[illegible] CONSTRUCTION OF DEVELOPMENT ENVIRONMENTS

W. Harrison

Rapporteurs: Miss J. Pennington
Miss M.J. West

Issues in the Construction of Development Environments

William Harrison
IBM Thomas J. Watson Research Center
Yorktown Heights, N.Y. 10598
harrisn@ibm.com

How We Got Here

Prior to examining a number of the important technical problems presented in the construction of modern software development environments, it would pay to look at the evolution of on-line software environments to the present. The earliest of the on-line environments were those inherent in the design of the earliest of the general-purpose time-sharing systems in the middle 1960's. Typically, the time-sharing system itself provided a basic monitor which controlled and coordinated access to the input-output equipment. It provided services by which applications could communicate with their users. The time-sharing system itself also provided a primitive command shell, typically one which would accept the name of an application and some additional parameters to the application and would arrange for the application to be run. Among the first applications were the familiar non-interactive applications like compilers, linkers, etc. In addition, however, some new *interactive* applications like file editors and debuggers were also provided. These applications each provided an environment for the user: a syntax, a function set, an output format, etc. In addition to providing the behavioral characteristics seen by the user, these environments typically both defined the format in which they chose to represent the user's information and controlled the repository of the information. Editors, for example defined file contents and their mode of storage in the file system, while interpretive executors did the same for program structuring information. We find examples of this in the early CTSS time-sharing work at MIT with the TYPESET and DEBUG environments and with most implementations of APL [Apl76,Cor62].

Over time, the interactive systems in use came to be differentiated by the different human-factors made available through different treatment of communication equipment. Keystroke capture systems (often called full-duplex although that term more appropriately refers to a communications protocol) like UNIX have emphasized the system's ability to respond to each keystroke. Command-bundling systems (often called half-duplex although that term more appropriately refers to a communications protocol) like CTSS or IBM's VM/CMS encouraged applications which maximized the user's return for each command entered. Over time, both of these types of systems developed capabilities for writing "canned" command sequences (often called "exec files" or a "shell scripts"). One of the results of the stress on maximizing the utilization of a "command-bundle" was the development of a system structure that provided both sophisticated languages for exec files and the facility for the writer of the exec file to direct commands to any of several active "subcommand" environments [Cow84]. Thus, for example, the exec file might invoke the "insert" and "change" editor commands to cause the desired changes to its data which is stored as the file being edited. The marriage of these "subcommand" system structures with interactive applications simplified the extension of command systems enough to cause a local explosion in the functionality available on the systems. Some of the editing environments have become so rich that they are used as the base for completely new environments, like syntax-directed editing, word processing, poster and file management. These new environments are implemented as functional extensions of the editor, exercising the editor's command set, data management and display management from within exec files to carry out their own extended command set. This begins to provide users with a coherent interaction framework (provided by the editor) within which specialized and widely varied functional environments are provided.

The growth of data-base/data-communications oriented systems occurred in parallel with the development of these "classical" command and subcommand shell environments. These systems emphasized greater flexibility in the content and format of data, with that information choice more in the application than in the data base shell itself. The data base shell provided facilities for managing the data repository, for coordinating communication with the user, and for more sophisticated output formatting for use by the individual applications. Over time, complete control over the physical data layout moved to the data base shell itself while the logical structure of the data continued to be owned as "views" by the application. The variety of different data models made available constitute, in a sense, a large number of varied environments within the coherent interaction framework provided by the data base shell itself.

Where We Are

These environments formed the bulk of the interactive computing milieu at the advent of the era of display-oriented high-power workstations. The subcommand-style environments exploited classical shells and highly extensible editors to make possible the creation of a large number of different application environments manipulating information stored as line files. The data-base environments exploited a central repository and more powerful display management coupled with many more different data structures and representations to also provide a wide variety of application environments. There have been two primary responses to the advent of display-oriented workstations: unstructured use via windows, and structured use via structure-directed environments. Although heavy use of windows did not appear prior to the workstation, the mode of operation in which a user context-switched among several similar tasks did appeared both in multi-process form and in subcommand-environment form on shared interactive systems. A number of display-oriented data base and editing systems [Emacs,Mal81,Xed85] also made their appearance prior to the workstations, but their use was hampered by either the absence of support for keystroke-capture interaction or the absence of cursor/mouse positioning constructs.

The unstructured use of large displays relies upon the interposition of a new element, the window manager, between the application and the monitor level I/O support. The window manager supports the existence of a multiplicity of concurrent activities, each of which has the same structure as earlier application environments: a primitive shell invoking an application or environment. The window manager composes the "virtual" output streams from each activity to form the display seen by the user. In addition, the window manager routes the interactions from the user to the appropriate activity. The window manager tends to be quite dependent on device characteristics. This has tended to mask much of the device-independence provided by the monitor and we can expect that effort will be placed in the near future on attempts to incorporate a device-driver-like interface into window managers so that installations can apply them to a variety of devices.

The structured use of large displays is an area of great activity. In response to the recognition that the existence of large numbers of specialized environments will become crucial to improving the usability of computer systems, a number of groups began to explore ways in which the cost of producing a new specialized environment could be reduced. Systems like the Cornell Synthesizer [Rep84] and the GANDALF environment [Not83] represent early explorations in this area. These systems were originally targeted at providing environments for programming, and as such, they focused on support of a variety of programming languages, rather on a more general view of structural editing. However, this focus also led to the recognition that a structured environment must provide mechanisms for managing consistency and inconsistency in the information base. [Rep84].

We can begin to see a new synthesis of these approaches emerging in the architecture of structure-based environments. One of the key elements of this integration is the object-oriented methodology [e.g. Hen86] which allows the partitioning of the design of an application into independent mini-applications called objects. As in the data-base environment, each object determines its own view of the data. However, the shell, let us call it a structural shell must provide the framework into which the various types of objects fit in order to federate them together to form higher level world-views. The Framework must provide the input routing and output composition functions analogous to that provided by the window manager. Like syntax-directed editors, it must also provide information propagation mechanisms to support cross-object consistency issues, and like the data-base environment it must provide data repository functions. However, unlike many of these more closed systems, the structure-based environment must provided an open-ended structure into which new applications can be easily fit. These new applications will take the form of collections of object types. Let us consider several examples.

1. In syntax-directed program editing the objects correspond roughly to the major nonterminal symbols in the program structure. For various reasons, it may be appropriate to make the smallest objects be expressions or statements rather than individual characters, but the sense of the decomposition is unchanged. Each syntactic element has a formatting rule or procedure associated with it which interacts with the general structural layout algorithms in the Framework. It also provides a procedure for handling alteration, whether by command or by keystroke interaction. These procedures are called upon by the Framework in response to user or subcommand driven interactions. In addition, the object requires checking methods which rely upon the information propagation mechanisms of the Framework in order to track the internal consistency of groups of related objects.
2. In the realization of a spreadsheet system, the objects correspond to the spreadsheet itself and to the various items in the rows and columns of the spreadsheet. Interactions between these objects and the Framework are used to organize the display and to handle interactions as described above, and again the information propagation mechanisms are brought into play to carry out the propagation rules.
3. In the editing and display of an organization chart, the objects are the various hierarchy nodes and the individuals. Display and command interactions are as above, although there may be less need for consistency management.

Existing lines of exploration of the use of object data bases are being followed in the PECAN and GARDEN Projects [Rei84]. One of the major emphases in these efforts is the use of an object data base embodying a common semantic model as the base for widely varying views of the information. The RPDE project [Har86] is exploring the development of a structural Framework and the realization of a software development system using it. The emphasis here is on facilities to form environments as the federation of large numbers of mini-environments.

Where are we going?

There are many important technological developments that will take place in the development of Framework-style Structural Editing Environments. We will focus here on the issues and problems in a few:

- Structural Repositories
- Structural Information Propagation
- Structural Output Composition
- Structural Algorithms

For each existing approach, a number of potential or real disadvantages must be addressed before its viability can be established. These investigations will constitute important work over the near future.

Issues in Structural Repositories

The repository is the focus for residence of all of the objects manipulated by the environment. It contains information representing large complex structures, which may be trees or graphs stored with a great variety of types of objects at the nodes of these structures. These nodes exhibit variety in storage needs, connectivity, and containment relationships. In the construction of the repository we must be concerned about the granularity of the information being stored, about the ease of adding new object types or of modifying types of objects there, and about the performance of the data base under new kinds of usage loads.

Assuming that the source for a moderate size program is around 5000 objects, the classical file system has adequate performance to initiate editing of the program. However, the grain size (the size of units which have permanent names) is quite large and cannot satisfy the needs for connectivities between elements of programs like code-part to declaration-part linkages or design to implementation linkages.

Under similar assumptions, the evidence [Lin84] indicates that conventional relational data bases, while they support an appropriate granularity and degree of connectivity, have inadequate performance to satisfy the load imposed when a program is edited. In addition, the simple entity model may not adequately support the great variety of types of objects needed for these environments.

Object data bases represent an attempt to deal with the granularity, connectivity, and variety requirements, and it is hoped that mixed-granularity storage schemes [Mai85] will provide the necessary performance.

Issues in Structural Information Propagation

When an application is decomposed into objects, the burden of dealing with consistency and checking issues falls upon an information propagation strategy. It must provide communication paths must within large complex information structures, which may be trees or graphs stored with a great variety of types of objects at the nodes of these structures. The nodes in this structure exhibit variety in their needs for information from other objects and in their ability to supply information to other objects. In the propagation of information we must be concerned about the ability to hide information structures from other objects, about the localization of the descriptions of information flows, and about the the ease of adding new object types or of modifying types of objects already there.

There are two styles of information propagation strategies in use: data-oriented and control-oriented. In the data-oriented models, each node in the structure is characterized by naming the information needs and supplies of the node and writing a series of functions to compute the supplies from the needs. The functional nature of the description allows a data-flow driven style of evaluation which proceeds until the values of the data stabilize. Examples of this style of model include Attribute Grammars [Dem81], Message-Augmented Attribute Grammars [Dem85], and Attribute Grammars with Non-local Productions [Jon85]. These examples are distinguished by the locality of the individual propagation steps. Although the data-oriented models present a simple style in which the information flows may be described, their functional nature leads to serious performance problems which are under study [Rep82]. In addition, many of the data-oriented styles are monolithic in that for performance reasons pre-computations are needed which require complete descriptions of the object types and their data requirements. This severely limits the ease of extension of object bases using this style of information propagation model.

In the control-oriented models, each node in the structure is characterized by naming some events which must cause the node to be activated. The activation of an event may cause changes or seek information by activating other events with the event sequence governing the outcome. Examples of control-oriented models can be found in ALOE and DOSE [Fei83], in action-equations [Kai85], in the broadcast messages of PECAN [Rei84], and in the structure-bound messages of RPDE [Har86]. These examples are distinguished by the locality of the individual propagation steps. Although this style allows simpler expression of sequence dependent information requirements and avoids the need for global structural information, the fact that designers must concern themselves with control rather than data flows increases the effort needed to build an application and makes it more opaque.

Issues in Structural Output Composition

Output composition provides the federating mechanism for composing the individual display elements of the objects being manipulated into a coherent whole. The composed images represent portions of large complex structures, which may be trees or graphs stored with a great variety of types of objects at the nodes of these structures. These nodes exhibit great variety in their connectivity, and perhaps in their base representation as text, image, or voice as well. In the search for composition algorithms we must be concerned about the degree of control allowed to the definer of objects, about the ease of adding new object types or of modifying types of objects there, about the adaptability of the algorithms to varying information volume and display capabilities, and about the performance of the display construction.

Several techniques have been explored for image composition: panning or scrolling over a large virtual field of view, explicit control by the user, and automatic elision of contextual information. Panning and scrolling are common techniques and have the best performance characteristics, unless a large virtual field must be completely elaborated. However it affords the user minimal control over the display, generally displaying only contiguous portions of the virtual field. In addition, there is no automatic compensation for the user's focus as information volume grows.

A number of systems also allow explicit inclusion and exclusion [Tei84,Spf85] of material to be displayed. This technique also performs well and affords the user a greater degree of control over his information display, but also suffers from degradation of results as information volume grows or display size shrinks. The manipulation of windows can help alleviate this problem somewhat.

Elision-based display algorithms [Mik81] can offer a great deal of user control over the display content and by design are intended to deal automatically with growth in the volume of information. Although more expensive to run, workstations of sufficient power to use these algorithms well appear to be becoming available now. Since the system exercises more control over the formatting, it remains to be seen whether sufficient frame-to-frame consistency can be achieved to make the automatic layout non-intrusive.

Structural Algorithms

A number of the technologies under development for structure-oriented environments share a common need for tree and graph processing algorithms. Although the past decade has seen significant work on the development of good performing graph algorithms, many of them have been predicated on precomputations over the graph to be processed. In structure-oriented environments, the graph in question may occupy many files and may even be distributed over many nodes in a network, making algorithms using such global computations infeasible. The requirement for good algorithms over large graphs can be satisfied by a class of algorithms whose bound is related to the number of nodes actually needed to determine the result rather than to the total size of the graph.

[Apl76] VS APL for CMS, IBM Publication Number SH20-9067

[Cor62] Corbato F., et. al., The Compatible Time Sharing System, MIT Press, Cambridge, 1962

[Cow84] Cowlishaw, M. F., The Design of the REXX Language, IBM Systems Journal, Volume 23, No. 4 (1984).

[Dem81] Demers A., Reps T., Teitelbaum T., Incremental Evaluation for Attribute Grammars with Application to Syntax-Directed Editors, Conference Record of the 8th Annual Symposium on Principles of Programming Languages, Williamsburg Va., Jan 1981, pp. 105-116

[Dem85] Demers A., Rogers A., Zadeck F.K., Attribute Propagation by Message Passing, Proceedings of ACM SIGPLAN 1985 Symposium on Language Issues in Programming Environments, June 1985

[Emacs] Stallman R.,EMACS - The Extensible Customizable, Self-Documenting Display Editor, MIT AI Memo 519a

[Fei83] Feiler, P., Kaiser G., Display-Oriented Structure Manipulation in a Multi-Purpose System, Proceedings IEEE Seventh International Computer Software and Applications Conference, November 1983

[Har86] Harrison William, A Program Development Environment for Programming by Refinement and Reuse, Proceedings of Nineteenth Hawaii International Conference on System Sciences, January 1986, pp. 459-469

[Hen86] Hendler James A., Viewing Object-Oriented Programming as an Enhancement of Data Abstraction Methodology, Proceedings of Nineteenth Hawaii International Conference on System Sciences, January 1986, pp. 117-126

[Jon85] Johnson G., Fisher C., A Meta-Language and System for Nonlocal Incremental Attribute Evaluation in Language-Based Editors, Proceedings of Twelfth ACM Symposium on Principles of Programming Languages, January 1985

[Kai85] Kaiser G., Semantics for Structure Editing Environments PhD thesis, Carnegie-Mellon University, May 1985

[Lin84] Linton M., Implementing Relational Views of Programs, Proceedings of ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments, April 1984 , pp. 132-140

[Mai85] Maier D., Otis A., Purdy A., Object-oriented Database Development at Servio Logic, Database Engineering 8:4, December 1985

[Mal81] Malhotra A., Pazel D., Burns L., BROWSER: A Visual Interactive Database Interface, IBM Research Report RC 8935, January 1981

[Mik81] Mikelsons M., Prettyprinting in an Interactive Environment, Proceedings of ACM SIGPLAN/SIGOA Symposium on Text Manipulation, Portland OR, June 1981, pp. 108-116

[Not83] Notkin D., Structure-Oriented Users' Environments, Proceedings of the Associated Simula Users' Workshop on Program Development Tools, Lund, Sweeden, February 1983

[Rei84] Reiss S., An Approach to Incremental Compilation, Proceedings of SIGPLAN 1984 Symposium on Compiler Construction, June 1984, pp. 144-156

[Rep82] Reps T, Optimal-Time Incremental Semantic Analysis for Syntax-Directed Editors, Proceedings of Ninth ACM Symposium on Principles of Programming Languages, January 1982

[Rep84] Reps T., Teitelbaum T., The Synthesizer Generator, Proceedings of ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments, April 1984, pp. 42-48

[Spf85] ISPF/PDF Guide, IBM Publication SC-34-4011

[Tei84] Teitelman W., A Tour Through CEDAR, IEEE Software Vol 1 No 2, April 1984, pp. 44-74

[Xed85] VM/SP Editor User's Guide, IBM Publication SC-34-5220

DISCUSSION:

Dr. Harrison concluded by emphasising that we are not just interested in programming environments - i.e. environments for the construction of programs and systems. These are simply an example, and by no means the limit. We are also interested in environments for the construction of other kinds of user models, such as spread-sheets and specialised editing systems for organisation charts. In general, wherever users have a perception of structure, we would like to bring in these kinds of environments - environments where the user interacts directly with the structure, rather than interacts with a description of the structure in a linear syntactic form. This is the more important point, and so the subject of his following lecture is, "If we had tools like this, what might one do with them?"

Prof. Katzenelson asked if Dr. Harrison was designing an environment for this.

Dr. Harrison replied that, yes, they have been building a prototype environment. They have already made the second set of mistakes, and are looking forward to beginning to make the third set of mistakes! Basically, the environment is oriented towards the "federation together of lots of little objects", with general algorithms that control the overall structure. Furthermore, they have separately been building, on top of this environment, a small "world view", of sorts, that incorporates program material in terms of nested modules, procedures and communicating processes. In fact, it incorporates a number of different paradigms into the editing model.

Prof. Katzenelson asked in what way this was different from Smalltalk or a Lisp machine.

Dr. Harrison explained that his environment was different, primarily, in that it is aimed at being "open-ended". In contrast, both Smalltalk and a Lisp machine provide both permanent paradigms and a permanent reflection of those paradigms to the user. On the other hand, his environment makes use of the "object-oriented" style as part of the interface, but this is not even necessarily part of the world view that the user would see. For example, an organisation chart editor or a spread-sheet can be built, fairly straightforwardly, on top of the object-oriented framework. Moreover, there is much less emphasis on trying to put forward a particular language as a way of building things. Of course, there are members of his group who are interested in language issues. And in the following lecture he will discuss where one can take some interesting language issues in the context of environments like his.

Mr. Jackson wondered about the possibility of adding artificial intelligence to such environments, in order to give advice to the user.

Dr. Harrison thought that A.I. fitted naturally into his kind of framework. Furthermore, in environments which federate together lots of different objects, the different technologies used for building the different objects can be independent. Of course, he had not done this, yet. However, over the next 10 years, what is going to happen?

Prof. Habermann commented on how expert systems fitted into Dr. Harrison's framework. With an object-oriented approach, we are able to both distribute and localise the expertise, with the types of the objects. This is in contrast to the current approach to expert systems, where the expertise is fairly global. One can dedicate, or localise, expertise via typing.