a NumberPad) result in the higher level abstractions encompassing more complex behaviors.

4. The environment should support two-dimensional controls and displays: It does because the presentation primitive kinds (artwork, sensors and displayers) do so (see more below, concerning the embedding of Trillium in interlisp-D.)

5. The environment should support multiple views (presentation and behavior) of the machine: Frame support different views of the same collection of cells.

6. The environment should support moving among the views of the machine: The frame stack supports motion among frames.

7. Designs should make clear distinction between machine state and presentation (controls and displays): The presentation primitive kinds (artwork, sensors and displayers) are distinct from the behavior primitive kinds (initiations, inhibitors and implications)

8. Designs should support restrictions on, and interactions between, parts of the state of the machine: The inhibitors and implications do this.

9. The environment should recognize similarity among the pieces of the interfaces: Items are instances of itemtypes. Itemtypes capture similarity through their different characteristics.

10. The environment should support the construction and use of new design abstractions, particularly composition and specialization: Composite itemtypes capture constrained sets of sub-items as forming interesting abstractions. Specialization is achieved by having a more general item be the sole sub-item of the specialization, masking some of the variability by setting the values of some of the characteristics of the more general abstraction within the expansion process.

11. The environment should support incomplete specifications: Characteristics of itemtypes have default values which are used in items when values are not explicitly specified by the designer. When initially created (as opposed to copied), all of the characteristics have default values. The designer creates an item to learn what its characteristics are, and then by experimenting with the values, discovers the range of its functionality.

12. The environment should support sharing of common parts of the interface: Superframes may be shared among other frames. Service frames provide for sharing of information in terms of which other items are defined.

In addition to meeting these criteria, the acceptance and use of Trillium within Xerox argues that it is a successful design environment for Functioning Frame interfaces. Trillium is in use at more than half a dozen sites within Xerox located on two continents. Most of the interfaces for the next generation of machines has been affected by experiments using Trillium. Some of them have been entirely designed using the tool. In one case, automatic transportation of interface designs out of the (Lisp-based) Trillium design environment into the final (non-Lisp) product has even been achieved[4].

In short, the design of Trillium has been driven by the needs of fast prototyping, the interfaces to be designed, and the design process itself. By restricting attention to only the "Functioning Frame" interfaces, Trillium provides designers with a powerful tool for fast prototyping. This tool is in use with Xerox for designing interfaces for copies and printers. The ability to fast prototype has changed the quality of interfaces produced by both permitting early experience with the interface to expose problems with the design, and by permitting exploration of more of the space of possible designs.

Acknowledgements

Although originating from research at Xerox PARC, Trillium is now used, maintained and enhanced by the Trillium Community, an informal confederation of designers, supporters, trainers, researchers and management from many organizations within Xerox, all of whom are responsible for making Trillium what it is today. Also, my thanks to Bill Anderson for commenting on an earlier draft of this paper.

References

Henderson, D. Austin, Jr., Trillium: A design Environment for Copier Interfaces, videotape, CHI'83, Boston, MA. December, 1983.

4. Trillium is written in Interlisp-D, running on the Xerox 1100 series processors. Interlisp-D provides interactive graphics, symbolic representation, and an "escape clause" to programming when appropriate Trillium itemtypes are not available.