PROJECT LOGOS AT CASE WESTERN RESERVE UNIVERSITY

BY C. W. ROSE AND J. P. BARDEN

The descriptive title of Project LOGOS is Computer-Aided Design and Certification of Computer Systems. This account is based on unpublished papers presented at the Primary Seminar on Project LOGOS, 27-28 Octover 1971 at the University and on reports to the sponsor, The Advanced Research Projects Agency of the Department of Defense.

The principal investigator, Edward L. Glaser, arrived on the Case campus in 1967, the veteran of a number of large-scale design efforts, both hardware and software. Glaser had certain critical convictions about computer system design[1] from which he and his fellow investigators have created the motivations and objectives of Project LOGOS.

The Critique

The processes currently used to design, implement, stage, and test large-scale, general purpose computing systems will not be adequate for the "supersystems" of tomorrow. The "pencil and paper" design approach results in inability to enforce design disciplines, incomplete evaluation of interface conditions between separate modules, the introduction of coding errors in transcription of the design for computer-aided implementation, and the lack of adequate diagnostic programs, especially for the operating system.

The splitting of the design team into two insular design groups--hardware and software--causes serious and expensive problems for the integration, staging, and redesign cycle. No practical amount of simulation of the operating system and handchecking of the operating system-hardware interfaces will avoid inconsistencies, overconstraints, and underconstraints when implemented by two groups brought together for the first time. This is especially true when the software designers are dependent upon a description of the hardware which the logic designers only think they have implemented.

This situation is passed on to the user. The system is likely to be "buggy". It will probably be late, and, in these days of unbundling, the price of the software will surely reflect the length and difficulty of the labor.

There are systems in checkout containing approximately 300,000 gates in the processor alone, and larger ones are planned. The complexity of the software necessary to use this computer power efficiently, much of it in parallel or concurrent capability, will increase correspondingly. It is probably safe to speculate that these systems will never be completely checked out, or, if they are, the cost will be prohibitive.

The LOGOS Objectives

The ultimate goal is to create a design environment in which a small team of designers at interactive consoles will specify the total design, operating system and hardware, of a target system implemented by a combination of algorithms and user interaction to do precisely what it was designed to do.

The design process views the entire computer system as an entity. It analyzes, integrates, and tests the entire design history before separating the hardware and operating system components for implementation. It uses the design history to generate sufficient diagnostics to check the integrated system. The bookkeeping and analysis tasks are beyond the scope of any manual system; the entire design process must be computer-aided.

The design team must carry out the design on an interactive tool computer system where the design history is accrued in a machine-analyzable form on which implementation programs can operate directly. The design system must have several components: A data base and the necessary translators to and from the external representation of the target system; analysis programs which determine the consistency of the target system, simulate it, and evaluate its performance; transformation programs which manipulate and restructure the data base; and implementation processors which compile the software portions of the data base and implement the hardware portions in micro or macro logic elements

Finally, the target system must be certifiable. It must do exactly what its specifications, on rigorous examination, say it will do. Certifiability in this sense depends upon the validity of the total design process and, in particular, precision as to what are computer processes and their interactions.

The Representational and Structural System

The heart of the LOGOS design system is its representational and structural form. Important leads on representation had been developed by Petri,[2] Holt,[3] Karp and Miller,[4] Slutz,[5] and Luconi.[6] Glaser and his fellow-investigators, however, laid down three crucial requirements:

(1) The representation must be declarative. It must define precisely the nature and structure of computer processes and their interactions;

(2) It must be a formal system in which the interactions and logical consistency can be analyzed algorithmically; and

(3) The target systems thus represented must be directly implementable in hardware and software by automated processes.

When the third requirement, a major objective, was established as feasible, the LOGOS investigators took off from the prior learning, and researches became exciting.

The representation system was developed in graph-theoretic form.[7] There are just two basic interconnected graphs, the data graphs and the control graphs. The data graph defines the data structure, transformations upon it, and data flows. The control graph sequences the transformations and defines the control flows. Any computational task can be characterized and represented by a data and control graph. An ALGOL-like block structure may be imposed on a task, or a set of related tasks, creating an environment tree allowing open and closed subroutines and procedures. Every block can be compressed as a matter of external representation and treated as a primitive function for purposes of higher level analysis. The internal representation is still there if the block itself ever has to be redesigned.

Inherent in this representation is a view of target system structure in which algorithms can analyze the representation, block by block and as a whole, for logical consistency, determinacy, race conditions, concurrency, and other performances as functions of time.[8]

The target system in view is a collection of facilities existing on a hierarchy of layers in which an activation of a facility creates a task. From the software viewpoint, a task is a program. The view from the top of the hierarchy is that the tasks on each layer are executed on virtual machines at lower layers having machine languages for the primitive operations called from higher layers. The inside view is that each layer has a group of facilities whose tasks simulate the machines necessary to execute the primitive operations requested from higher layers.

As the onion is peeled, the actual software primitives--machine language instructions--are reached. Consistency requires extension downward through the hardware-software interface layers to the hardware logic modules and LSI chips. The goal is to force the structure of the representation into exact correlation with the structure of the algorithms of the target system. Analysis of the representation then becomes analysis of the target system processes.[9]

Each facility may have a full set of elements. The elements are (1) a set of resources, (2) control thereof, (3) an interpreter of users' directives from upper layers, and (4) a set of algorithms for performing its task and directing facilities below to do others. No facility, however, need have the full set. A storage allocator, for instance, has resources but no algorithms to be activated from a higher layer.

Concurrency deserves a special note. Concurrency or parallel processing occurs whenever the temporal order of two events is unspecified. True parallelism occurs when operations are simultaneous. Apparent or logical parallelism occurs when the operations are ordered due to limited equipment, but the order is not fixed. Maximization of true or apparent concurrency is often a goal for multiprogrammed target systems. A major problem is determining whether the results of a computation are independent of the order in which unordered operations are performed. The LOGOS representation analysis allows algorithmic detection of pathological parallelisms and provides a means for the synchronization of system primitives on lower layers.

So much for the representation system incorporating the layered hierarchy view of target system structure. It has certain advantages, chiefly its application to users, software, and hardware.

Designers can pool total system resources. This allows the analysis of resources and their allocation in the target system to prevent system deadlocks. Subsets of resources, for instance, can be allocated to a facility, managed locally for a process, and returned to the system.

The layer discipline is consistent with stepwise decompositions and associated proofs of the correctness of algorithms.

Since the internal structure of each layer is fully known, performance at each layer is predictable, and so is the performance of the total system or any portion thereof. At any interface, the higher layer users' computational requests can be modeled with an arrival distribution at any lower interface defined with facilities and a distribution of service times.

Rigorous representation of the target system allows evaluations of design alternatives and redesign without committing the hardware to implementation or the operating system to code.

The location of the software-hardware interface in the representation is a postponable decision, allowing flexibility of implementation. When the interface is fixed, both hardware and software design information can be specified in detail by automated processes.[10]

Data Bases and Their Management

PDMS, the primitive data management system for LOGOS,[11] has been installed on the PDP-10 computer which has 42 million words of on-line disk storage. It implements a set of management procedures with the capability of creating and saving controlled-access data bases. There is a general set of access and manipulation operations.

PDMS incorporates a virtual memory addressing scheme with a paging system and is not limited to the 42 million words of disk storage. It has an extensible set of procedures for the on-line creation, deletion, and updating of the stored information. It operates in a timesharing environment with local (accessible to a unique user) and global (controlled sharing by many) data bases. Linking between local and global bases is provided. There is capability for internal data and data structure description that allows references to data elements without prior knowledge of their physical structure or format.

PDMS, too, has a layered structure. The PDP-10 monitor and file-handling facilities are at the bottom. The paging system is next up as the interface between the computer's file system and the PDMS data management procedures on the next layer. Since the compilers, interpreters, and applications programs reside at higher layers, they all can execute PDMS procedures. The LEAP facilities (a list-based associative data structure and manipulation facility) of SAIL (Stanford Artificial Intelligence Language) are being implemented in the PDMS primitives. Still higher layers contain application-oriented software so that several different language processors, separately or together, can access through PDMS the same or different files.

Designers do their own work in the local bases, with read-only access to global bases. They may link local files into global bases for evaluation of designs before incorporating them in the global bases. These have a unique data processor which, using PDMS primitives, checks the validity and resolves conflicts in all "write" requests made upon the global bases.

PDMS gives users capabilities of defining and accessing general network structures based on linked list concepts, binary tree structures, indexed sequential files, or inverted file organization in terms of attributes or properties.

Present Status

The representational and structural system has been implemented on the PDP-10 system in SAIL. PDMS has been installed and debugged. Research on the detailed syntax and semantics for data operators and the integration of data structure/data operator syntaxes in the representation system is nearing completion. Research has been completed in the main, and the development phase is under way.

Logical and structural consistency analysis algorithms are being defined, and many are now implemented on the PDP-10. A unique internal documentation system is being developed. Work is progressing on the generation of target software and hardware from the representations of target systems.

LOGOS principles have been successfully applied to the design problems of interfacing IMLAC terminals to the PDP-10 system; full implementation was accomplished in February 1972.

As it stands, LOGOS is an open-loop design system with performance information feeding back to the designers directly. A proposal to close the system with design decisions based on optimization techniques for performance is being seriously explored.[12] The closed

LOGOS system should eventually produce, with human designers still making the crucial decisions, certifiable computing systems with design iteration occuring semi-automatically. The lead-times should be measured in months rather than years. The implementation on LOGOS of a primitive target computer system is about a year away.

References:

1. Glaser, E. L., Computer-Aided Design for Computing Systems, IEE Conf., Manchester, U.K., June 1969.

2. Petri, C. S., Kommunication mit Automaten, trans. by C. F. Greene, Applied Data Research, TM RADC-TR-65-377.

3. Holt, R. C., On Deadlock in Computer Systems, Doct. Dissertation, Cornell University, Ithaca, New York, January 1971.

4. Karp, R. M., and Miller, R. E., Parallel Program Schemata, J. Comp. and Syst. Sci. 3,147 (1969).

5. Slutz, D. R., The Flow Graph Schemata Model of Parallel Computation, Doct. Dissertation, M.I.T., Cambridge, Mass., September 1968.

6. Luconi, F.L., Asynchronous Computational Structures, Doct. Dissertation, M.I.T., Cambridge, Mass., September 1968.

7. Rose, C. W., A System of Representation for General Purpose Digital Computer Systems, Doct. Dissertation, CWRU, Cleveland, Ohio, August 1970.

8. Rose, C. W. and Bradshaw, F. T., The LOGOS Representation System, Primary Sem. Proj. LOGOS, CWRU, October 1971.

9. Bradshaw, F. T., Structure and Representation of Digital Computer Systems, Doct. Dissertation, CWRU, Cleveland, Ohio, January 1971.

10. Heath, F. G., and Rose, C. W., The Case for Integrated Hardware-Software, with CAD Implications, Primary Sem. Proj. LOGOS, CWRU, October 1971.

11. Pliner, M. S., PDMS--A Primitive Data Base Management System for Representing Structured Data in an Information Sharing Environment, Doct. Dissertation, CWRU, Cleveland, Ohio, September 1971.

12. Lynch, W. C., unpublished communication, CWRU, November 1971.