# Productivity Engineering in the UNIX† Environment



## A High-Level Design for PAN

Technical Report



S. L. Graham

(415) 642-2059

"The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government."



Contract No. N00039-82-C-0235

November 15, 1981 - December 30, 1985

Arpa Order No. 4031





†UNIX is a trademark of AT&T Bell Laboratories

# DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

# A High-Level Design for Pan

Robert A. Ballance †

*June, 1985*

## 1. Overview

Pan is a multilingual language-based editor for manipulating tree-structured documents. The editor supports both tree- and text-oriented operations. The expected use of this system is as the front-end for a development environment in which experienced developers use several languages while creating a complex program or other document. One task of the front-end is to gather and make available information about the document for use by the developers and by other tools.

Multiple languages are handled by separating the language-specific information from the generic utilities supplied by the editor. Language-specific information, in the form of a language description, is preprocessed into tables for use by the editor. The editing component itself is table-driven. New languages can be added to the system by creating and loading a new set of tables. Pan is designed to handle different languages in different editing workspaces; switching workspaces within an editing session allows the user to edit different languages.

There are two major components to the Pan system: the editor and the table generator. The editor supplies editing operations while checking that the document meets the requirements of the language in which it is written. These requirements fall into three categories: lexical, syntactic, and contextual. (Contextual requirements are often called the "static semantics" of a language.) Information concerning errors or inconsistencies in the document is communicated to the user during the course of editing.

The editor uses both the concrete representation of a document (the representation as seen by a user of the system) and the abstract syntax of the document to implement its editing operations. The correspondence between the two representations is maintained by an incremental scanning and parsing system. The abstract syntax is in the form of an operator/phylum tree[4]. Contextual constraints are enforced using only the abstract syntax. Other tools in the environment may add information to the internal tree representation; it is the structure of the tree which is of primary interest to the editor.

The table generator takes a language description, checks it for consistency and for the properties required by the algorithms used in Pan, and then generates the tables used by the editor. In fact, the table generator is a collection of tools, many of which already exist in the UNIX[1] programming environment.

## 2. The Editor

The editor includes the basic utilities, a general-purpose text editor, a display manager, and components for incremental lexical analysis, incremental parsing, and incremental contextual constraint checking. The user interface of the editor makes use of the workstation environment by

† Sponsored by the Defense Advanced Research Projects Agency (DoD), Arpa Order No. 4031, monitored by the Naval Electronics System Command under Contract No. N00039-82-C-0235.

[1] UNIX is a trademark of AT&T Bell Laboratories.

supporting mouse/menu interaction along with keyboard interaction. Multiple windows into different edit workspaces are anticipated.

The basic utilities supply editing workspaces (buffers), interactions with long-term storage, undo processing, a help system, and command definition facilities. The editor is extensible and customizable while integrating mouse and menu forms of interaction. The design allows the user to override most system configurations with workspace-local customizations affecting the contents of menus, display options, and bindings of commands to key stroke sequences. Initially, documents are being stored as specially formatted files in the UNIX file system.

The ability to undo commands is provided by a set of conventions for communicating information to an undo processor. The undo processor is invoked by the "Undo" command. This design allows different undo strategies to be investigated. The initial strategy to be implemented simply restores the workspace to the state prior to the command; only the most recent command is undoable.

Command definition facilities allow new commands to be added to the editor. Users are free to develop their own libraries of commands. The definition facilities include procedure definition, accumulation of help and undo information, and specification of initial bindings. A general help facility is provided as a part of the editor.

A general-purpose text editor is included in the design. The text editor operates on an active region of text which could be the entire document. In this case, the entire system functions as a display-oriented text editor.

Operations upon text are themselves implemented as operations on "text-regions"; the elementary editing operations are "Insert-Region" and "Delete-Region". Character-oriented operations are modeled as operations on single-character regions. Internally, text-regions are implemented using linked lists of contiguous arrays of characters. This representation permits one to designate a particular character in the region without updating the designation until it is used. For this reason, the designation is called a sticky-pointer[2]; the pointer sticks to its designation. Sticky pointers are used to implement text-regions, to implement undo operations, and to maintain the mapping between tokens and the textual representation when necessary.

The display manager and the user interface take advantage of bit-mapped graphics and a mouse. Initially, the display manager will be text-based, with the intention of substituting an object-oriented display manager at a later time. The user interface is itself defined using the extension facilities of the system, allowing interface designers the ability to experiment with alternative dialogues between system and user.

Incremental lexical analysis maps a textual representation to the basic lexical units of the language under consideration. Pan provides a generic interface for communicating with lexical analyzers. The actual code for detecting lexical units can either be supplied as a hand-written analyzer or as a specification for the *lex*[5] lexical analyzer generator. The result of incremental lexical analysis is a list of tokens together with information as to how the token sequence has changed. This information is used by the incremental parsing algorithm to reparse the affected area. Tokens not relevant to the parser are screened out by the incremental lexical analyzer. These are placed on in a separate data structure where they can be attached to nodes in the abstract

syntax tree.

The incremental parsing algorithm allows changes in the external representation (the text) to be reflected in the internal representation (the abstract syntax tree). The algorithm uses a bottom-up (LALR) parser to perform the actual parsing.

In order to parse a change incrementally, the state of the parser (at any point) must be recoverable. When certain relationships hold between the abstract syntax, the external representation of the abstract syntax, and the grammar used to parse the external representation, the state of the parser can be recovered from the abstract syntax tree, while the abstract syntax tree can be derived directly from the parse tree. These relationships can be checked at table generation time, so that the actual conversions amount to simple table lookups. The actual parse tree is never explicitly represented. This is important because the tree that is generated directly from a bottom-up parser is generally much larger and more complicated than an abstract syntax tree. The incremental parsing algorithm of Jalili and Gallier[3] has been chosen for Pan. It can be extended easily to perform the necessary transformations.

Error recovery during parsing will use the panic mode method proposed by Robert Corbett in [1]. When an error is detected, the recovery algorithm isolates the affected area and continues. The tokens isolated during recovery, together with a message describing the error, will be attached to an "error node" in the abstract syntax tree. The user will then be able to select that node in order to see the message.

Pan provides a general notion of "attachments" to nodes in the abstract syntax. Some tokens, such as comments, are in fact attached to nodes as system-known properties. Operations to add or delete attachments are included in the repertoire of editing operations. Other tools can attach other information, provided that they ignore any attachments that they don't recognize. Attachments themselves may be either "prefix" or "postfix". To attach lexical items such as comments, the lexical item is declared as an attachment. When encountered in the token stream, the token is either attached (postfix) to the top node in the tree-building stack, or is placed on a separate stack. As nodes in the abstract syntax tree are created, the attachment stack is consulted, and if the created node accepts the kind of attachment on the stack, the connection is made and the attachment stack popped.

Using an operator/phylum tree allows the actual structure of the internal tree to be hidden from the user. Each node in the tree is represents an operator in the abstract syntax; groups of operators are called phyla. Commands which traverse trees can be defined in terms of either the operators or the phyla, such as "Next-Function", instead of being defined solely in terms of the tree structure ("Next-Sibling").

Contextual constraints are specified and checked using a new method modeled on logic programming. In this model, a database of information is built up during editing; the information in the database can then be consulted by the contextual constraints during constraint enforcement. A description of contextual constraints includes the global (context independent) axioms of a language, the definitions of facts to be entered in the database, and the actual constraints. Both fact definitions and constraints are attached to the rules in the abstract syntax.

The database itself is logically structured to reflect the naming rules of a language—in a

programming language, these are the scope rules. The constraint checker ensures that the database structure is up to date before evaluating other constraints. As facts are added and deleted from the database, a dependency tracking mechanism will ensure consistency. This model relieves the author of a language description from explicitly defining dependencies as is the case with attribute grammars. In addition, a user of the system (or other tools, including the editor itself) can access the database to get information about the document.

## 3. Language Descriptions

To add a new language to the repertoire of languages know by Pan, one must provide information about the lexical, syntactic, and contextual structures of the language. This information is gathered together as a language description. The description has separate parts for each of the above aspects, plus parts for information about external representations and pretty printing.

The lexical description of a language can be either a lex-like specification (which will be processed by *lex*) or the designation of a procedure. In the latter case, a hand-coded lexical analyzer is being supplied. Associated with the token definitions is such information as whether the token is to be screened from the parser. Also provided are standard routines for detecting lexical items not easily specifiable by regular expression such as nested sequences of brackets.

The syntactic description has three subparts: the abstract syntax, the external representation, and the grammar to use for generating a parser. This latter grammar will be passed to a parser generator to create the actual parse tables. The relationships among those three descriptions required for incremental parsing will be enforced prior to table generation.

The contextual constraint definition consists of clauses attached to rules in the abstract syntax, of axioms independent of the syntax, and of other information required by the evaluator.

## 4. Implementation

Pan will be implemented on a SUN workstation[2]. The primary implementation language will be LISP, with recourse to C for low level routines and access to the screen.

## 5. References


1. Robert Paul Corbett. *Static Semantics and Compiler Error Recovery.* PhD thesis, Electronics Research Lab, College of Engineering, University of California, Berkeley, CA., 1985.
2. Michael J. Fischer and Richard E. Ladner. *Data Structures for Efficient Implementation of Sticky Pointers in Text Editors.* Technical Report 79-06-08, Department of Computer Science, University of Washington, Seattle, Washington, 98195, June 1979.
3. F. Jalili and J. H. Gallier. "Building friendly parsers." In *Proc. ACM Ninth Symposium on Principles of Programming Languages*, pages 197-206, 1982.
4. G. Kahn, B. Lang, B. Mélèse, and E. Morcos. "Metal: a formalism for specifying formalisms." *Science of Programming*, 3:151-188, 1983.


[2] SUN workstation is a trademark of Sun Microsystems, Inc.

5. M. E. Lesk and E. Schmidt. "Lex—a lexical analyzer generator." In *UNIX Programmer's Manual: Supplementary Documents*, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California, 1984.