

PROCEEDINGS

90000

of the

**THIRD SEMINAR
ON THE
DOD COMPUTER SECURITY
INITIATIVE PROGRAM**

**NATIONAL BUREAU OF STANDARDS
GAITHERSBURG, MARYLAND**

NOVEMBER 18-20, 1980

TABLE OF CONTENTS

	<u>Page</u>
Table of Contents	i
About the Seminar	iii
About the DoD Computer Security Initiative	iv
Acknowledgements	v
Program	vi
List of Handouts	ix
"Introduction and Opening Remarks," Stephen T. Walker, Chairman, DoD Computer Security Technical Consortium	A-1
"Opening Remarks," Seymour Jeffery, National Bureau of Standards	B-1
"DoD Computer Security Initiative," Stephen T. Walker, Chairman, DoD Computer Security Technical Consortium	C-1
"Honeywell Trusted ADP Systems," Irma Wyman, Honeywell	D-1
"Computer Security Research at Digital," Paul A. Karger, Digital Equipment Corporation	E-1
"Security and Protection of Data in the IBM System/38," Viktors Berstis, IBM	F-1
"Gnosis: A Secure Capability Based 370 Operating System," Jay Jonekait, TYMSHARE, Inc.	G-1
"Computer Security Developments at Sperry Univac," Theodore M. P. Lee, Sperry-Univac	H-1
Panel: "How Can the Government and the Computer Industry Solve the Computer Security Problem?" Ted Lee, Sperry-Univac, Jim Anderson, James P. Anderson, Inc., Steve Lipner, MITRE, Marvin Schaefer, System Development Corporation, Bill Eisner, CIA	I-1
"Quality Assurance and Evaluation Criteria," Grace H. Nibaldi, MITRE Corporation	J-1
"Specification and Verification Overview," William F. Wilson, MITRE Corporation	K-1

"FDM: A Formal Methodology for Software Development," Richard Kemmerer, System Development Corporation	L-1
"Building Verified Systems with Gypsy," Donald I. Good, University of Texas	M-1
"An Informal View of the HDM Computational Model," Karl N. Levitt, Stanford Research Institute International, Inc.	N-1
"AFFIRM: A Specification and Verification System," Susan L. Gerhart, University of Southern California Information Information Sciences Institute	O-1
"An Overview of Software Testing," Mary Jo Reece, MITRE Corporation	P-1
"Update on KSOS," John Nagle, Ford Aerospace and Communications Corporation	Q-1
"Assurance Practices in KVM/370," Marvin Schaefer, System Development Corporation	R-1
"Kernelized Secure Operating System (KSOS-6)," Charles H. Bonneau, Honeywell	S-1

Third Seminar on the
DEPARTMENT OF DEFENSE COMPUTER SECURITY INITIATIVE

November 18-20, 1980

National Bureau of Standards
Gaithersburg, Maryland

ABOUT THE SEMINAR

This is the third in a series of seminars to acquaint computer system developers and users with the status of "trusted"* ADP system developments within the Department of Defense and current planning for the integrity evaluation of commercial implementations of similar systems. The two previous seminars have stressed user requirements for trusted computer systems within both the government and private sector. The first day of this seminar includes presentations by five computer manufacturers of the trusted system development activities within their organizations. Following these presentations there will be a panel discussion on "How can the government and the computer industry solve the computer security problem?" Panelists are drawn from industry and government.

The second day of the seminar opens with a discussion of the technical evaluation criteria that have been proposed as a basis for determining the relative merits of computer systems. The assurance aspects of those criteria provide the context for the second and third days of the seminar. After the context has been set, we provide an introduction to formal specification and verification technology to include descriptions of the basic types of formal specification and the implications of design and program verification. Representatives of several prominent specification and verification research groups will then discuss their systems.

As a way of rounding out the assurance criteria and providing further context for the later talks, the opening talk on the third day discusses software testing techniques. Current acquisition program testing approaches are contrasted with the formal verification techniques discussed on the second day, emphasizing the role of such testing in revealing errors which formal verification cannot detect today. Then the developers of the DoD-sponsored trusted systems will discuss the techniques they have used to assure a quality product. The seminar will conclude with a panel discussion on "Where should you put your assurance dollars?" Panelists are drawn from the verification, development and testing communities.

*A "trusted" ADP system is one which employs sufficient hardware and software integrity measures to allow its use for simultaneously processing multiple levels of classified and/or sensitive information.

ABOUT THE DOD COMPUTER SECURITY INITIATIVE

The Department of Defense (DoD) Computer Security Initiative was established in 1978 by the Assistant Secretary of Defense for Communications, Command, Control and Intelligence to achieve the widespread availability of "trusted" ADP systems for use within the DoD. Widespread availability implies the use of commercially developed trusted ADP systems whenever possible. Recent DoD research activities are demonstrating that trusted ADP systems can be developed and successfully employed in sensitive information handling environments. In addition to these demonstration systems, a technically sound and consistent evaluation procedure must be established for determining the environments for which a particular trusted system is suitable.

The Computer Security Initiative is attempting to foster the development of trusted ADP systems through technology transfer efforts and to define reasonable ADP system evaluation procedures to be applied to both government and commercially developed trusted ADP systems. This seminar is the third in a series which constitutes an essential element in the Initiative Technology Transfer Program.

The NBS Institute for Computer Sciences and Technology, through its Computer Security and Risk Management Standards program, seeks new technology to satisfy Federal ADP security requirements. The Institute then promulgates acceptable and cost effective technology in Federal Information Processing Standards and Guidelines. The Institute is pleased to assist the Department of Defense in transferring the interim results of its research being conducted under the Computer Security Initiative.

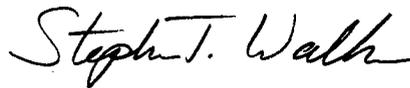
ACKNOWLEDGMENTS

A number of people in and outside of the DoD Computer Security Technical Consortium have helped to make this seminar a success. At the MITRE Corporation, Grace Nibaldi and Bill Wilson helped to organize the speakers; Karen Borgeson and Dianne Mazzone managed registration, and Annie Discepolo and George Huff prepared some of the handouts.

Also, we are grateful to Jo Ann Lorden and Greta Pignone of NBS for arranging the splendid facilities.

DISCLAIMER

The presentations in this proceedings are provided for your information. They should not be interpreted as necessarily representing the official view or carrying any endorsement, either expressed or implied, of the Department of Defense or the United States Government.



Stephen T. Walker, Chairman
Computer Security Technical Consortium

PROGRAM

November 18, 1980 Red Auditorium

9:15 Opening Remarks

Seymour Jeffery,
Institute for Computer Sciences & Technology
National Bureau of Standards

DOD Computer Security Initiative

Stephen T. Walker, Chairman
DOD Computer Security Technical Consortium

INDUSTRY TRUSTED SYSTEM ACTIVITIES

Paul A. Karger
Digital Equipment Corporation

10:45 Break

11:00 INDUSTRY TRUSTED SYSTEM ACTIVITIES - Continued

Irma Wyman
Honeywell

Viktors Berstis
IBM

Jay Jonekait
TYMSHARE, Inc.

Theodore M. P. Lee
Sperry-Univac

1:00 Lunch

2:00 PANEL: "How Can the Government and the Computer
Industry Solve the Computer Security Problem?"

Theodore M. P. Lee, Sperry Univac
James P. Anderson, Consultant
William Eisner, Central Intelligence Agency
Steven P. Lipner, Mitre Corporation
Marvin Schaefer, System Development Corporation

3:00 Break

3:15 PANEL - Continued

4:30 Adjourn

November 19, 1980

Red Auditorium

9:00 "Quality Assurance and Evaluation Criteria"

Grace H. Nibaldi
Mitre Corporation

9:50 "Specification and Verification Overview"

William F. Wilson
Mitre Corporation

10:45 Break

SPECIFICATION AND VERIFICATION SYSTEMS

11:00 "FDM: A Formal Methodology for Software Development"

Richard Kemmerer
System Development Corporation

12:00 "Building Verified Systems with Gypsy"

Donald I. Good
University of Texas

1:00 Lunch

SPECIFICATION AND VERIFICATION SYSTEMS - Continued

2:00 "An Informal View of HDM's Computational Model"

Karl N. Levitt
SRI International

3:00 Break

3:15 "AFFIRM: A Specification and Verification System"

Susan L. Gerhart
USC Information Sciences Institute

4:15 Adjourn

November 20, 1980 Red Auditorium

9:00 "An Overview of Software Testing"

Mary Jo Reece
Mitre Corporation

THE EXPERIENCES OF TRUSTED SYSTEM DEVELOPERS

9:45 "Update on KSOS"

John Nagle
Ford Aerospace and Communications Corporation

10:45 Break

11:00 KVM/370

Marvin Schaefer
System Development Corporation

12:00 "Kernelized Secure Operating System (KSOS-6)"

Charles H. Bonneau
Honeywell

1:00 Lunch

2:00 PANEL: "Where Would You Put Your Assurance Dollars?"

Panelists: Developers, Researchers, & Testers

3:00 Break

3:15 PANEL - Continued

4:15 Adjourn

LIST OF HANDOUTS

In addition to the information documented in these Proceedings, the following materials were made available at the Seminar:

Computer Security Initiative Program Trusted Systems Bibliography.

Computer Security Initiative Program Glossary.

M. H. Cheheyl, M. Gasser, G. A. Huff, J. K. Millen, "Secure System Specification and Verification: Survey of Methodologies," MTR-3904, The MITRE Corporation, Bedford, Massachusetts, 20 February 1980.

G. H. Nibaldi, "Proposed Technical Evaluation Criteria for Trusted Computer Systems," M79-225, The MITRE Corporation, Bedford, Massachusetts, 25 October 1979.

G. H. Nibaldi, "Specification of a Trusted Computing Base (TCB)," M79-228, The MITRE Corporation, Bedford, Massachusetts, 30 November 1979.

J. D. Tangney, "History of Protection in Computer Systems," MTR-3999, The MITRE Corporation, Bedford, Massachusetts, 15 July 1980.

E. T. Trotter and P. S. Tasker, "Industry Trusted Computer System Evaluation Process," MTR-3931, The MITRE Corporation, Bedford, Massachusetts, 1 May 1980.



OPENING REMARKS

STEPHEN T. WALKER
DIRECTOR, INFORMATION SYSTEMS
ASSISTANT SECRETARY OF DEFENSE FOR
COMMUNICATIONS, COMMAND, CONTROL AND INTELLIGENCE

Good morning and welcome to the third seminar on the DoD Computer Security Initiative.

My name is Steve Walker and I am Chairman of the DoD Computer Security Technical Consortium which is the sponsor of these seminars.

I am very pleased to be with you today to report on the progress that has been made in the area of trusted computer systems in the past several years and indeed in the past few months.

I am particularly pleased to acknowledge two very significant developments in the world of computer security that have made major strides since our last seminar.

First, as you can tell from looking at your program, the major external objective of the Computer Security Initiative, that of getting the computer manufacturers involved in the development of trusted computer systems is being accomplished. The credit for this belongs to many factors over and above the efforts of the Initiative but as I hope you will realize from today's presentations, the manufacturers are now seriously involved in building trusted computer systems.

The other point I want to emphasize is that the Initiative's major internal objective, that of getting the government organized to perform the technical evaluation of the integrity of computer systems is also nearing an accomplished fact. I had hoped to be able to formally announce the establishment of some form of Computer Security Evaluation Center. I cannot do that but I can describe some of the concepts being considered at high levels within the Government and I am sufficiently optimistic about these developments that I am willing to predict that within a year there will be a technical integrity evaluation process in being to serve the DoD and perhaps one to serve the Federal Government as a whole.

I am excited about both of these developments because of the significant impact that they will have, indeed are now having, on the quality of computer systems for all users.

I would like now to review with you some of the background leading up to these developments and to share with you my feelings about where we are and where we may be going.

Following this we will hear from 5 manufacturers' representatives about trusted computer system activities in their companies.

This afternoon we will have an expanded version of the "Ted and Jim" Show from the last seminar. We have a select panel of cynics to discuss the status and pitfalls of developing and using trusted computer systems.

Tomorrow we will focus on the area of formal specification and verification, hearing from several researchers. Thursday we will hear the experiences of several of the DoD system development efforts in their use of these verification tools.

OPENING REMARKS

THIRD SEMINAR ON THE
DEPARTMENT OF DEFENSE COMPUTER
SECURITY INITIATIVE PROGRAM

Seymour Jeffery
Institute for Computer Sciences and Technology
National Bureau of Standards

November 18, 1980

On behalf of the Directors of the National Bureau of Standards and the Institute for Computer Sciences and Technology, I would like to welcome you to this Third Seminar on the Department of Defense Computer Security Initiative Program. ICST is pleased to sponsor a forum for DOD to present the progress made in the important area of computer security, DOD has defined the term "trusted" ADP System as one which satisfies the DOD requirements of simultaneously processing multiple levels of classified or sensitive information. We at NBS feel there is a strong need to transfer this technology to the non-DOD Government sector as well as to private industry so that the technology may be used to satisfy their computer security requirements. I believe that this transfer of technology is an important part of the NBS program in computer security.

This is the third DOD-NBS Seminar on trusted operating systems. Some of you are new to the field; some of you have been involved as long as I have; and some perhaps even longer.

Dr. Willis Ware of the RAND Corporation, who keynoted the first seminar in this series, was the first to articulate the computer security problem and to outline some approaches to solving it. In his opening remarks at the first seminar in January, 1979, Dr. Ware reviewed the computer security problem as he perceived it in 1967. He noted the successes and the failures in solving the problems during the last 12 years. I, too, would like to spend a few minutes looking back at one of the milestone events in the computer security area. This event was the Controlled Accessibility Workshop co-sponsored by NBS and ACM and held at Rancho Santa Fe, California in the Fall of 1972. Controlled Accessibility was the term used to denote the set of controls which could be used to limit the access to, and use of, a computer only to authorized users performing authorized activities. The Workshop brought together 65 computer security technologists and managers. The group was tasked with identifying technical and management controls which would provide

the desired protections. The controls were divided into five areas:

- Audit
- EDP Management
- Personal Identification
- Security Measurement
- Access Controls

Since this seminar emphasizes the automated controls of a "trusted computer operating system", I want to spend a few minutes describing the findings of the Access Controls Working Group of the 1972 Workshop. This group was led by Clark Weissman of System Development Corporation. The goal of the Working Group was to define the nature of an automated access control mechanism and to identify the technology involved in ensuring secure computer system operation. Regarding the primary threats which must be combatted by automated access controls, the group wrote, "System security is most threatened by the vulnerability of the internal access control mechanism to unauthorized modification by subversion of normal internal system service, or exploitation of system weaknesses, such as incomplete design and coding errors."

Leading towards the technology which will be discussed here the next three days, the following points were noted in the Report of the Controlled Accessibility Workshop published by NBS in 1973, and I am sure you will hear several of these repeated in subsequent sessions this week.

- One - Control mechanisms should be formal and always invoked, never by-passed for efficiency or other rationalized reasons.
 - Two - The design must accommodate evaluation and easy system maintenance.
 - Three - The principle of "least privilege" should be applied to system operation.
 - Four - The computer system vendor will have the ultimate responsibility for delivering systems that can be operated securely.
- Finally - Product acceptance will require application of certification techniques.

It has been eight years less 23 days since the Controlled Accessibility Workshop. In some areas technology has advanced rapidly. The capability of micro-computers has risen dramatically. The Federal Data Encryption Standard is now available in 13 different electronic devices which have been

validated by NBS. The need for such a standard was identified at that Workshop. In other areas technology has made only modest advances. For example, automated personal identification through voice or signature recognition. In the area of "trusted" systems, DOD has carried the research and development initiatives. In other areas identified as having high priority at that Controlled Accessibility Workshop, NBS has initiated the development of technical standards and management guidelines to address computer security requirements. These areas include risk analysis, contingency planning, security audit and evaluation, data communication and storage protection and physical security. We have had some successes in these areas. At the first DOD-NBS seminar, Willis Ware challenged NBS to, I quote, "STEP OUT SMARTLY" in developing new and innovative standards in computer security. We are pleased to sponsor this forum so that the technology being developed to meet DOD's needs is also made available to satisfy similar needs in the private and public sectors. Security is not well understood, and in some cases not well accepted, outside DOD. We feel it is important that the vendors and the users of the technology underlying "trusted systems" exchange their views in an open forum.

As we listen to the needs of the DOD and the private and public sectors, we will initiate a plan for a tenth anniversary workshop of the work that was started in 1972.

DoD
**COMPUTER
SECURITY INITIATIVE**

**...TO ACHIEVE THE WIDESPREAD
AVAILABILITY OF TRUSTED
COMPUTER SYSTEMS**

Stephen T. Walker
Chairman
DoD Computer Security
Technical Consortium

**SEMINAR
ON
DEPARTMENT OF DEFENSE
COMPUTER SECURITY
INITIATIVE**

NOVEMBER 18-20, 1980

**NATIONAL BUREAU OF STANDARDS
GAITHERSBURG, MARYLAND 21738**

**COMPUTER
SECURITY INITIATIVE**

**TRUSTED: SUFFICIENT HARDWARE AND
SOFTWARE INTEGRITY TO
ALLOW SIMULTANEOUS USE
AT MULTIPLE SECURITY/
SENSITIVITY LEVELS**

WIDESPREAD: COMMERCIALY SUPPORTED

COMPUTER SECURITY

PHYSICAL SECURITY

ADMINISTRATIVE SECURITY

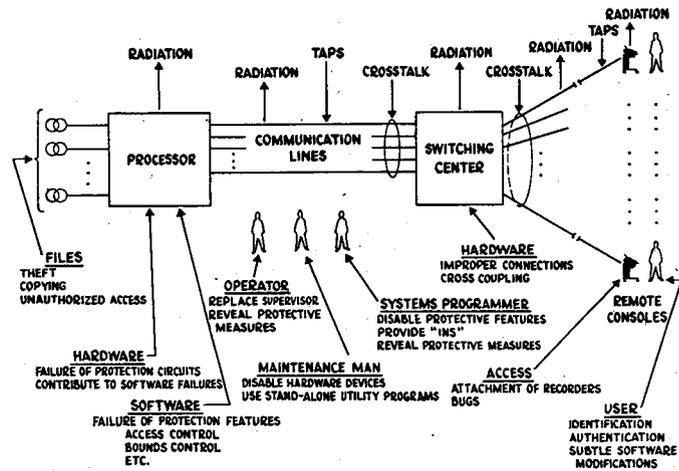
PERSONNEL SECURITY

COMMUNICATIONS SECURITY

EMANATIONS SECURITY

**HARDWARE/SOFTWARE
SECURITY**

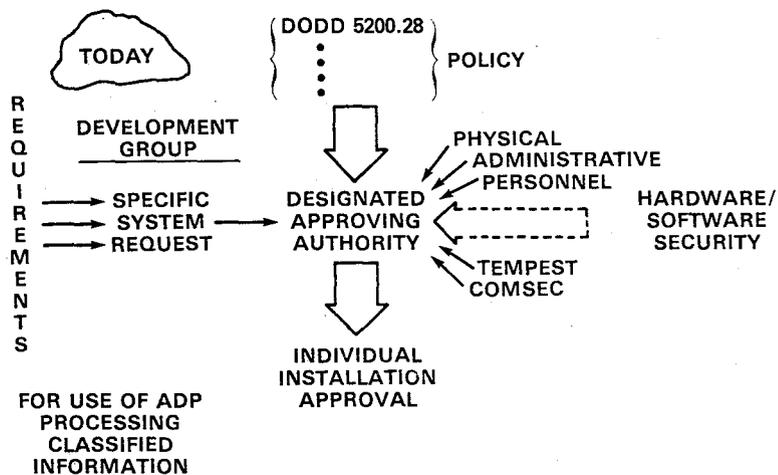
COMPUTER NETWORK VULNERABILITIES



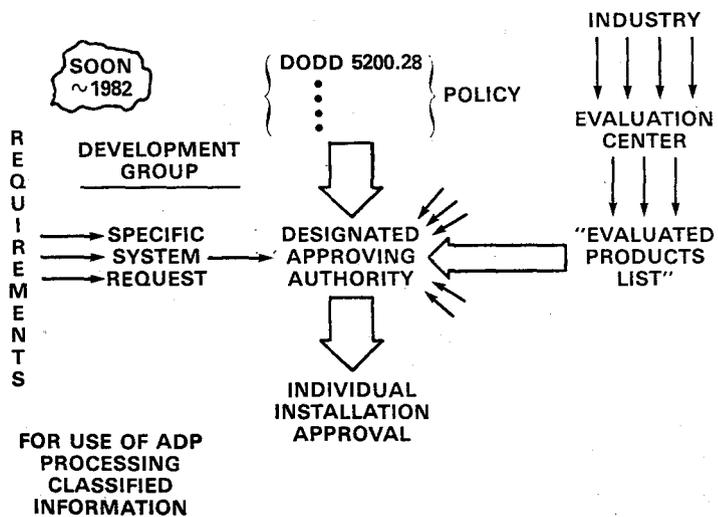
HARDWARE/SOFTWARE SECURITY

- DEVELOP A COMPUTER SYSTEM THAT WORKS CORRECTLY WITH RESPECT TO THE CONTROL OF INFORMATION FLOW

APPROVAL FOR DoD USE



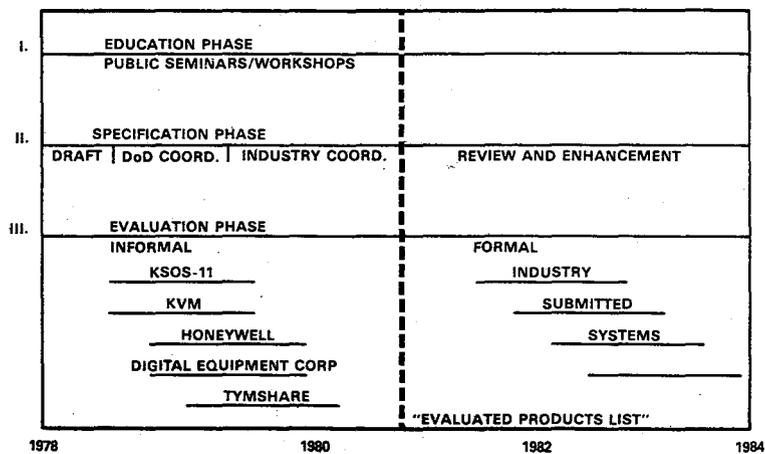
APPROVAL FOR DoD USE



EVALUATED PRODUCTS LIST

CLASS	TECHNICAL FEATURES	EXAMPLES	POSSIBLE ENVIRONMENTS
1	-	MOST COMMERCIAL SYSTEMS	DEDICATED MODE
2	FUNCTIONAL SPECIFICATION REASONABLE PENETRATION RESULTS	"MATURE" "ENHANCED" OPERATING SYSTEM	BENIGN, NEED TO KNOW ENVIRONMENTS
3	REASONABLE MODERN PROGRAMMING TECHNIQUES LIMITED SYSTEM INTEGRITY MEASURES	MULTICS	AF DATA SERVICE CENTER TS-S
4	FORMAL DESIGN SPECIFICATIONS SYSTEM INTEGRITY MEASURES		NO USER PROGRAMMING TS-S-C
5	PROVEN DESIGN SPECIFICATIONS VERIFIABLE IMPLEMENTATION LIMITED COVERT PATH PROVISIONS	KSOS KVM	LIMITED USER PROGRAMMING TS-S-C
6	VERIFIED IMPLEMENTATION AUTOMATED TEST GENERATION EXTENDED COVERT PATH PROVISIONS REASONABLE DENIAL OF SERVICE PROVISIONS		FULL USER PROGRAMMING TS-S-C

COMPUTER SECURITY INITIATIVE



COMPUTER SECURITY INITIATIVE

DoD R&D IN 1970s

OPERATING SYSTEMS

- MAJOR EMPHASIS
- MOSTLY SOFTWARE, SOME HARDWARE

APPLICATIONS

- MINOR FOCUS UNTIL LATE 70s

VERIFICATION TECHNOLOGY

- COMPUTER SECURITY WAS ONE AMONG MANY POTENTIAL USERS

DoD R&D THRUSTS IN 70s

OPERATING SYSTEMS

EXAMPLES:

KERNELIZED SECURE OPERATING SYSTEM
KERNELIZED VM370 SYSTEM

DRIVEN BY

WHAT CAN WE HOPE TO ACHIEVE IN
3-5 YEARS?
WHERE WOULD WE LIKE TO BE IN
5-8 YEARS?

INTENDED AS DEMONSTRATION
CAPABILITIES, NOT AS COMPETITION
WITH MANUFACTURERS

DoD R&D THRUSTS IN 70s

APPLICATIONS

GUARD, SECURITY FILTERS

- BETWEEN EXISTING SYSTEMS

COMMUNICATIONS, FRONT END SYSTEMS

- ACCESS PROTECTION TO EXISTING SYSTEMS

MULTIPLE SINGLE-LEVEL FUNCTIONS

- KVM

TRUSTED MULTILEVEL SYSTEMS

- SPECIAL PURPOSE - MESSAGE HANDLING
- GENERAL PURPOSE - DBMS

DoD R&D THRUSTS IN 70s

VERIFICATION TECHNOLOGY

- EVOLVED FROM EFFORTS TO BUILD "CORRECT PROGRAMS"
- SEVERAL APPROACHES ARE EVOLVING
NONE HAVE COMPLETE PACKAGE
- PROGRESS EMPHASIZED IN REST OF THIS SEMINAR

DoD R&D IN 1980s

OPERATING SYSTEMS

RELY MAINLY ON INDUSTRY
EVOLUTION
SOME SPECIALIZED DEVELOPMENT

APPLICATIONS

MAJOR EMPHASIS BY R&D AND
USER COMMUNITY

VERIFICATION TECHNOLOGY

MAJOR THRUST BEGINNING
ENSURE UNDERSTANDING OF
PRODUCT INTEGRITY

TECHNOLOGY EVOLUTION

- **HARDWARE CHEAPER, MORE POWERFUL**
 - COMPLEX SOFTWARE FUNCTIONS MOVING INTO HARDWARE
- **BETTER UNDERSTANDING OF OPERATING SYSTEMS**
 - WHAT IS NEEDED, HOW TO PROVIDE EFFICIENTLY
- **ASSURANCE TECHNIQUES IMPROVING RAPIDLY**

INDUSTRY THRUSTS IN 70s

DRIVING FORCE: IMPROVE PRODUCT QUALITY

- EASE MAINTENANCE, MODIFICATION
- IMPROVE PERFORMANCE
 - FLEXIBILITY
 - INTEGRITY
 - SECURITY

CONSTRAINT: EXISTING CUSTOMER BASE

EVOLUTIONARY VERSUS REVOLUTIONARY

MANUFACTURERS PROGRESS

DIGITAL EQUIPMENT CORPORATION

HONEYWELL CORPORATION

**INTERNATIONAL BUSINESS
MACHINES CORPORATION**

TYMSHARE CORPORATION

SPERRY UNIVAC CORPORATION

HONEYWELL

TRUSTED ADP SYSTEMS

Irma Wyman

INTRODUCTION

It is my pleasure, and my privilege, to share with you this morning, the position, and philosophy of Honeywell Information Systems regarding computer security. And, also, to let you know about our current activities and future goals in this important area.

Slide 1

POSITION

Computer Security theorists tend to view computer security in absolute terms,...and properly so. Their visions of absolutely secure hardware/software systems provide us with the conceptual upper limits of computer security on what we at Honeywell believe must be viewed as a spectrum...the "Perfect Ten" on a scale of "zero" to "ten".

Slide 2

Honeywell's position with respect to computer security is that computer hardware/software products should provide the systems integrity necessary to reduce risks of unauthorized penetration to a level acceptable to the intended product markets, subject to the constraints of technology and acceptable costs and performance.

We believe this position, and even more importantly our activities in pursuit of the elusive "Perfect Ten", to be supportive of the Computer Security Initiative Program's objective of achieving "trusted" ADP systems.

ISSUES

Before describing our philosophy and current activities, I'd like to comment on three of the issues that must be resolved before a "trusted" ADP system is likely to become a commercial reality.

Slide 4

1. Perhaps the most obvious issue, and one that I understand will be addressed in some detail during the next three days, is that of PROVABILITY. Edsger Dijkstra (well known for his contributions to the concepts of structured programming) once suggested that "Testing only reveals the present of 'bugs', not their absence". What then are the criteria and mechanisms to be used to prove a system as "trusted"? Furthermore, should this be a binary designation? Should there, in fact, be a hierarchy of "trustworthiness"?

2. A second issue is concern with terminology--- specifically the terms "system integrity" and "adequate" (or "sufficient" as used in the Initiative Program's definition of a "trusted" system).

- I know of no generally accepted definition of "system integrity" and strongly suspect that if ten of us here were to write down and compare definitions, we'd come up with at least nine different answers.
- "Adequate" and "sufficient" are relative terms to start with. When applied to various definitions of "system integrity", the resulting differences of opinion should be no surprise to anyone.

PHILOSOPHY

Honeywell's philosophy, our school of thought, is that system integrity--"trusted" ADP systems--will be achieved through a hardware/software mechanism called a SECURITY KERNEL, based on the REFERENCE MONITOR concept, and implemented through DESCRIPTOR-DRIVEN PROCESSORS.

Honeywell and other vendors can, of course, offer their own definitions. I suggest, however, that groups in the public and private sectors, such as yourselves, address this issue to mitigate nature vendor biases. The December 1979 issue of the EDP Analyzer might be useful in such an endeavor.

3. The third issue deals with the relationship between technological advancement and practical business economics. Technological advancement in computer security (or any other area) is largely dependent upon

the resources devoted to that end. Allocation of resources, of course, is in turn dependent on management's estimates of return on investment as the advanced technology is applied to perceived needs in the markets served. As I think you will see at the conclusion of this presentation, Honeywell has perceived an increasing demand for improved computer security and is aggressively addressing this issue.

Slide 5

Numerous computer security groups in studying access control mechanisms recognized the need for provability of correctness. This led to the recommended technical approach that computer security must start with a statement of an abstract, ideal system. This ideal system became known as the reference monitor. The reference monitor abstraction permits or prevents access by subjects to objects, making its decision on the basis of subject/object identity and the security parameters of the subject and the object. The implementation of the abstraction both mechanizes the access rules, and assures that they are enforced within the system. The mechanism that implements a reference monitor must meet three requirements: Complete mediation, isolation and verifiability. These requirements demand that the reference monitor implementation include hardware as well as software. The hardware/software mechanism that implements the reference monitor abstraction is called a Security Kernel. It is felt that to implement a trusted ADP System, the Security Kernel concept must be used. And to implement the kernel, descriptor-driven processors must be utilized.

Slide 6

The kernel mechanism must provide for complete mediation, and be invoked on every access by a subject to an object.

Slide 7

The kernel mechanism must provide for complete isolation for itself, its data base, and for all users.

Slide 8

The kernel mechanism must be small, simple and understandable so that it can be completely tested and verified that it performs its functions properly. This kernel mechanism is the key to certifiable, multi-level security and a trusted ADP system.

Slide 9

One of the current challenges in verification and certification is to find an agency or committee which will - and can - with authority - say that: The design is sound, the implementation is correct, the verification methodology is correct, and it has been correctly applied to proving the design and implementation of the trusted ADP system.

Now, let us examine Honeywell's involvement in trusted ADP Systems.

Slide 10

- 1964 - Start of the MULTICS Program - An architecture designed for controlled sharing from the beginning. Utilize modified 600.
- 1968 - The GCOS II O.S. - which included enhanced software security features.
- 1969 - MULTICS - Became operational on a G-645.
- 1969 - GCOS III - Enhance software security on GCOS.
- 1972 - Implement Multics on a 6190. Additional access control implemented in hardware.
- 1972 - The GCOS III O.S. - Provided the vehicle to investigate and enhance software access control mechanisms.
- 1974 - Multics implemented on the 6800. Speed up access control mechanism. Develop an access isolation mechanism to enforce DOD security policy.
- ALCOM 700 - Design and implement a secure remote batch terminal. The only computer system to be certified secure. Still the only certified secure ADP system.
- 1974 - Project Guardian - Based on Multics - was begun with the objective to build a provably secure, general purpose system with a secure front end processor.
- 1977 - Level 6 SCOMP Program was initiated to develop secure communications processor.
- 1977 - CP-VI Plus Level 66 - Implementation of new, controlled sharing access mechanisms on Level 66 hardware in order

to provide access control enhancements and provide an upgrade path for CP-V users.
1979 - DPS-8/GCOS-8 was announced - New product with advanced controlled sharing, access mechanisms to replace the Level 66 and GCOS III. Security is a primary design goal.

Slide 11

Let us review some of these significant events in more detail. First, Multics.

Multics was designed as a general purpose computer utility with interactive processing and controlled sharing of all information. Data security was a primary design goal. This controlled sharing is achieved by a unique file system with virtual memory integration and hardware enforced access controls.

Slide 12

To enforce complete mediation and allow controlled sharing of all information, Multics utilizes descriptor driven processors with segmentation. Each segment has an access control list. The access control list is checked by the system when the segment is opened. The system then sets the access control bit into a descriptor. Thereafter, the hardware enforces access controls on every reference.

In addition to the access control list, the Multics access isolation mechanism extends the basic access controls of Multics to insure isolation of users according to DOD security policy. Each user and segment is assigned an access isolation mechanism access code which enforces eight levels of clearance and eighteen need-to-know category sets. This access code is checked when a segment is opened, when the access control list is changed, and when information is exchanged between users in the system.

Slide 13

For isolation, the Multics structure provides for eight hierarchical rings which separate the operating system from system utilities and users, and the users from each other, providing for complete hardware enforced isolation.

Slide 14

For example:

Ring 4 contains procedures "A-1"

Ring 5 contains procedures "B-1"

Procedures "B-1" has read permissions for data in ring 4.

Slide 15

Procedure "B" requests access to procedure "A" data. The request is made via the control mechanism in ring zero.

Slide 16

The control mechanism in ring zero "O.K.'s" the request and verifies that a gate mechanism exists between procedure "A-1" and "B-1".

Slide 17

This gate mechanism permits procedure "B" to "read" data from procedure "A". (Via Program Q).

Procedure "A" actually writes the data via the gate to procedure "B".

It should be noted that if the access isolation mechanism were activated then the request would have been denied. (It would have been a lower clearance level attempting to read data at the higher clearance level.)

This access isolation mechanism was defined as a part of Project Guardian.

Slide 18

Project Guardian, started in 1974, was the first attempt to implement a kernel on a Honeywell system. As a result of government funded studies, the Multics

system was selected as the host computer for the design and implementation of a kernel mechanism which would meet the three requirements of: Complete Mediation, Isolation, and Verifiability. Unfortunately, Project Guardian was cancelled because of funding problems prior to achieving its ultimate goal.

However, as a result of Guardian, the Multics system has been approved to run in a two level security mode, simultaneously servicing Secret and Top Secret users. Project Guardian demonstrated that complete mediation of access, isolation from unauthorized access, and verifiability - (that is, provability and testability) - of a security kernel was possible. In addition, a proof methodology was defined and a secure front end processor was defined.

Slide 19

The secure front end processor was based on a commercial Tempest Honeywell Level 6 minicomputer which was to be enhanced by a hardware security protection mechanism and special kernel software. To understand how the security protection mechanism was to be implemented, let us quickly review the standard Level 6 memory management mechanism.

Slide 20

Memory management on the Level 6 is embodied in a hardware memory management unit, which provides for a four ring architecture, and a descriptor driven processor with segmentation. The descriptors define location, size and access controls very similar to a miniaturized version of the Multics access control mechanism, and ring architecture. For the secure front-end processor, this memory management unit was to be replaced by a "virtual memory interface unit" and extra hardware called a "security protection module" which was to implement kernel functionality.

Slide 21

SCOMP -

The Secure Communications Processor (SCOMP) project was started after the Guardian project. The object of this project is to pick up the secure front end

processor development after it was stopped under project Guardian and make it more general purpose. In other words, to design and implement a provably secure multi-purpose minicomputer, also known as a Trusted Computing Base (TCB).

The SCOMP consists of a Level 6 central processor with a virtual memory interface unit and security protection module which runs with all other Level 6 hardware. The virtual memory interface unit replaces the memory management unit previously mentioned. The security protection module consists of additional boards. The design of the security protection module is based on the reference monitor concept and implements a large portion of the security kernel functionality in hardware. A key point is that hardware access controls were extended to include I/O.

That portion of the security kernel functionality which is not implemented in hardware is handled by a software security kernel termed KSOS-6.

Some people use the acronym KSOS-6 to refer to both hardware and software.

Slide 22

The software security kernel (KSOS-6) resides in ring zero of the SCOMP system. This software security kernel works in conjunction with the hardware kernel which is called the security protection mechanism. In the outer two rings, user application and system utilities operate. As a part of this project, certain specialized trusted system routines are being designed and implemented to form a comprehensive trusted computing base. A more detailed presentation on the SCOMP will be given later in this seminar.

While working on joint effort projects, such as SCOMP and Guardian, Honeywell has also been working on system integrity control mechanisms.

Slide 23

We have applied the knowledge and experience gained through these efforts to our product lines.

And, it is felt that our recently announced DPS-8 - with an evolving GCOS-8 - is "potentially" a provable trusted ADP system.

Slide 24

DPS-8 hardware supports:

1. Virtual Memory
2. Security Mechanisms that are emphasized in the hardware
3. Domain Protection

Slide 25

The DPS-8 virtual memory architecture allows for eight trillion bytes of virtual memory. All of the security access control mechanisms are implemented in the central processing unit and in the I/O processing unit, and reference a common set of working space tables.

Slide 26

The working space tables are controlled by descriptors which provide for segment definition and access controls. This access control mechanism accomplishes the first requirement of a security kernel, to provide for complete mediation. The descriptor mechanism provides protection, segment boundary control, access control, and the ability to reduce the size or the access rights to a segment.

Slide 27

The domain mechanism satisfies the second requirement of a security kernel: to provide absolute hardware enforced isolation. A domain is a logical system territory consisting of all segments referenced by a user's procedure. This mechanism differs from Multics in that there is no implied hierarchical ring structure.

Slide 28

The domain concept allows information to be delivered strictly on a need-to-know basis for process execution. All domain context switching is handled by hardware.

Slide 29

For Example:

The domain of procedure "A" is comprised of the program "A", systems software working space tables, and different parts of a data base.

Slide 30

The segments of domain "A" may in turn be shared by other domains all under hardware enforced access control mechanisms. This means that any given entity needs to exist only once within the operating system.

Slide 31

The domain mechanism also permits temporary sharing between domains.

For example, procedure "A" desires to query procedure "B" in domain "B".

Slide 32

Procedure "B" within domain "B", if the access permissions are acceptable, will increase its domain territory to include the argument segment of procedure "A". Procedure "B" then deposits the requested information in a temporarily shared portion of domain "A".

Slide 33

Upon completion, procedure "B" then executes a command to "shrink" its domain back to its original territory.

Slide 34

We believe that the DPS-8 architecture provides the base on which a secure system can be built, and that it

will prove to be the key to an effective, flexible, multi-level "trusted" ADP system. Possibly a "Perfect 10".

As indicated by our past and current activities, Honeywell has been, and is now, committed to aggressive action in responding to the needs for improved computer security.

Slide 35

To reduce the costs associated with providing effective security features in our computer products.

Slide 36

To provide mission optimized, multi-level solutions to the problems of computer security.

Slide 37

To optimize system efficiency in the multi-level "trusted" ADP system environment.

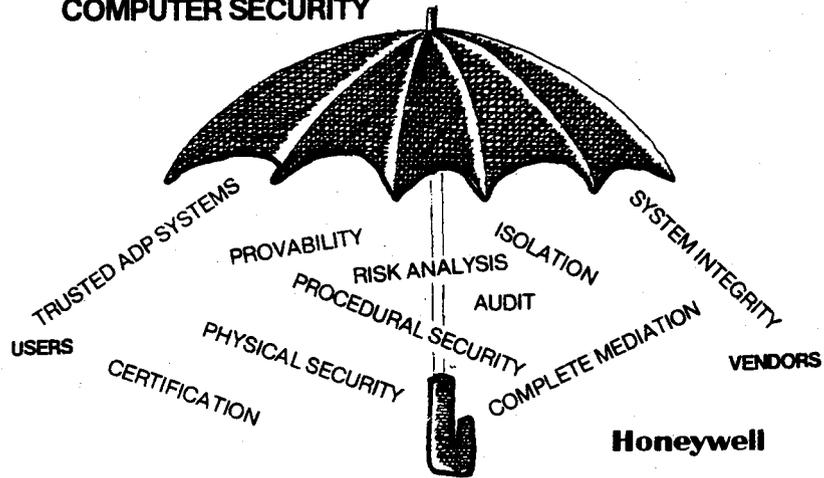
Slide 38

To work cooperatively with government and private industry to resolve the issue of provability--to establish the criteria, process and accrediting authority for "trusted" ADP systems.

We believe we offer the most secure systems available today, and are determined to maintain our leadership position in the future.

Honeywell
Trusted ADP Systems
The Leader

COMPUTER SECURITY



2

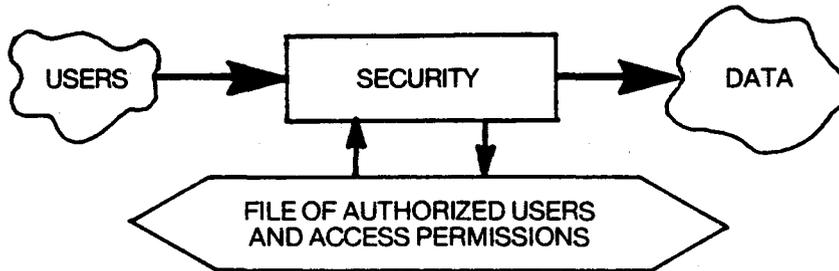
HOW DO YOU ACHIEVE TRUSTED ADP



4

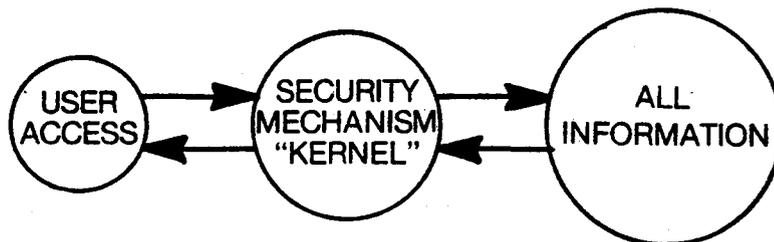
PRINCIPLES OF SECURE SYSTEM

- COMPLETE MEDIATION
- ISOLATION
- CERTIFICATION/SIMPLICITY



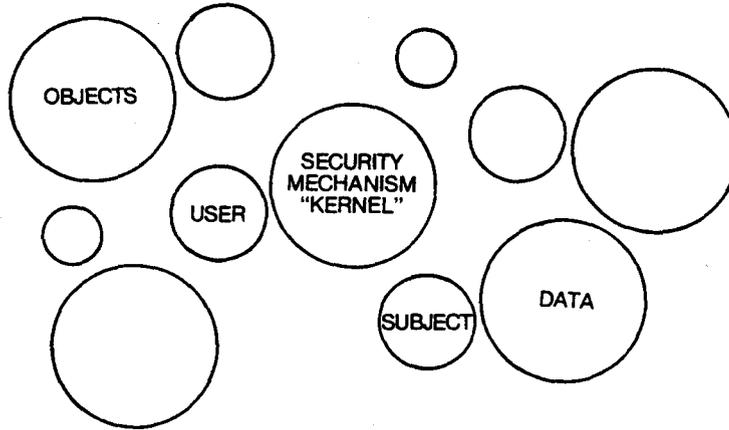
5

COMPLETE MEDIATION



6

ISOLATION



7

VERIFIABILITY/CERTIFIABILITY/ACCREDITATION

SIMPLE AND STRAIGHTFORWARD
KERNEL
TO PERMIT ANALYSIS

Honeywell

8

“KERNEL”

IS KEY TO

CERTIFIABLE MULTILEVEL SECURITY

Honeywell

1964
● MULTICS
● DESIGNED
MODIFIED G600
1968 GCOS II—G635 ● 1969
GCOS III—G635 ● 1969
MULTICS—G645 ● 1972 GCOS III
—H6000 ● 1972 MULTICS—
—H6190 ● 1974 PROJECT
GUARDIAN ● 1975 ALCOM
700 ● 1977 SCOMP—
LEVEL 6 ● 1977
LEVEL 66—CPVI
● 1979 DPS8
—GCOS
8

MULTICS
SECURITY

11

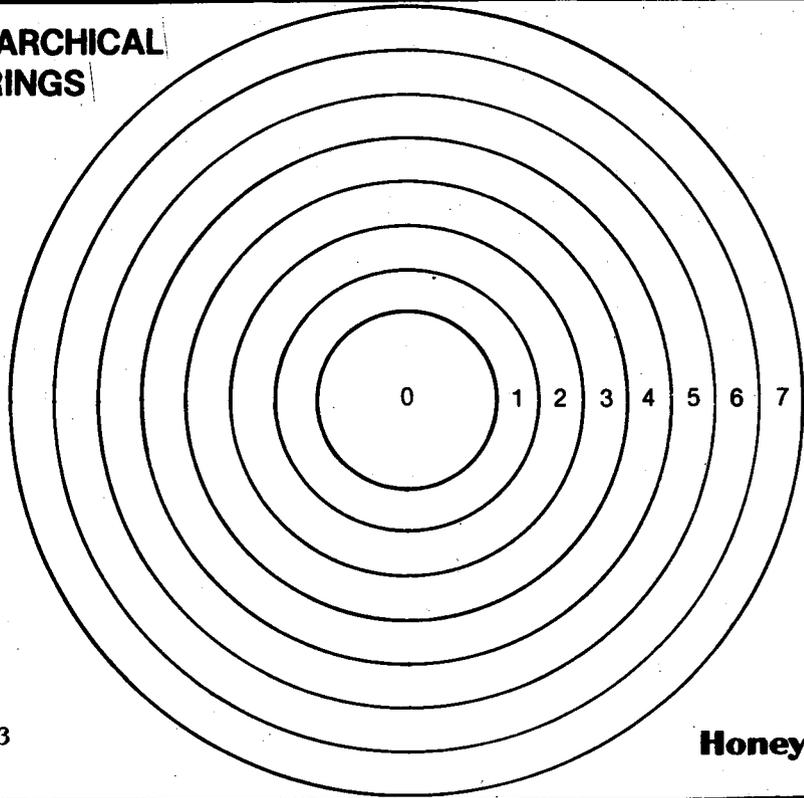
MULTICS-THE SECURE SYSTEM

SECURITY MECHANISMS UNIFORMLY APPLIED

- ACCESS CONTROL LISTS
- ACCESS ISOLATION MECHANISM
- RING PROTECTION MECHANISM
- PASSWORDS
- AUDIT TRAILS
- USER DEVICE ATTACHMENT CONTROLS

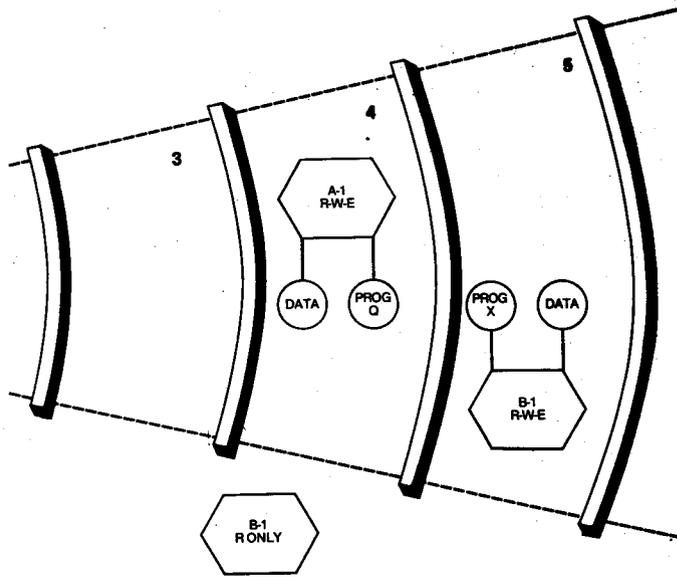
12

HIERARCHICAL RINGS



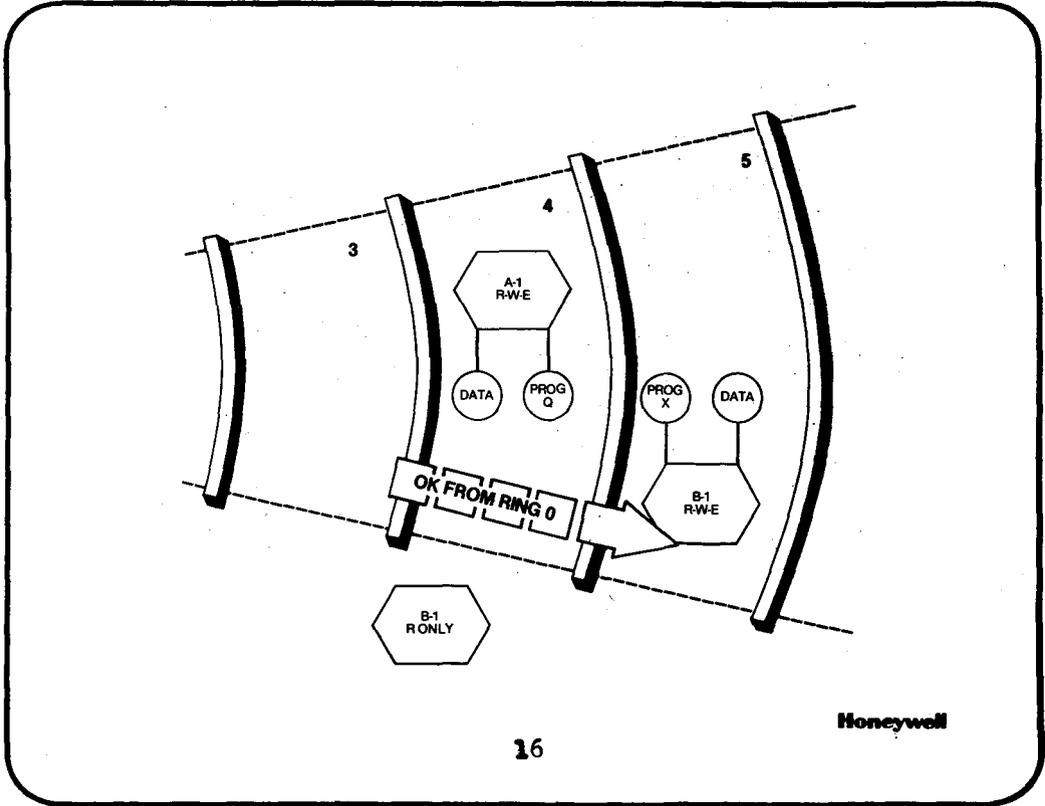
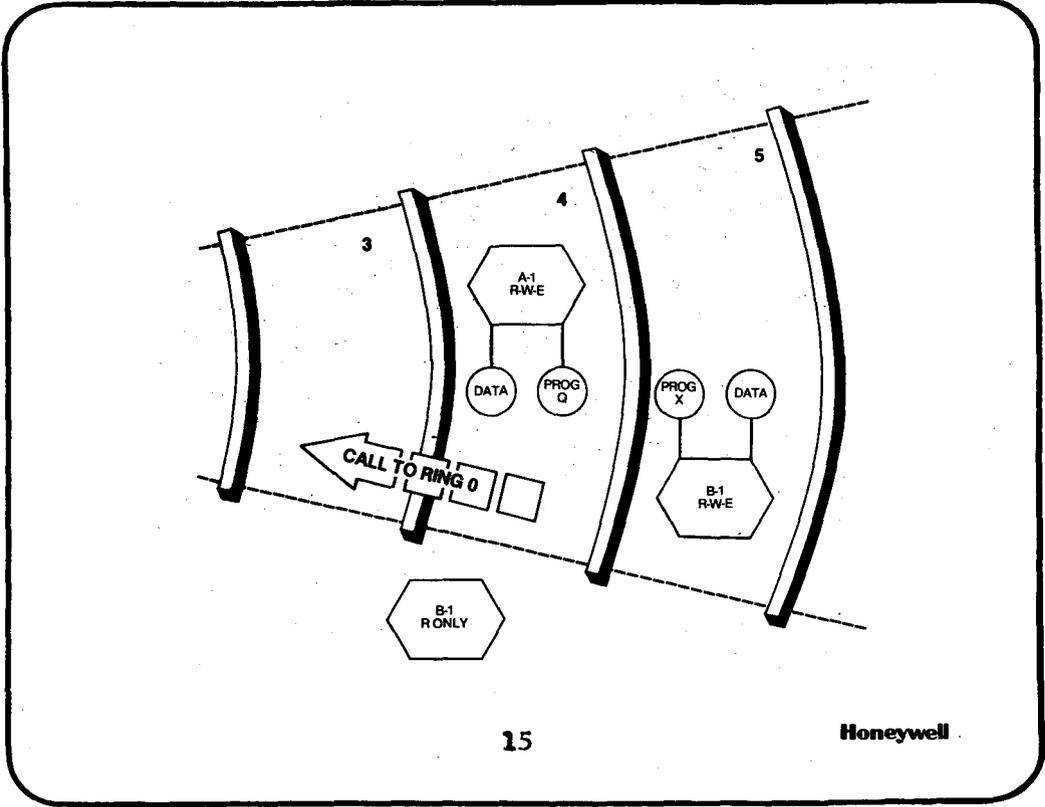
13

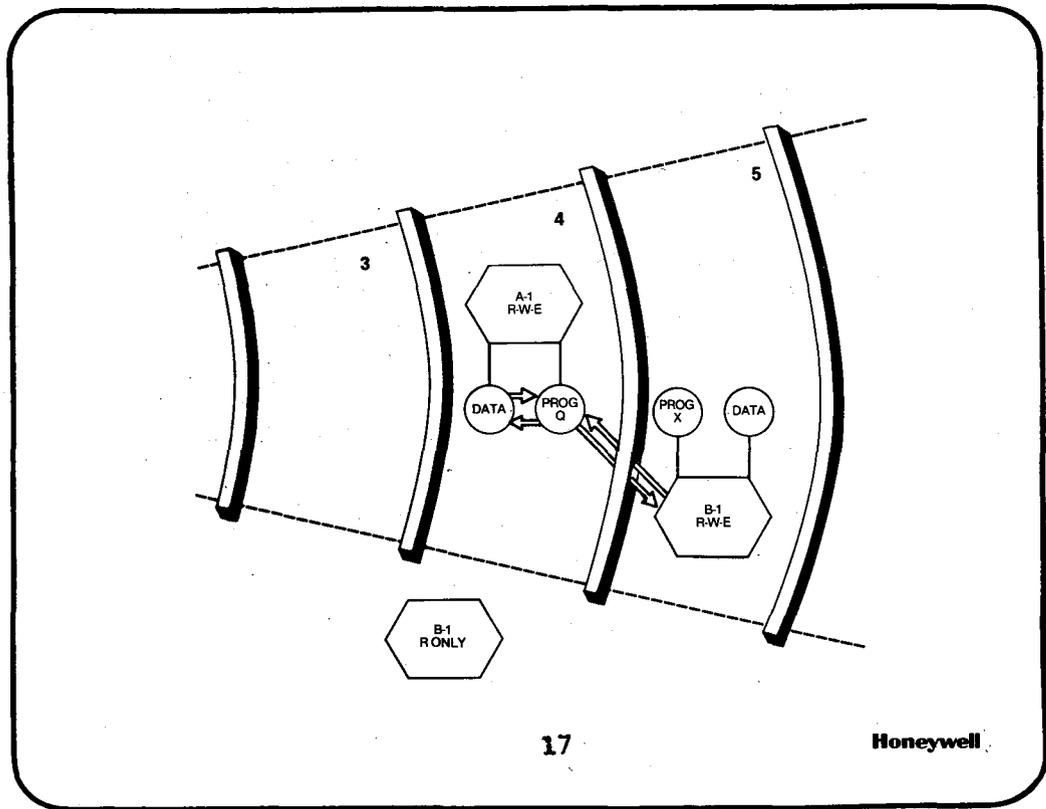
Honeywell



14

Honeywell





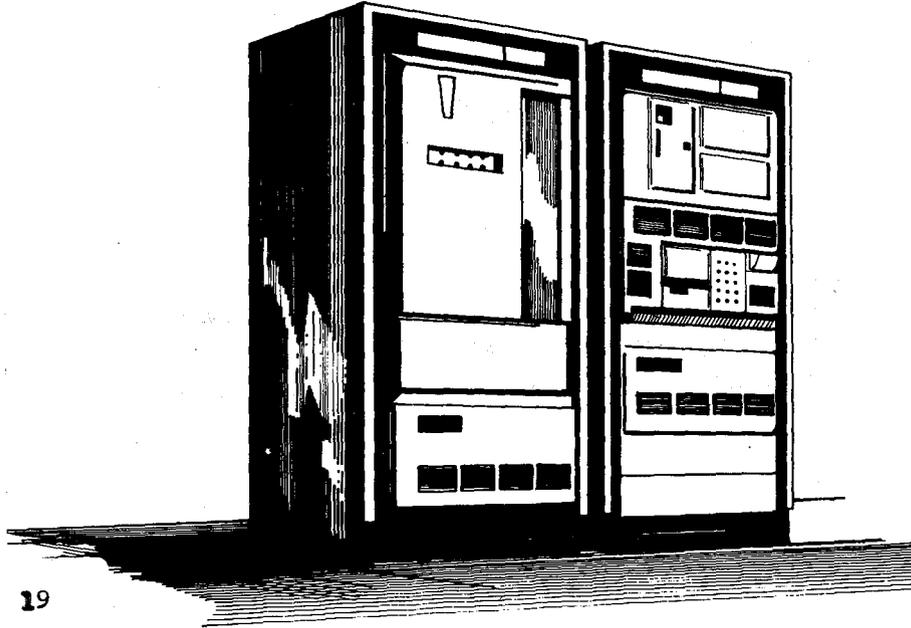
PROJECT GUARDIAN

- ACCESS ISOLATION MECHANISM
- KERNEL FEASIBILITY ESTABLISHED
- PROOF METHODOLOGY DEFINED
- SECURE FRONT-END PROCESSOR DEFINED

Honeywell

Honeywell

TEMPEST LEVEL 6 MINICOMPUTER

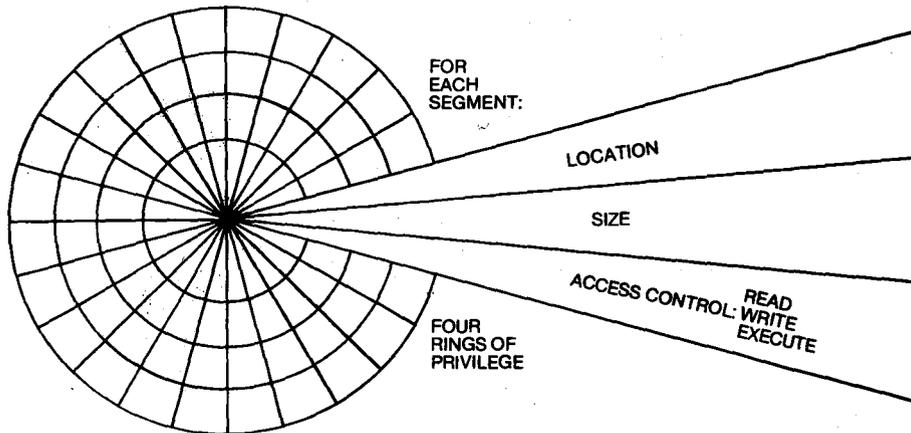


19

Honeywell

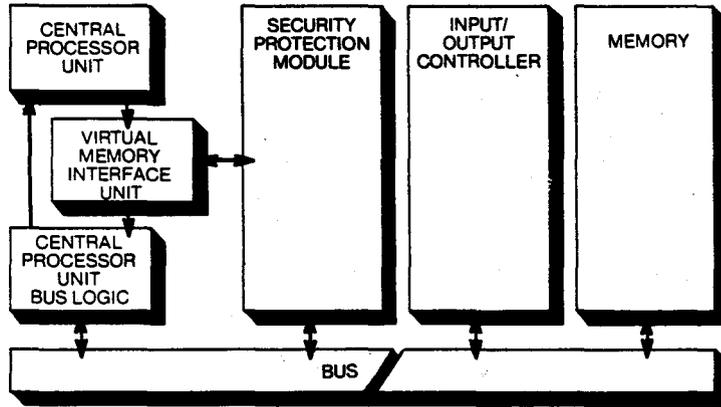
MEMORY MANAGEMENT

31 SEGMENTS



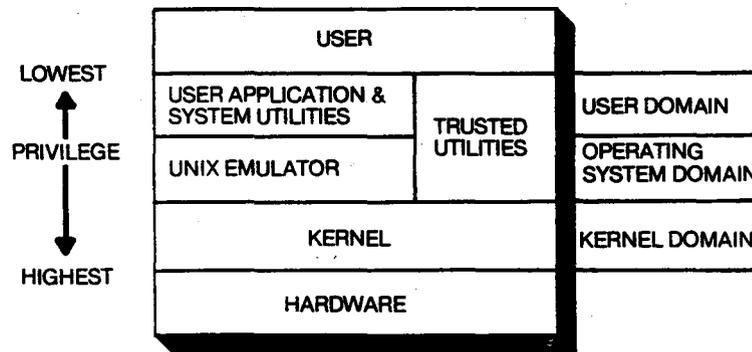
20

SPM + LEVEL 6 MINICOMPUTER = SCOMP



21

SOFTWARE OVERVIEW



22

DPS 8/GCOS 8

AN EVOLVING SYSTEM

DPS 8 + GCOS 8 =
POTENTIALLY PROVABLE TRUSTED ADP SYSTEM

23

DPS 8/GCOS 8

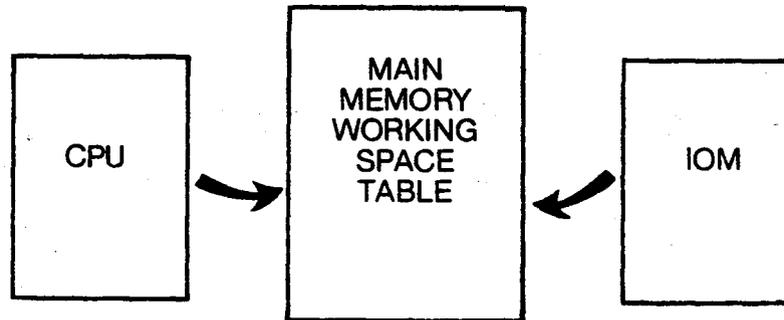
SECURITY

- VIRTUAL MEMORY
- SECURITY MECHANISM EMPHASIZED
IN HARDWARE
- DOMAINS

Honeywell

24

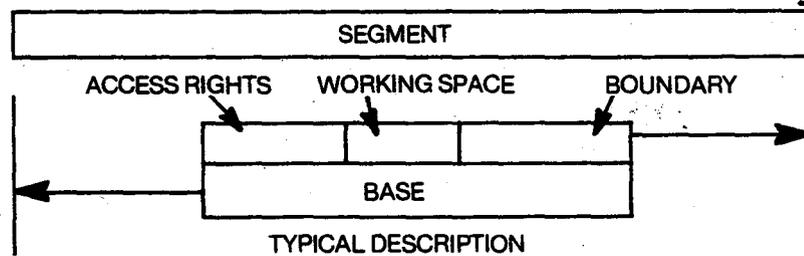
MAIN MEMORY WORKING SPACE TABLE



CPU & IOM USE
SAME VIRTUAL REFERENCES

25

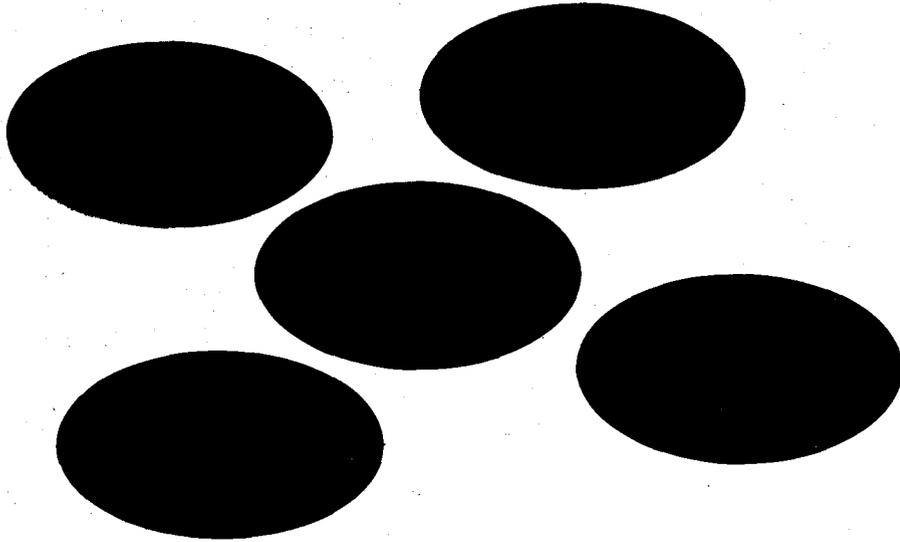
ACCESS CONTROL MECHANISM = COMPLETE MEDIATION



26

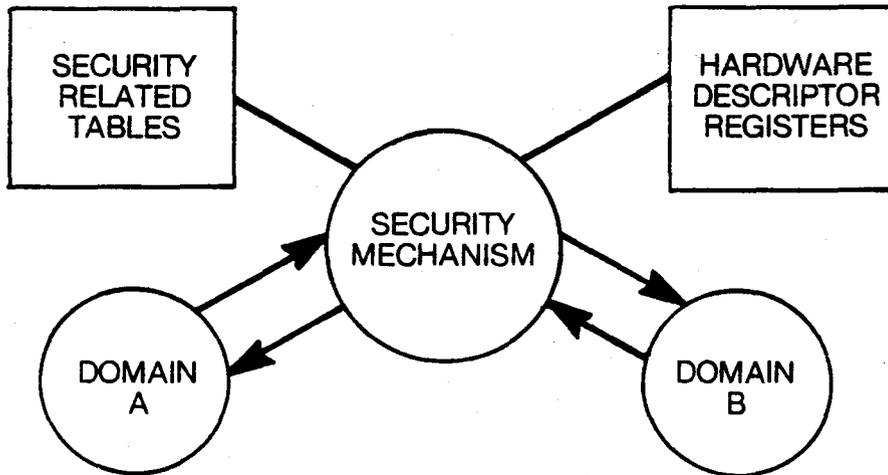
Honeywell

DOMAIN ISOLATION



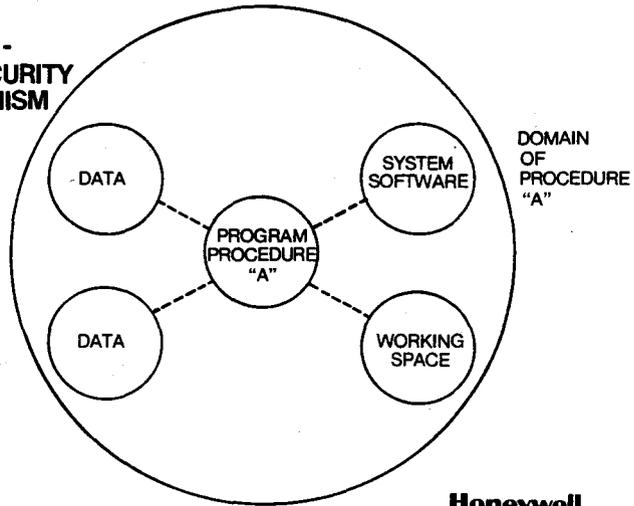
27

Honeywell

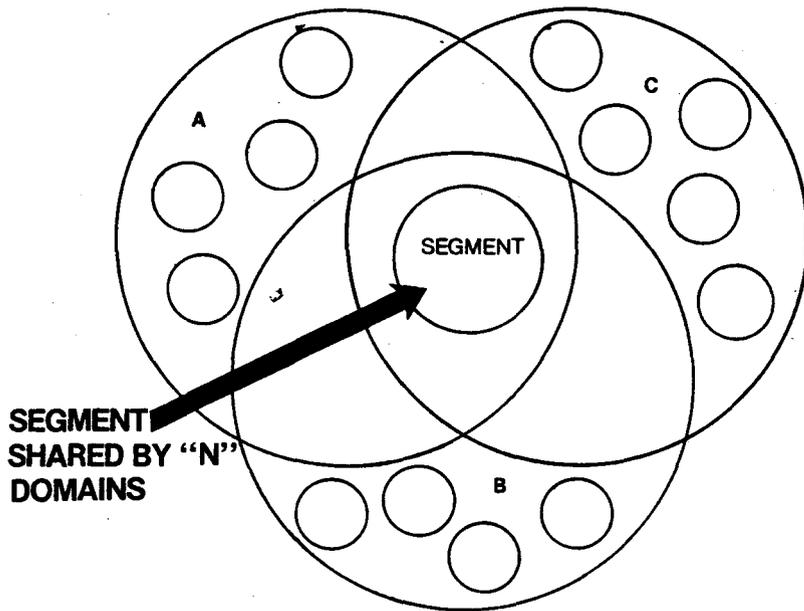


28

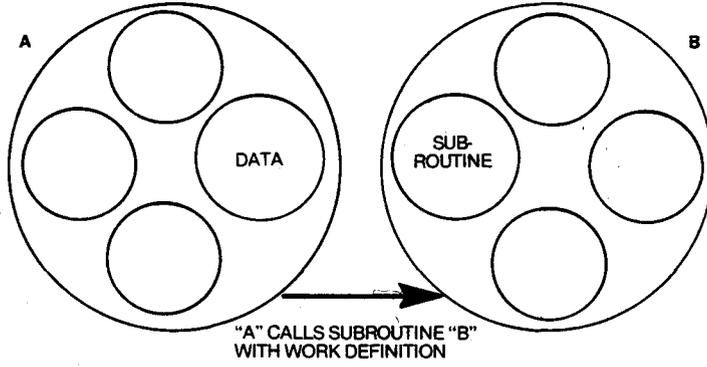
DOMAIN-BASED SECURITY MECHANISM



Honeywell

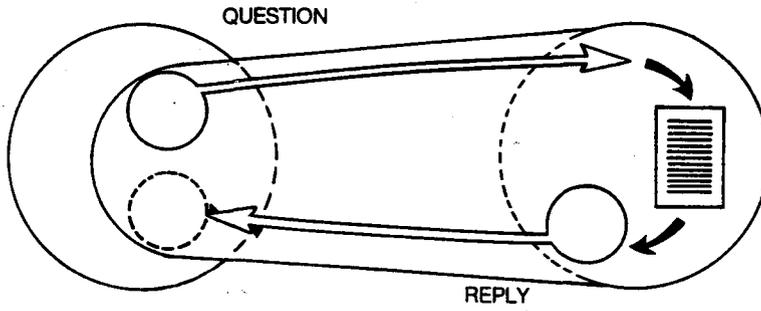


TEMPORARY SHARING BETWEEN DOMAINS

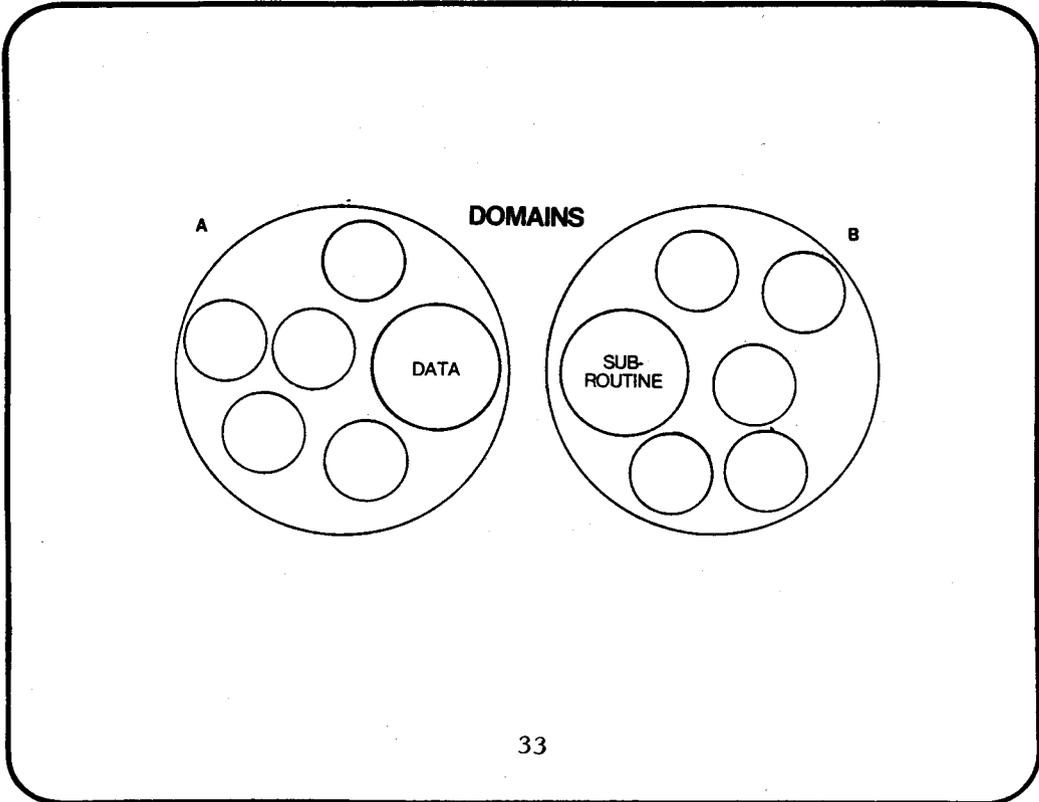


Honeywell

31



32



DPS 8/GCOS 8

THE ARCHITECTURAL KEY TO PRODUCTION ORIENTED
MULTI-LEVEL SECURITY

HONEYWELL GOALS

TO REDUCE THE COST FOR SECURE ADP OPERATIONS

35

HONEYWELL GOALS

TO PROVIDE MISSION OPTIMIZED MULTI-
LEVEL SOLUTIONS

36

HONEYWELL GOALS

TO OPTIMIZE SYSTEM EFFICIENCY IN THE MULTI-
LEVEL TRUSTED ADP SYSTEM ENVIRONMENT

37

HONEYWELL GOALS

TO WORK WITH GOVERNMENT AND INDUSTRY TO
DEFINE AND "HOPEFULLY SOLVE" THE ACCREDITED/
CERTIFIABLE TRUSTED ADP SYSTEM PROBLEM

38

COMPUTER SECURITY RESEARCH AT DIGITAL

Third Seminar on the Department of Defense

Computer Security Initiative

18-20 November 1980

**Paul A. Karger
Digital Equipment Corporation
Corporate Research Group
146 Main Street (ML3-2/E41)
Maynard, MA 01754
(617)493-5131
ARPAnet: KARGER@DEC-MARLBORO**

**COMPUTER SECURITY
RESEARCH
AT**

digital

RESEARCH GOAL

**UNDERSTAND HOW TO BUILD AND
SUPPORT SECURE SYSTEMS FOR
GOVERNMENT AND COMMERCIAL
USERS**

RESEARCH ISSUES

- **EVOLVABLE SECURITY**
- **PRODUCTION QUALITY VERIFICATION TOOLS**
- **NETWORK SECURITY PROTOCOLS**
- **ENCRYPTION**
- **LAYERED PRODUCT SECURITY**

EVOLVABLE SECURITY

- **SECURITY MUST FIT IN WITH EXISTING PRODUCTS**
- **SECURITY ENHANCED SYSTEMS FIRST**
- **THEN VERIFIED SECURITY KERNELS**

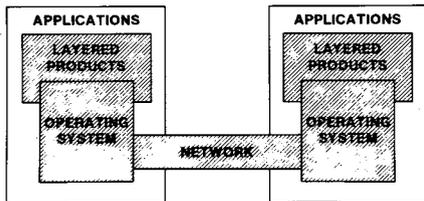
SECURITY ENHANCED SYSTEM

- **FEATURES OF SECURITY KERNEL**
 - **LATTICE MODEL**
 - **ACCESS CONTROL LISTS**
- **NOT VERIFIABLE**
- **BUILT RESEARCH PROTOTYPE ON VAX-11/780**

KERNELIZED SYSTEMS

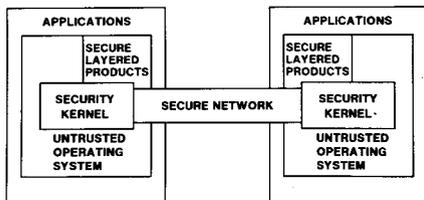
- PERFORMANCE QUESTIONS
- PRODUCTION QUALITY VERIFICATION TOOLS
 - MUST RE-VERIFY FOR NEW RELEASES
- CODE PROOFS IMPORTANT
 - TOP LEVEL SPEC PROOFS DON'T FIND SUBTLE CODING ERRORS

SECURITY ENHANCED SYSTEM



○ SECURITY ENHANCED

SECURITY KERNEL BASED SYSTEM



NETWORK SECURITY PROTOCOLS

- AUTHENTICATION FORWARDING
 - PASSWORD CONTROL
- ROUTING UNDER LINK ENCRYPTION
 - ROUTING NODES ARE HOST COMPUTERS
- NETWORK-WIDE DISCRETIONARY CONTROLS

END-TO-END ENCRYPTION

- ESSENTIAL FOR ETHERNETS
- OUTBOARD FROM OPERATING SYSTEM
 - CANNOT TRUST THE HOST
- WHAT WILL DOD SUPPLY?
 - CRYPTOGRAPHIC DEVICES
 - KEY MANAGEMENT
 - SESSION LEVEL PROTOCOLS

LAYERED PRODUCT SECURITY

- PROTECTED SUBSYSTEM SUPPORT FOR
 - DATA BASE SYSTEMS
 - ELECTRONIC MAIL SYSTEMS
 - TRANSACTION PROCESSING SYSTEMS
 - ETC

WHAT SHOULD GOVERNMENT DO?

- **MAKE CLEAR RFP REQUIREMENTS**
 - ASK FOR BELL & LAPADULA LATTICE MODEL
 - ASK FOR VERIFICATION
 - ASK FOR KERNELIZED SYSTEMS
- **OTHERWISE VENDORS WON'T BE MOTIVATED**
- **INITIALLY SEPARATELY PRICED OPTIONS**

WHAT ABOUT KSOS-11?

- **DIGITAL IS WATCHING KSOS-11
DEVELOPMENT**
- **WE WOULD LIKE TO EVALUATE IT**
- **EXTENSIVE HANDS-ON REVIEW
REQUIRED**

CONCLUSION

- **DIGITAL IS ACTIVE IN SECURITY RESEARCH**
- **SECURITY IS IMPORTANT IN GOVERNMENT
& COMMERCIAL MARKETS**
- **SECURITY WILL EVOLVE IN DIGITAL
PRODUCTS**

digital

CORPORATE RESEARCH GROUP

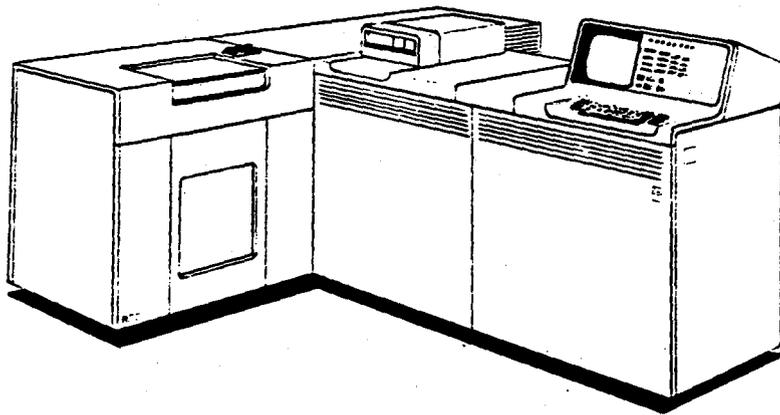
SECURITY AND PROTECTION OF DATA

IN THE IBM SYSTEM/38

VIKTORS BERSTIS

IBM

ROCHESTER, MINNESOTA USA

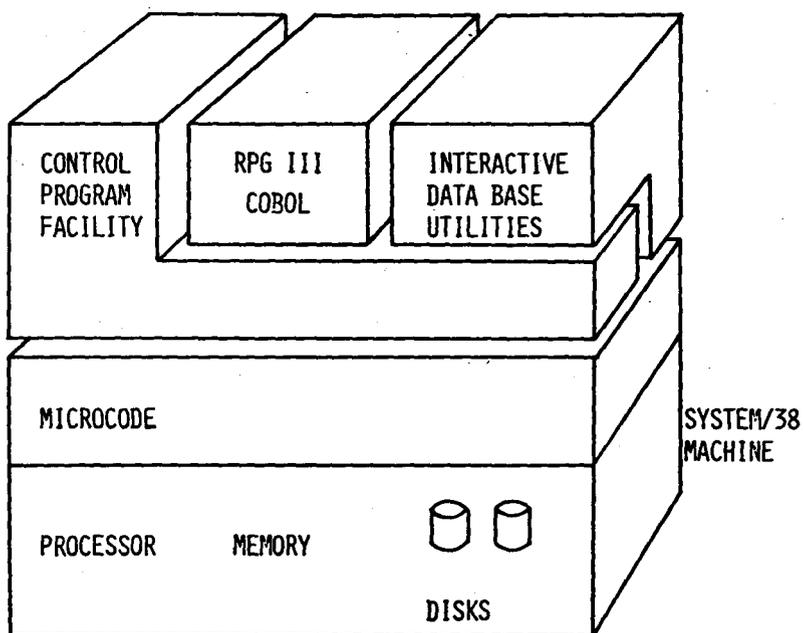


IBM SYSTEM/38

WHAT IS THE IBM SYSTEM/38 ?

- SMALL BUSINESS COMPUTER FROM INFORMATION SYSTEMS DIVISION
- REPLACEMENT AND GROWTH FOR SYSTEM/3 USERS
- AVERAGE CUSTOMER HAS 1-2 PROGRAMMERS
- EASE OF USE PRIMARY GOAL
- LANGUAGES ARE RPG-III, COBOL, CL, QUERY AND DDS
- CONTROL PROGRAMMING FACILITY (CPF)
- HIGH LEVEL MACHINE INTERACE
- DATA BASE FUNCTIONS
- SYSTEM INTEGRITY AND SECURITY

WHAT IS SYSTEM/38?

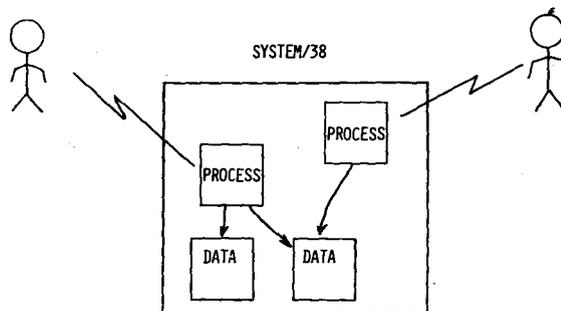


HOW IS WORK DONE ON SYSTEM/38 ?

- USER SIGNS ON TERMINAL WITH PASSWORD
- SUBSYSTEM STARTS PROCESS
- USER PROFILE ASSOCIATED WITH USER
- PROGRAMS CALLED TO DO WORK
- OBJECTS AND PROGRAMS ACCESSED
- MACHINE CHECKS AUTHORITY TO USE

IMPLEMENTATION OF SECURITY IN IBM SYSTEM/38

- CAPABILITY BASED ADDRESSING
- USER PROFILES
- PROCESSES



CONTROL PROGRAMMING FACILITY (CPF)

- COMMANDS
- CONTROL LANGUAGE
- OBJECTS
 - FILES
 - PROGRAMS
 - USER PROFILES
 - MESSAGE QUEUES
 - SUBSYSTEMS
 - JOBS
 - DEVICES
- PROMPTING AND HELP
- SPOOLING
- DEBUGGING
- RECOVERY

SECURITY FEATURES

- MATRIX OF USER PROFILES VS OBJECTS
- AUTHORIZED TO CLASSES OF OPERATIONS
- SECURITY OFFICER
- CPF COMMANDS
 - GRANT OBJECT AUTHORITY
 - REVOKE OBJECT AUTHORITY
 - DISPLAY OBJECT AUTHORITY
 - CHANGE OBJECT OWNER
 - DISPLAY USER PROFILE

 - CREATE USER PROFILE
 - DESTROY USER PROFILE
 - CHANGE USER PROFILE
 - DISPLAY AUTHORIZED USERS
- DISCRETIONARY AUTHORIZATION
- NO MANDATORY POLICY

AUTHORITY CATEGORIES

CATEGORY
AUTHORITY

RESOURCE
 STORAGE ALLOTMENT

PRIVILEGED INSTRUCTIONS
 CREATE USER PROFILE
 INITIATE PROCESS
 TERMINATE MACHINE PROCESSING
 CREATE LOGICAL UNIT DESCRIPTION
 CREATE NETWORK DESCRIPTION
 CREATE CONTROLLER DESCRIPTION
 MODIFY USER PROFILE
 MODIFY RESOURCE MANAGEMENT CONTROL
 DIAGNOSE

SPECIAL AUTHORITIES
 ALL OBJECT
 DUMP
 SUSPEND
 LOAD
 PROCESS CONTROL
 SERVICE
 MODIFY MACHINE ATTRIBUTES

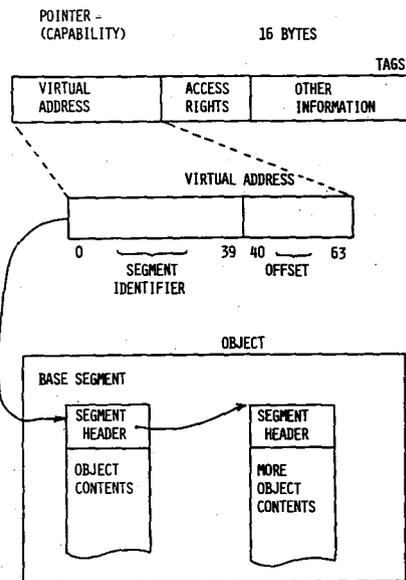
OBJECT AUTHORITIES--AUTHORIZED ON A PER OBJECT BASIS

EXISTENCE:
 OBJECT CONTROL

ACCESS:
 OBJECT MANAGEMENT
 AUTHORIZED POINTER

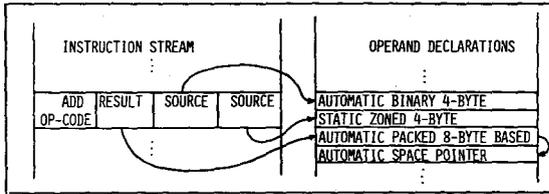
CONTENTS:
 SPACE
 RETRIEVE
 INSERT
 DELETE
 UPDATE

ADDRESSING



PROGRAM CREATION

INPUT TO THE CREATE PROGRAM INSTRUCTION DEFINING THE PROGRAM:

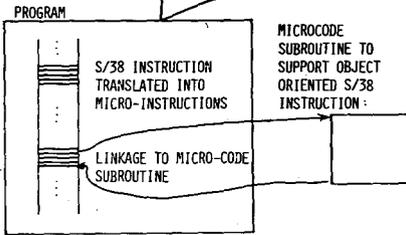


CREATE PROGRAM INSTRUCTION

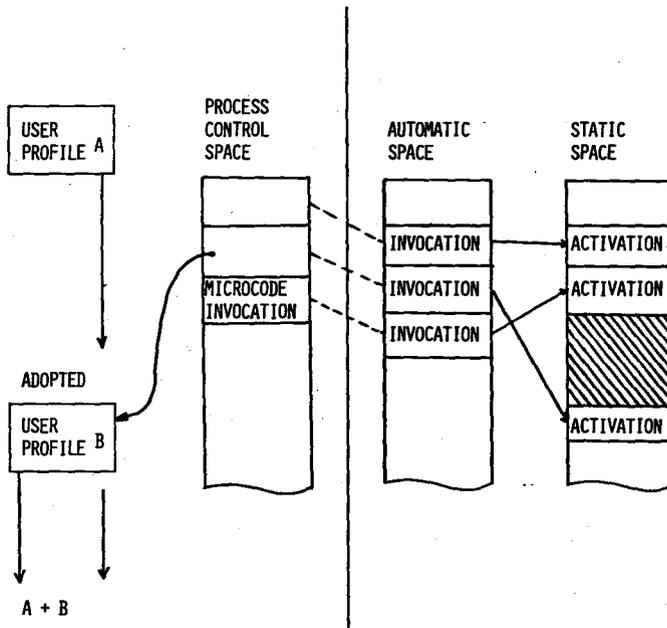
MACHINE INSTRUCTION INTERFACE

VISIBLE IN A SPACE OBJECT

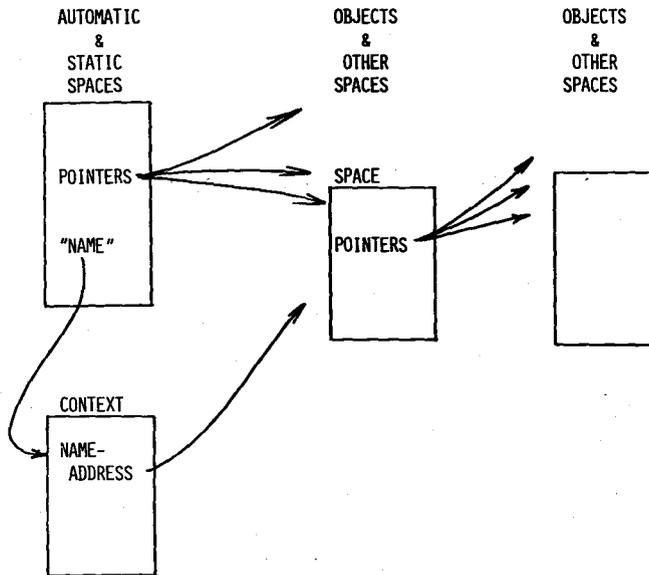
NOT VISIBLE



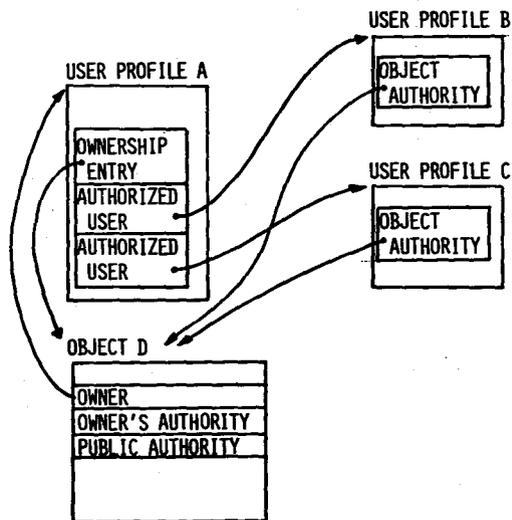
EXECUTION STRUCTURE



ADDRESSABILITY FROM A PROCESS



OBJECT AUTHORIZATION



AUTHORITY CHECKING

- 1 RETRIEVE POINTER (CAPABILITY)
- 2 CHECK FOR AUTHORITY IN POINTER
- 3 CHECK FOR PUBLIC AUTHORITY
- 4 CHECK IF USER IS OWNER
 - IF OWNER, LOOK IN HEADER FOR AUTHORITY
- 5 CHECK FOR "ALL OBJECT" AUTHORITY
- 6 CHECK IF USER IS EXPLICITLY AUTHORIZED
 - LOOK IN USER PROFILE
- 7 OPTIONALLY PUT AUTHORITY IN POINTER FOR NEXT REFERENCE

GOALS OF SECURITY MECHANISM

- CONTROL ACCESS TO DATA
- MINIMUM OVERHEAD
- INTEGRITY/RELIABILITY

GNOSIS:
A SECURE CAPABILITY BASED 370 OPERATING SYSTEM

Presented by Jay Jonekait
Advanced Systems Development, TYMSHARE Inc.

ABSTRACT

Gnosis is a capability based operating system which runs on 370 architecture computers. This paper describes why TYMSHARE developed Gnosis, introduces some basic Gnosis concepts, and shows how they can be applied to application programs. Gnosis appears to be an attractive base for applications run in the high security environments of both DOD and non-DOD portions of the government. Possible alternatives for Gnosis are explored at the conclusion of the paper.

INTRODUCTION

About 1972, Tymshare business planners recognized the need to evolve into new markets in order to sustain profitable growth. One of the emerging trends that was observed was that hardware was becoming cheaper and that the market for selling raw time-sharing was likely to flatten out and perhaps evaporate in the future. At the same time machine cycles were becoming cheap, access to usable information was becoming more and more expensive. Tymshare decided to specialize in the organization and dissemination of the information.

Analyzing the market, Tymshare noticed several obvious business opportunities for general on-line databases. In this market, the ability to protect proprietary data and programs from accidental misuse or theft was a vital prerequisite. One example of this type of business is an online chemical patent database. No customer would ever query the database if they thought the queries might become known to one of their competitors. This kind of security breach would allow the competitor to find out what research they were pursuing.

In essence, what Tymshare wants to do is to develop an information utility, with a large number of on-line databases and a large number of programs that create information from those databases. Most of the programs and databases would not be owned by Tymshare. Protecting the integrity and the security of those programs is a vital concern to both their owners and to Tymshare.

THE DEVELOPMENT FRAMEWORK

In the process of researching how to build such a system and analyzing the options available, it was noted that the existing operating systems running on existing computer systems were not adequate to do the job. The basic problem was that there was no protection mechanism for programs or for data, that there was no way to let two programs that were written by different people interact without having them trust each other. This kind of interaction would expose one or the other to possible theft or misuse. Tymshare had discussions with several manufacturers, and did a lot of research on its own, while trying to envision what would happen over the next 5 or 10 years. We concluded that none of the manufacturers were likely to build the system that would solve our problems.

During the course of its research, Tymshare discovered that there was a rather well-known architecture called "capability based operating systems" which had been prototyped in several universities, such as Hydra from Carnegie Mellon, and CAL TSS from University of California. These systems seemed to offer great promise for being able to solve the kinds of problems Tymshare needed to solve in order to create the businesses that it wanted to create.

So, contrary to the then widely accepted philosophy that it takes a large army or a small hoard to build an operating system, a very low key project was chartered at Tymshare in 1974 to build a commercial quality, capability-based operating system.

There was a precedent for such temerity. The company had taken another similar risk about six years earlier when it went against all then common technology wisdom to produce what is now Tymnet. That

investment was very successful. It was on the strength of that investment and the fact that some of the same people involved with Tymnet have been involved in Gnosis, that the project was approved.

Thus, with a very small group, Tymshare set off to build an operating system. One of the ideas that Tymshare had to face up to was the fact that it was not a hardware vendor and that therefore did not have the luxury of being able to specify the design of the system hardware.

Tymshare deliberately selected 370 hardware, despite the fact that 370 hardware is probably not favored in the security conscious environment. The primary, and single biggest reason for selecting 370 hardware was that there is a wide range of available CPU's that extend from very small to very large configurations. You may have noticed in the last couple of years, that the small 370 CPU's are becoming smaller and the largest ones are becoming larger. The trend appears to be continuing and we expect both 370's on a chip and 15 MIP processors to appear very soon.

The idea of extensibility was of particular interest to Tymshare. We have built many applications on small machines and have been somewhat embarrassed when those applications became successful and suddenly there were more users than we knew what to do with. We couldn't move them to a larger machine because there was no larger machine. We picked the 370 in part because if an application is built on a small machine and the market grows, it is possible to move it to a larger machine. Clearly, it is also convenient to be able to take advantage of Gnosis on the very small mini and micro 370's.

The second critical feature is that the 360/370 hardware has become an implicit industry standard. This architecture is going to have a very long life cycle, and we expect the evolution of the 370 to continue. There will probably not be any revolutionary changes to the 370 which will impact Tymshare's business. Even if there are drastic changes, at this moment, there are a large number of second sources for 370 hardware available. We expect to take advantage of that fact if anything is

announced which precludes Gnosis operation on future IBM mainframes.

Until now, hardware has been emphasized. There is also a strong motivation on the software side for picking 370's. There are literally tens of billions of dollars of software invested in 370 based operating systems right now. There is a wide range of language processors, debugging tools, database management systems and utilities. Having limited resources, Tymshare didn't want to have to write all those programs and wanted to take advantage of the software that other people have written for 370's.

DESIGN GOALS OF GNOSIS

To penetrate the markets described earlier, Tymshare decided to build this system with several design goals in mind. First and foremost it is necessary to be able to protect proprietary programs and data; this involves such things as being able to provide execute only protection, or at least to have the image of execute only programs where the source and object cannot be displayed or tampered with. It also involves the ability to have dynamic databases which cannot in any way be accessed except through the database management system. It involves ultra secure file systems and so forth.

Second, in order to build this information utility type system, Tymshare had to have a very high performance system to do transaction processing. One of the systems analyzed when considering possible operating systems was the Airline Control Program, ACP. ACP meets many of the performance objectives, however, it is very difficult to work with and has almost no security. Thus ACP tends to provide very high performance, non-secure transaction processing applications.

The third requirement is that the information utility business tends to lead to very complex application programs (although complex application programs and complex operating systems are not solely the property of the information utility). However, one of the things observed is that a complex application is very difficult to enhance. It is also a well known fact, that in most installations 80% of all

in-house manpower is utilized doing maintenance and extending existing applications. Thus the problem is that when one changes a program to add new function, often something which used to run is destroyed.

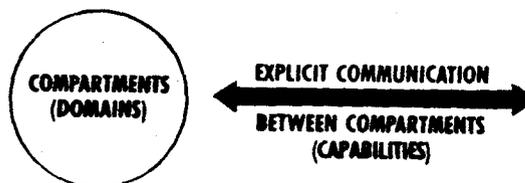
Tymshare needed to have a system in which changes could be introduced in a controlled manner without impacting existing operational software. Gnosis is such a system.

In addition, even when the system was not being changed, it was necessary in the utility environment to have a system which would degrade gracefully instead of crashing around our ears. It is the property of current operating systems and current applications to crash and disintegrate, kind of like an old fashioned string of Christmas tree lights; when one goes out, they all go out. This is not acceptable in Tymshare's environment. So Tymshare designed an environment where very small portions of an application program or a very small portion of the operating system can fail without affecting the rest of the system. Any user who is not particularly involved with that portion of the application or that portion of the operating system will be able to continue to run unimpaired. All of this leads to rather substantial benefits in programmer productivity.

Tymshare was not planning on building a military security type operating system. However, when Tymshare heard about KVM, KSOS, and PSOS, we wondered if we might have developed a system which is suitable in this kind of an environment--not because it was designed for that environment--but because by using proper design techniques it solved the problems of protecting proprietary programs, of simplifying application maintenance, of building a fail soft system with high performance and in the process it also solved most of the security problems that have been grappled with by many people in this room. Tymshare obviously hasn't solved all the problems, but it has solved a great number.

ESSENTIAL GNOSIS CONCEPTS

To clarify how it is possible to make these statements, it may be appropriate to introduce you very briefly to two of the basic ideas of Gnosis. First, Gnosis, like any other capability based operating system, allows you to take a program and break it up into a bunch of small compartments. Second, it provides for explicit communication paths between compartments.



This compartmentalization of both the operating system and of application programs serves the same purpose as compartmentalization of information within other kinds of secure environments. Each component may have access to information only on a need to know basis, and may make changes only where it has the explicit authority to do so.

COMPARISON OF GNOSIS AND 370 SOFTWARE ARCHITECTURE

How is Gnosis different from other operating systems? Again, this architecture is not particularly proprietary to Gnosis but is common to all capability based operating systems. In a standard operating system, there are a bunch of objects called virtual memories or tasks or control regions which contain user programs. Underneath them is a supervisor which keeps users from getting in each other's way, decides which user can do what to whom, schedules resources and generally controls things.

If an application package contained several programs, some mathematical subroutines, an interface to a graphics system, and a data base management system interface, all the code supporting these functions would co-exist in a single virtual memory. Since all the programs share the same memory, there is no assurance that the code in any one of these components will not destroy or alter data belonging to any of the other components.

In this environment, it is possible for any part of the application to access the data buffers of the data base manager (if it can find them). Even though the data base manager carefully cleans up after itself, a security exposure exists if the application program processes interrupts from various external sources. Similarly, a bug in the graphics package can clobber code in the mathematical subroutine package without leaving a clue as to who did it. These complex unintended interactions lead to unreliable operating systems and application programs, frequently with disastrous consequences.

Reliability, integrity, and security can be attained by breaking applications into separate, isolated components which can communicate with each other only through explicit and controlled interfaces. In such a case, the graphics package, for example, could exist in its own virtual memory with its code and data completely protected. If any failure occurred in the graphics package, it would be possible to know with great certainty that that failure was due to a flaw in that graphics package, and those parts of the application that did not depend upon the operation of the graphics package would continue to run.

In Gnosis, every application, and in fact most of the operating system itself, is divided into small, self-contained units called domains. Domains may communicate with selected other domains via explicitly authorized communication paths called capabilities. Domains are created and supervised by a very small kernel of system code. A Gnosis domain serves the same purpose as an address space or a virtual machine in today's systems: it provides a place for the program and its data to exist and to execute. The difference is that a Gnosis application will typically consist of several domains, each containing a small subsystem (typically 50-1000 lines of source code) implementing a specific function.

Each domain will typically hold capabilities which let it communicate with a small number of selected domains. It is not possible for a domain to access its capabilities directly, or to counterfeit the ability to interact with another domain. Thus, a domain may only interact with those domains with

which it has been given a capability to interact, and the interaction may only be of the form represented by the capability. (A domain with a read-only capability to a file may not write into the file.)

The same compartmentalization into domains has been applied to the operating system, so the difference between the operating system and the application in Gnosis is very blurred. In fact, almost everything, except the kernel is in domains. One of the interesting properties which results from this is that there is a not a monolithic operating system. That is, the user does not have to take the whole thing. If you do not like the Gnosis command system, you are perfectly free to build your own command system. If you do not like the Gnosis file system, you are perfectly free to build your own file system and so forth.

In particular, this also means that application code can be replaced selectively. If there is a piece of an application which is not performing properly and you want to replace it, the piece can be safely replaced with another without jeopardizing the remainder of the application.

Gnosis has a kernel which performs some of the tasks normally assigned to the supervisor. The Gnosis kernel is small, about 10,000 lines of code, as opposed to half a million lines on some of the large IBM operating systems. The kernel has been made very small by making it a mechanism whose task is to implement and enforce policy rather than define policy.

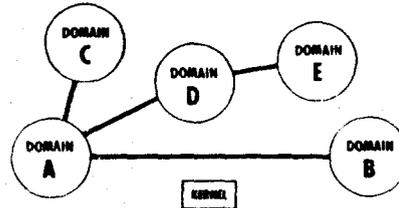
Because the kernel is small, we expect it to be more trustworthy and reliable.

One other fact to be noted, is that the Gnosis kernel has been designed in such a manner that it can be easily put in microcode.

GNOSIS DESIGN OBJECTIVES/INTERDOMAIN COMMUNICATION

Consider the relationship between any two domains (here called A and B) of a Gnosis application package. If the program in domain A breaks, with a 100% probability, the bug is in the program in domain A. There's nothing that domain B can do in any way to impact the internal operation of domain A. This makes debugging much simpler, since faults can be clearly isolated.

GNOSIS application architecture



When domain A calls domain B, information passes using some protocol agreed upon by the authors of A and B. The way one can tell that B is working is by building a test program which exercises all the appropriate parts of the protocol with A and checking to see if B gives the right responses in every case. Outside of that no one need really care what goes on inside of B. We have used this property to great benefit in building the operating system. We spent a lot of time working on the protocol between any two domains. The code that goes inside the domains is often implemented using the simplest possible algorithms.

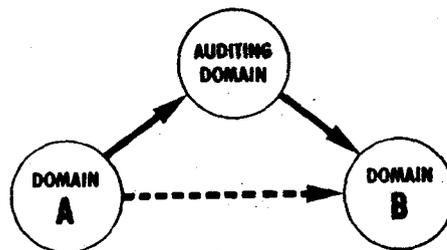
So, for example, it's very easy to build a quick and dirty application domain which implements a protocol between A and B. If one decides some day later to put in a high performance version of B, it's a very simple matter to write a new B to replace the old B. If the new B obeys its protocol, then A and B will continue to work with a 100% probability.

One other thing is important to remember about the connection between A and B. This connection is put there by a person who has the authority to put the connection there. This connection cannot be forged in any way shape or form. There is no password protection, no possibility of A being able to introduce itself to B unless someone who has the proper authority makes the introduction and connects the two domains. So there is a tremendous amount of security involved in the architectural structure of Gnosis.

Let us now discuss the idea of connecting domains together to perform an audit function.

AUDITABILITY

The ability to audit specific transactions is vital in any security conscious environment. Gnosis has extremely powerful facilities to assist this activity. If an auditor wishes to examine the transactions between A and B, (and if the auditor has the authority to do so,) it is possible to take the connection between A and B and splice an auditing domain into that connection. What is vitally important is that A and B will continue to interact without being able to detect the auditor's presence.

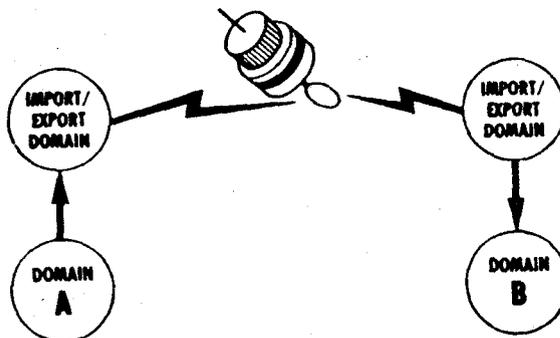


The ability to splice an auditor in between any two domains is a significant property of capability based architecture. It is possible to use this function for other advantages. For example, one can insert debugging routines, performance monitors, or transaction logs.

DISTRIBUTED COMPUTING

One of the more interesting ways in which it is possible to use this technology is to implement distributed computing. It is possible to move B physically to a remote machine without making any changes to the code of either A or B. This is done by inserting two general purpose import-export domains in the same manner the auditor was inserted. An import-export routine is attached to A. B is moved to a remote computing system and attached to another import-export domain. When a telephone or satellite link is established between the import-export domains, A and B may communicate as before. No changes were required in either routine, in fact, it is not possible for either domain to know that B has

been moved to another system.



This technique will make it much easier for Tymshare to develop distributed applications because all the import-export logic and all the remote communication logic have been removed from the application program. Thus, the application programs can be developed on one machine. If the application grows and will not fit on one machine it can be split and the pieces put on additional machines as required.

UNIQUE FEATURES OF GNOSIS

There are no major architectural innovations in Gnosis. The only thing that is unique about Gnosis is the implementation. Gnosis is an instance of a capability based system. Unlike the predecessors built in universities, Gnosis is a commercial quality system. Gnosis is the only instance, that we know of, of the union of a capability based system with 370 architecture, which means union of the capability based system and 370 program compatibility. This allows the use of most IBM compilers, languages and application programs.

In addition, the system has been built not as a research project, but as far as we know, the first production quality capability based system.

Gnosis programs can be written in common languages which provide a great deal of compatibility. Again, the innovations are in the implementation, not in the design.

370 ARCHITECTURAL WEAKNESS AND HOW GNOSIS OVERCOMES THEM.

At the current time, the unique feature of Gnosis is that it combines capability, architecture and 370 architecture. The 370 architecture is much maligned because of its security weaknesses, and with due cause. However, what many analysts have confused is the 370 hardware architecture and the architecture of the software systems that run on 370's.

Let us address several of the architectural weaknesses that are often quoted.

The first perceived weakness is that 370 is a two state machine. (Some other computers have three machine states.) Gnosis extends the limited two state architecture of the 370 by the use of domains for both the operating system and application programs. The result is a system with an unlimited number of distinct states without an implied heirarchy between them.

The second well known problem with 370 architecture is that the I/O architecture is very complex and fraught with security exposures. Gnosis solves the problem by architecturally prohibiting any domain programs from executing any channel programs. The kernel provides I/O services through a very small set of simple channel programs which can be thoroughly debugged.

The third common charge is that 370 system software has massive denial of resource exposures. Gnosis has been architected and implemented in such a way that all denial of resource exposures are closed, assuming the hardware is performing correctly.

SYSTEM STATUS

Briefly, this is where Gnosis is today:

We are scheduled to do a performance benchmark on a real machine by the end of 1980.

The kernel is complete and working. It compiles and runs programs written in any standard 370 language.

However, with limited resources we have not been able to put in all the support functions which one would normally expect in an operating system. For example, Gnosis does not have: 1) a data base management system, 2) full screen display capabilities, or 3) a sophisticated procedure language at this time.

SECURITY EVALUATION STATUS

We have been involved for the last year with the Computer Security Initiative and the evaluation team has come up with a report evaluating Gnosis from a security standpoint.

The same team is now defining a security policy which they will recommend be implemented on Gnosis.

We are also evaluating the need to develop formal specifications for Gnosis.

SECURITY AND OTHER POLICY ISSUES

During the early parts of the evaluation, it was discovered that the Army had a different set of security requirements from the Navy, which in turn had a different set of requirements from the CIA, the NSA, and so forth.

Being confronted by a multiplicity of requirements and few resources, TYMSHARE realized that since no security policy was universally acceptable, it was better to provide universal tools which would enable users to implement their own specific security policies.

One of the advantages of Gnosis is that it can provide an environment in which more than one, in fact in which a number of policies can coexist. Each user must follow the policy established by those in authority who devise the policy for his group.

A PERSPECTIVE ON GNOSIS

Tymshare, during the course of its research, has tried to visualize where Gnosis fits in the spectrum of currently available and proposed operating systems. On the Computer Security Initiative rating

systems, our current implementation will probably achieve a level 3 rating. If we choose to produce formal specifications, it seems possible to achieve a level 4 or 5 rating. Thus, Gnosis fits somewhere between IBM's mainline products and KVM. On a performance spectrum we expect extreme variations depending upon whether the application program can take advantage of Gnosis features. We expect most applications to run within a binary order of magnitude (either faster or slower) on Gnosis compared to IBM's operating systems. Most will run at about the same speed.

SUMMARY OF GNOSIS ADVANTAGES

In summary, we expect Gnosis to provide significant productivity benefits, major enhancements in ease of maintenance for changing applications, high performance, compatibility with existing IBM programs and applications, and a high degree of protection for both programs and data. These advantages may be realized over a wide range of hardware configurations, and will allow Tymshare to develop a number of computer service businesses which cannot be realized today.

POTENTIAL SECURE APPLICATIONS

Tymshare is now at the crossroads--with a limited staff we can help prospective clients develop a trusted environment for selected applications. Essentially, there are four products that readily come to mind that seem to have the highest payoff in terms of meeting a need for which there is no existing product.

The first, and these are not necessarily in order, is to combine Gnosis with a relational database system, to produce a database engine (commonly called a backend or a database machine) which can be used to support multi-level secure databases. In this case we can support relations with multiple security levels.

A second product would be a trusted message switch, using Gnosis as a front-end processor to connect two or three or five or ten machines, none of which trusts any other. It might also be used as a mes-

sage switch to transport messages between different users who should not communicate with each other, except through controlled channels.

The first two examples illustrate that Gnosis is not particularly a replacement for MVS or for any standard operating system, but a tool with which to build almost any kind of trusted high-performance computer system.

If one combines the message switch and the relational database in the same machine, one can build a secure transaction processing system. We have investigated the possibility of using this system to help defense contractors who need to have subcontractors' information collected in some safe place but cannot allow subcontractors to see each other's information.

A network of computers between government agencies who don't wish to share their all secrets can also be envisioned.

The trusted intermediary - An example in the commercial world is the case where a person has written a program which processes seismic oil data and another person has some oil data that he needs to have processed. Neither entity is willing to give up the program or the data and yet the two of them can cooperate with great mutual benefit.

CONCLUSIONS

Tymshare is planning the future of Gnosis. We need more information about where Gnosis is appropriate, and where in government there is a need for Gnosis. We have tried to mention a few potential applications here which come to mind. We would like very much to get more information about whether the applications mentioned are appropriate.

We also are attempting to decide the value of formal specifications. We would very much like to have some information as to whether having formal specifications would make a difference in terms of the potential market for Gnosis. To answer these questions, we need your help.

In order to provide you with more background that has been possible in this brief 30 minute presentation, we have a considerable amount of available literature. Some information is still preliminary, but it describes in more detail the system as it stands, and what we expect to be able to do with it. We have available the report from the evaluation team, which deals with the security-oriented aspects of the system rather than the functionally oriented aspects of the system. Finally, we are willing to engage in considerable technical discussion with those who are interested.

Computer Security Developments at Sperry Univac

Theodore M. P. Lee
Manager, Systems Security
Sperry Univac
Roseville, Minnesota

November 18, 1980

Good morning. You have heard much -- and will be hearing much more -- about a number of efforts at the fore-front of computer security technology research and development. We thought it would be useful to set these efforts in perspective by talking about how the company I work for has dealt with the subject of computer security in the context of very large, mature operating systems and a diverse and well-established customer base.

As you know, Sperry Univac is the computer manufacturer with the second-largest installed customer base in the world. Our share of the federal government market is larger than our share of the over-all market, especially when you include our Defense Systems Division -- which produces the U.S. Navy standard ruggedized ship-board computers.

One would think that with that kind of customer base we would feel strong pressures and recognize a strong incentive to quickly produce a "trusted computer system," as that phrase is understood here. We do perceive a concern and a need, but not ones with much urgency or clarity; the reasons why this is so are mostly what I am going to be talking about.

Before I begin, however, I'd like to make a comment, lest anyone misinterpret my purpose. We believe that we do build trustworthy computer systems. You trusted them when you flew into the airport here, or almost anyplace else; in fact, you most likely trusted them when you asked the airline to hold a seat on the plane for you. Many of you trusted them when you took your paycheck to the bank. If the situation in the MidEast -- or Africa -- or Afghanistan -- or anyplace else -- gets much worse there are many people who are going to trust some of our computers to do what they are supposed to do in that eventuality. There are also many people who are trusting our machines to help them know if things are getting worse. And these people really do know and care about computer security, even if they don't talk to anyone much about anything. So, in a way, by replacing the word "secure" in discussions like this by the word "trustworthy" -- so as not to give the false impression that the computers in the U.S. government's inventory are insecure -- my friend Steve may be making a different set of people upset with him.

So with that off my mind, what am I going to talk about?

First, I'm going to tell you a little about Sperry Univac and what it makes.

Then I am going to tell you about what we have done over the last ten years or so in the name of computer security -- or that has been done to us.

Finally, I will tell you what we have going on now and in the near future that I think does show progress towards more trustworthy computer systems.

What is Sperry Univac?

Sperry Univac is the major revenue and profit-generating part of the Sperry Corporation (until recently known as the Sperry Rand Corporation.) It was in effect started by the U.S. Government shortly after the second World War and has a fascinating history -- much of which, as they say, remains to be told. Its early progenitors -- Eckert-Mauchly and Electronic Research Associates -- produced the first modern commercial computers; (I'll let the courts argue over exactly how to word that and exactly what it means)

We have six major product development centers -- each of which is responsible for a different -- but coordinated -- set of products, a number of manufacturing locations, and scores of sales and customer support offices all around the world. (About half of our business is outside the United States.)

The major product lines, then, are:

In Blue Bell, Pa. -- company headquarters -- we make our series 90 and System 80 lines of small and medium-scale byte-oriented computers with an architecture similar to the IBM 360/370-style architecture, supported by our own software.

In Salt Lake City we produce communications processors and terminals -- smart and dumb -- used on all the mainframes.

In Irvine, Cal. our Mini-Computer Operations -- acquired from Varian Data Machines a few years ago -- supplies the V77 line of mini-computers, which are sold both on their own or as OEM products, to ourselves and to others.

In Cupertino, Cal., ISS makes disk-storage devices.

In the Minneapolis-St. Paul area there are two other major divisions.

The Defense Systems Division produces ruggedized and other special-purpose systems, mostly for the U.S. Defense Department, mostly for the U.S. Navy. But it is out of there that the air traffic control computers used at most of the major U.S. airports come.

And finally, in Roseville, Mn -- a suburb of St. Paul -- we make the large-scale 1100 series family of computers. The currently produced products in that family range in size and cost from the 1100/60 -- selling for about \$500,000, running at about 600,000 instructions-per-second -- to the large-scale 1100/84 -- about \$10,000,000 at about 8 million instructions-per-second. Previous products in that family trace back to the ERA 1101, although the first machines with truly similar architectures began with the 1107 and 1108 in about 1962.

The operating system for the 1108 -- called Exec 8 -- was the first modern multi-processing operating system that had a full-service file system, full suite of utilities and compilers, and supported multi-programming and interactive time-sharing. We take pride -- and incur much technical challenge -- in the fact that even though the hardware has been continually enhanced over the years, the current version of the operating system still supports -- from a single source tape of the system -- all previous versions of the hardware since the 1108. And this includes the fact that we have added more base registers, added new instructions, and changed I/O and error-reporting interfaces with almost every new model of the hardware.

The complete set of systems software for OS/1100 contains about ten million instructions, of which maybe 500,000 are the executive itself, a couple of million lines are in compilers, and the rest are the data management system, transaction processing system, and utilities. It has been estimated that the core of the operating system -- what would form a Trusted Computing Base -- could be pared down to about thirty-two thousand instructions.

I know the foregoing sounds like a sales talk, but it is very relevant: we have much history behind us and cannot start from scratch. (I'll have more to say on that shortly, because we did try -- twice, in fact -- to start again from scratch.)

History of Computer Security at Sperry Univac

Although it can be claimed that Sperry Univac's history of computer security activities stretches back to the beginning -- we had the first equipment approved under TEMPEST criteria before it was even called TEMPEST -- serious attention was really given to the problem at the start of the Exec 8 operating system first delivered in about 1967. Just to make a multi-user, multi-processing, interactive system work reliably we had to have protection features in it -- features that we thought were quite effective for their intended purpose.

It has taken us just as long as our customers and the other vendors to recognize that the picture wasn't as comforting as it seemed.

The history of our loss of innocence parallels that of everyone else. It probably started with our attempt to bid an 1100 series system on the WWMCCS program. We did bid and were technically responsive. We did meet the half-formed "security requirements" of the RFP through major special additions to the standard software. Partly as a result of this WWMCCS experience, but also following close on the issuance of DoDR 5200.28, our federal government marketing organization put together a task force to make recommendations on what we should be doing about computer security. Other members of the task force came from both our domestic and international marketing groups, and from product development. Customer representatives were invited to present their needs and thoughts. Perhaps coincidentally, a subcommittee of our user's organization was formed at about the same time to make computer security recommendations: the report of the marketing task force mostly echoed and endorsed the user's report.

Both reports were issued in March of 1973. Notice that DoD 5200.28 had just been issued in January, the Ware report was still classified, and the Anderson report had not yet been widely read.

The report of our user's group is interesting, for its history tells much about the education and communication problems in this field. The committee writing the report was chaired by the head of the University of Maryland's computer center and the other members came from NSA, the Navy, the National Bureau of Standards, and RCA. Neither of the two reports said anything about assurance -- as we now understand that subject -- or much about security labelling of output media. The user's report said nothing about special access categories or compartments or about need-to-know lists. The marketing report strongly felt it was impossible to fix on a single form of security policy -- such as the DoD policy -- for all customers and instead asked for a quite general, almost programmable, means to specify the security "authority" of a user and the security "requirements" to be met for accessing a particular file.

It took us back in Roseville a number years to draft our response to the marketing report -- for it contained numerous detailed recommended changes that needed to be coordinated with our other development plans and commitments -- and we are now just about to ship the first pieces of code implemented in response to that process.

During this long period we have had until recently very few additional demands from our customers. In 1973 NRL commissioned a small penetration study of a particular widely-used but already obsolete version of Exec 8. They documented one already-known small class of vulnerabilities -- not applicable to later versions of the exec -- and despite the fact that -- and probably partly because of it -- the report of the study was classified for about six months its not-very-favorable conclusions made the

national press, starting with Jack Anderson's column, and even resulted in congressional and DoD-wide investigations. I understand there may have been a few other risk assessments and penetration studies of our systems, but we are generally not told of their happening or of their results.

About the only other "demands" have been in the form of the "security requirements" of various requests-for-proposals. I want to give you several examples, all within the last year. For the most part, these have not clarified customer requirements.

A very large procurement from the Air Force said that the system "must provide the capability to process personal information under the ... Privacy Act of 1974 [and] to process defense classified information ..." without giving much of any criteria for what that meant. It said that "An access control mechanism which denies unauthorized access and allows authorized users to selectively share data files without violating established access authorizations ... must be provided" without saying what constitutes an authorized access. The initial version of the RFP asked that user identifiers and passwords be up to 10 characters long and be system-generated, but a later re-issue of the RFP deleted those requirements.

A Navy RFP specified that the system shall include "functions to establish relationships between password/identifiers and any data base or file." Nothing about what that relationship should be; nothing about security assurance. Another Navy RFP specified that "It is desired that the system provide multi-level security operations; i.e., it shall be possible -- under NSA regulations -- to process unclassified and classified jobs concurrently." Not providing that would entail a penalty of \$1,000,000 in the first month of the life cycle cost estimate of the system. I don't know of any regulations even being contemplated by NSA regarding Navy multi-level security.

Our commercial customers naturally seem to be even less demanding than our government ones. This includes, for instance, financial institutions, service bureaus, manufacturing industries, or airlines. The major requirements we do see here derive from the various privacy acts of the countries we do business in, and these are met with slight modifications to existing software.

Now, to summarize what I've just said: as far as I know -- and I've done some careful checking -- we have not lost a procurement -- or even declined to bid on one -- because our systems could not meet the customer's computer security requirements.

Other Computer Security Developments at Sperry Univac

In addition to this main thread of the security developments concerning the series 1100 systems there have been several other activities throughout Univac related to security. In a sense,

these parallel my career through the company, but I do not want to take credit for them.

I started in this computer security business back in about 1972 while I was in our Defense Systems Division. At that time my main technical expertise was in interactive computing, especially graphics. For some reason I was visiting in the Boston area and wanted to stop by AF ESD to see what the latest in computer graphics was; my contact said, "We aren't doing much in computer graphics anymore, but we have this guy who is really gung-ho to talk to computer manufacturers about computer security." That guy was Roger Schell.

Not long after, we started a small project on company IR&D funds to learn about computer security. We ran into two problems -- we never made enough progress that we could interest someone like ARPA or NSA in giving us real money, and the Navy still seemed (to us) to be of the view that computers on ships were isolated out in the middle of the ocean and had no security problems.

Anyway, in mid-1973 I was drafted by headquarters to move from the Defense division to our commercial division in Roseville to work on a project that was developing a completely new product line. The goals were ambitious, but there was excellent management support. Amongst many other things, the system was to have all the security architecture features anyone would want -- descriptors, virtual-memory, stacks, domain-protection, programmed entirely in a modern high-level language. We managed to get many people to understand what a security kernel was. We hired Jim Anderson as a consultant -- a process that required approval by the President of Univac. But we had to deal with a fundamental fact of life -- the new system would not be compatible with the existing well-established series 1100 or 90 machines, although we did intend for it to support multiple virtual machines, some of which would emulate the old modes. We did know when we began the project that one over-riding constraint on it was that of preserving our customers' software investment. Ultimately, we could find no convincing way to overcome that hurdle on a radically innovative hardware architecture and the project was cancelled after over five-years of work.

It was shortly after the cancellation of this project -- and partly as a consequence of what we learned during it -- that our management recognized we did indeed need to better focus the attention paid to computer security issues. It was at this time that I was appointed to my current position with the responsibility to over-see all computer security activities.

The same recognition that the best way to move forward would be to have a new architecture surfaced in our newly-acquired mini-computer operations a year or two later. In some ways, that effort made even more progress: it had as a stated goal the need to support DoD multi-level security (in the full meaning of that), had in fact programmed a rough-cut at the security kernel,

and was starting to inquire about obtaining formal specification and verification tools or services from outside suppliers. Things were going well enough that we took DoD up on its offer to look over our shoulder in an informal security evaluation. Unfortunately, much the same fate overtook this project: the need for preservation of the existing customer base, experience, and software led to its cancellation.

Future Developments

Both of the cancelled projects I've just mentioned were not wasted investments. We learned a lot -- not just about security -- and the results of that learning are being directly applied to several future products of a more evolutionary, rather than revolutionary, nature. Without giving away any company secrets, let me tell you some about them.

We are making a number of changes to the series 1100 operating system and the hardware architecture with security specifically in mind, although we are doing these things for many other reasons as well.

First, we will be enhancing the hardware -- in an upward-compatible way -- to add what some of you would understand as a segmented capability addressing structure, with a domain protection scheme. This will give finer control over accessibility, allow the more flexible creation of protected subsystems, and regularize interfaces so that state-switching can be made faster through specific hardware assists. There will also be a virtual machine facility that at least gives us the option of doing a KVM kind of system.

Secondly, we are restructuring the operating system. Although it already attempts to have as much code outside of privileged mode as possible, much more will be broken out and placed into separate domains that have only exactly as much privilege and accessibility as required. We are using more rigorous (but not yet mathematically formal) specification and configuration management tools.

We are also well-along in creating a massive computer-based model of the existing software to document its internal and external interfaces and data structures. This includes not only the executive itself but also the data management system, utilities, compilers, etc.

A second development is taking place in our communications processors. The hardware has been modified to explicitly recognize the kind of job it is doing -- i.e., it has data structures specifically designed to take care of messages and queues of messages. In particular, coupled in an unaccessible way with a message are address descriptors that govern exactly what kind of

access any code processing a given message needs to have; this includes the micro-processors that are attached to each communications line. The hardware is now designed so that the software can be structured into many small procedures, each of which can only access small parts of memory and can only call specific other procedures. The planning people in Salt Lake City are setting their security goals for the software that will use that hardware; the requirements contain strong words about policy, mechanism, and assurance that were directly influenced by the kinds of things being talked about at these seminars.

Our just announced system-80 machines already have a more useful architecture for protection than that of their ancestors and future improvements are well underway.

Concluding Remarks

To summarize, Sperry Univac is a large company, with diverse interests, customers, and products. I hope I have been able to give you an accurate and instructive picture of how we perceive the computer security problem and are responding to it.

We are closely following all the research activities discussed at these seminars, but can't yet commit ourselves to their applicability. This is a very expensive business to make experiments in -- a small kernelized secure text-editor, filing system, and desk calculator can in no way be viewed as a pilot-plant for a large centralized corporate database system.

I thank you for this opportunity to share my thoughts on the subject. Notice that we all will have a second chance this afternoon to raise some of these questions in even more detail.

How Can the Government and the Computer Industry
Solve the Computer Security Problem?

A Panel Discussion

Ted Lee, Sperry Univac
Jim Anderson, Consultant
Steve Lipner, Mitre
Marvin Schaefer, SDC
Bill Eisner, CIA

[At the Second Seminar on the DoD Computer Security Initiative Program, January 15-17, 1980, Ted Lee — attempting to speak for the computer industry — and Jim Anderson — attempting to speak for the government — presented a "dialogue" on the subject of "What every vendor always wanted to know about government computer users' security needs (but was afraid to ask)" There was considerable audience interest in the dialogue, but little time for audience participation. In fact, the interest was so strong that we have invited them back again to pursue the issues in more detail, with more time for audience participation, and we have put three additional people on the panel to ensure that all viewpoints are heard.]

[At the last seminar Lee and Anderson were guided by a list of questions and answers that had been prepared in advance — the questions obtained through an informal canvassing of several vendors, the answers written by Anderson. For this seminar, the major points of those questions have been reduced to ten questions, which are printed below. The answers will come from the panel.]

[All participants are speaking as individuals out of their own experience and do not necessarily represent the views of their respective organizations.]

1. We are generally talking about the data security needs and desires of "the government computer user." Is it meaningful to undertake such a discussion — i.e., is there a "typical government computer user"? Does he care about computer security? How does a vendor discern the computer security needs of that user? Are those needs unambiguously documented in accessible forms, consistent throughout the government? And does responding to them REALLY make a difference (now or ever)?

2. What kinds of applications for computers — e.g., communications, transaction processing, data management, process control, general user-programmable data processing — and what kinds of configurations — e.g., networks, centralized, distributed — are going to have the most severe computer security requirements? Which are of lesser importance? And what portion of the total usage of computers does each represent?

3. In various forms and in various places, such as in DODR 5200.28, AR 380-380 or NBS Special Pub 500-57, attempts have been made to categorize computer systems into a small number of classes of increasing sensitivity based on factors like the amount and mix of classified or other sensitive information involved, how benign the physical and personnel environment is, and what kinds of interaction with the system are allowed. Without arguing about the details of any particular categorization scheme, what mixes of data sensitivity, user trustworthiness, and application environment is it going to be important or highly desirable to support? (e.g., is it meaningful and important to think about handling Top Secret information on a system with some people having only Confidential clearances programming in assembly language?)

4. In the first question we asked generally about whether the "typical government computer user" knew and could express what he needed or desired in the way of computer security. Specifically then, what kind of security policy DOES that user want his computer system to support — i.e., what rules should it enforce? What information is to be used in enforcing the rules? How is the system to interface with the manual world (e.g., marking of output)? And what kind of auditing procedures are to be supported? How fine a granularity (e.g., file, record, field within record) are the rules and other measures to be applied to?

5. How badly does he care that the policy discussed above be applied? What is the perceived importance of the possible threats to it? (e.g., external physical attack, active or passive wiretapping, human error or culpability, malicious legitimate user — cleared or not — attempting technical subversion of the operating system, collusion through Trojan Horses and covert channels, or trap-doors planted at the vendors hardware or software factory?)

6. We are all generally aware of the efforts being made to establish some form of government bureaucratic apparatus for certifying the trustworthiness of computer systems. Will this really happen? When? Where will it be? How will it operate? Will the criteria it applies look much like the draft criteria that now exist? Will it truly be able to make a more standard approach to computer security possible throughout the government? What effect will it really have on future procurements — both inside and outside the government? (And, how reliable are the answers to those questions?)

7. Some aspects of the technology and the certification criteria being developed imply radical changes in the way vendors develop their systems and how they interact with at least their government customers. To what extent is the government going to need closer scrutiny of a vendor's internal development operations? Will it be able to do so in an impartial way and without directly or indirectly — for instance, by the way it words a procurement — revealing proprietary information of one vendor to another? What aspects and physical copies of a highly trustworthy computer system are going to need to be treated as classified? Who will have the responsibility for maintaining the security kernel software? What new export control restrictions will apply to this new technology?

8. A significant amount of new software technology is involved in the current government-fostered development of "secure computer systems." Of the various options being currently explored — security kernels on more-or-less conventional architectures, capability architectures, encryption as a substitute for other forms of security, different specification, verification, and implementation tools and languages — will any particular ones emerge as "best" (either through natural selection or through government fiat)? Will computer security technology ever be good enough that less attention needs to be given to other forms of security?

9. Are the current R&D efforts credible? — they ignore hardware and micro-code problems, appear to have grossly unacceptable performance penalties, and are perceived to have been done on only limited purpose or "toy" systems. What about enforcement of "need-to-know" principles and other rules in addition to the over-simplified partitioning of the world into a few security levels and compartments?

10. What is the economic impact of all these computer security developments — i.e., how much are users willing to "pay" for security (including incompatibility, overhead)? Does it make sense for a vendor to attempt to offer security as a (possibly high-priced) option? When will strong requests for security show up in RFP's? What kind of market forecast could one make — i.e., \$ value of systems to be bought in each of the years 1980-1995 at each of the levels 0-5 of the Mitre TCB evaluation criteria?

OPENING STATEMENT

COMPUTER SECURITY

(S. B. LIPNER)

In late 1970--just about ten years ago--I returned from a field assignment and was asked by MITRE to look at the computer security problem. At the time we were looking at needs for a multilevel secure time-sharing system and a multilevel secure command system--both at unclassified through secret levels. Neither system has yet gone operational as required, though in the intervening years we did achieve some significant things. As far as I'm concerned three of the most significant (in no special order) were:

- (1) The development of the Bell-LaPadula (star-property) model and a set of formal techniques for proving that system security complies with the model;
- (2) The development of a Multics time-sharing system that embodies the *-property (but is not proven) and is in multilevel use today (though all users have some level of clearance); and
- (3) The development of a prototype security kernel for the PDP-11/45 that was subject to limited proofs of compliance with the *-property and demonstrated in simulated multilevel applications.

In the early seventies if we talked to industry about security, the responses we got were "if you just tell us your requirements, we'll meet them". I think those responses were oversimplified. If the requirements are the star-property and proofs nobody in industry is enthused about meeting them. And I'm not sure whether they should be or not.

I do think a lot can be done to make systems better for many requirements. The Multics effort--adding the star-property, plugging the holes, and limiting the risk--is a neat example. I'm not sure that industry is really seizing on that example and emulating it to give customers more choices. I'm also not sure that the government is emphasizing the utility of such systems.

I also worry about security kernels. The original kernel idea (from the Anderson Report) was to have a mechanism that was always invoked, tamperproof and small enough to be subject to complete analysis and tests. Our prototype for the PDP-11/45 and Jerry Popek's were about 1000 lines of HOL each. KSOS-11 is around 10,000 lines. Some of that growth is for efficiency and real-world features. Some is the introduction of neat advanced operating

system concepts that may not be necessary for a small simple secure kernel. I wonder if our desire to do things in the neatest, most advanced way has compromised our ability to adhere to the original Anderson Report principles. I read Lee Schiller's kernel (cover to cover) one night in a hotel room. A proof has to be awfully good to be as convincing as reading and comprehending the entire kernel.

Since leaving the security business in 1976, I've been working on acquisition of fielded systems for the Air Force. Security has raised its head a few times and I've thought of the option of building a kernel for the job. I've always avoided that option in favor of the best off-the-shelf approach available--even if that approach was less secure than I'd like or operationally painful. The cost and schedule risk of building a kernel for a real fielded system has just been too great. But I've been dissatisfied both with what I've had to do and with the quality of the options available to me. If there were more products comparable to the Multics system I mentioned above in level of security (not in specific features) I'd have been much happier with my options and results. This represents a reversal from positions I took in 1973-75--but a realistic one. And if there were off-the-shelf usable kernels that, of course, would be great. The important point is that off-the-shelf options will get used while development gets avoided.

I'd like to think that some synthesis would occur merging the advanced security ideas with the needs of the broader market and the realities faced by industry. Everybody can compromise some and still get significant improvements in capability and security. The important thing is off-the-shelf capabilities available to a user. I hope these conferences are a step toward dialogue, compromise and the delivery of more real systems.

Quality Assurance and Evaluation Criteria

Grace H. Nibaldi
MITRE Corporation

Problem

**How does One Build Quality Trusted Software in
the Face of:**

Large, complex operating systems
High-integrity applications
Easily penetrable computer systems

Solution

**Integrated Software Engineering Approach
Incorporating:**

Policy
Mechanism
Assurance

Policy

Security
Integrity
Denial of Service

Mechanism - Trusted Computing Base

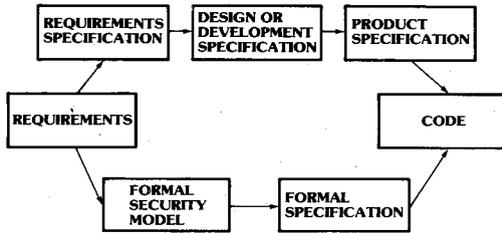
Complete
Isolated
Verifiable

Assurance

Software Life-Cycle
System requirements
Design
Code and debug
Testing
Operations and maintenance

Assurance

Software Development Approaches



Evaluation Criteria for Trusted Systems

- 0 No protection
- 1 Limited access control
- 2 Extensive mandatory security
- 3 Structured protection mechanism
- 4 Design verification
- 5 Code verification
- 6 Hardware specifications

Policy

- 1 Access Controls
- 2 Discretionary Security
- 2 Mandatory Security
- 2 Denial of Service

Mechanism

Prevention	Collusion	Recovery — Hardware
Data protection	5 Timing & storage channels	Software Fault Det.
1 Access control	Detection	1 Diagnostics
System integrity	Audit logging	2 Subverter program
1 Isolated OS	2 Violations	H/W fault tolerance
1 User per process	2 Classified output	4 Limited operation
3 Isolated protection mechanism	2 Time of use	H/W fault recovery
3 Complete mediation	2 Logins	5 Backup systems
Authentication	4 Leakage channels	6 Self-diagnosis & correction
1 Login (user)	5 Real-time surveillance tools	Operations/Maintenance
2 Special character		1 Backup/recovery
Denial of Service		2 Output labeling
2 Time-slicing		4 Configuration management
2 Masquerading		4 Reverification
5 Space quotas		

Assurance

Design	Testing	Verification
Methodology	Production testing	Design to model proof
1 Good engng practice	1 Debugging	4 Flow analysis
3 Structured methodology	1 Functional testing	4 Invariants
3 Top-down design	3 Based on TLS	5 Code to design proofs
Specifications	Test case generation	6 Object code to source proofs
3 Top-level design	4 From TLS	6 H/W spec analyzed against TLS
4 Formal TLS	5 From low level specs	
5 Low level specs	6 From H/W specs	
6 H/W specifications	Penetration analysis	
Implementation	2 Penetration & patch	
1 Inspections	5 Timing channels	
2 Modern programming techniques		
3 Structured programming		
5 Verifiable implementation		

Assurance

Design

Implementation

Testing

Verification

Assurance

Design

- Methodology
- Formal specifications

Assurance

Implementation

- Methodology
- Verifiable implementation

Assurance

Testing

- Production testing
- Test case generation
- Penetration analysis

Assurance

Verification

- Design to model
- Code to design
- Object code to source code
- Hardware

To Come

- Specification & Verification Overview**
- Specification & Verification Researchers**
- Software Testing Overview**
- Trusted System Developers**

Goals of This Seminar

- Terminology**
- Role of Verification in Security**
- Trusted System Acquisition**

Specification and Verification Overview

William F. Wilson
MITRE Corporation

Questions

What is formal verification?

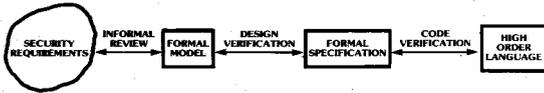
What properties can be proved about a system design?

What properties can be proved about an implementation?

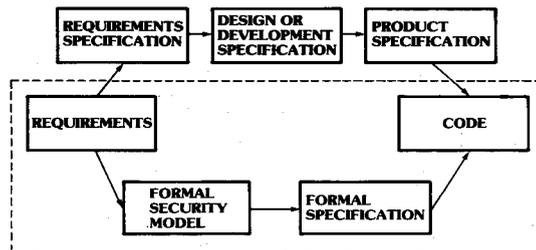
The Problem



The Problem Dissected



Software Development Approaches



Types of Models

Access Control

Considers subjects and objects

Requirements:

- If S has read access to O, security - level (S) \geq security - level (O)
- If S has write access to O, security - level (S) \leq security - level (O)

Flow Analysis

Considers system variables

Requirement:

- If information can flow from A to B, security - level (A) \leq security - level (B)

Formal Specifications

State Machine

Relates values of variables before and after operations

Example

Exchange (X, Y)

New — value (X) = Y;

New — value (Y) = X;

Algebraic

Relates results of sequences of operations

Example

Exchange (Exchange (Pair)) = Pair;

First (Exchange (Pair)) = Last (Pair);

Last (Exchange (Pair)) = First (Pair);

Levels of Specifications

Stepwise Refinement

Lower levels describe the same operations in greater detail

Hierarchical

Lower levels describe operations used to implement higher levels

Design Verification - What is Proved?

Proof of Consistency Between Model and Specification

State invariants

Transition properties

Assumes:

Model is appropriate

Specification is complete

Design Verification - Practical Considerations

Usually Done with Automatic Theorem Provers
Easier than Code Verification
Can be Useful without Code Verification
Must be an Early Part of Software Development

Code Verification

Entry Assertion	$I \geq 0$	$J \geq 0$
·		
·		
program	exchange	
·		
·		
Exit Assertion	$I_{\text{final}} = J_{\text{start}}$	
	$J_{\text{final}} = I_{\text{start}}$	

Prove: If the entry assertion is true when the program begins, the exit assertion will be true when the program ends.

Inductive Assertion Method

Introduce Intermediate Assertions

Assertion 0 (Entry)

Code 0

Assertion 1

Code 1

·

Assertion N-1

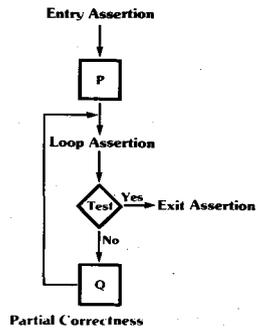
Code N-1

Assertion N (Exit)

Prove: If Assertion I is True, then Assertion I + 1 will be True After Code I is Run.

Verification Conditions

Loops



Code Verification - Practical Considerations

Harder than Design Verification
Many long verification conditions
Need loop assertions

Practical Only for Critical (Small) Portions of Code
Requires Automatic Theorem Provers
Part of the Software Development Process

Role of Automatic Theorem Provers

Many Long Theorems to Prove
Repeatable Results

Summary

Formal Verification: Proof of Consistency

Design verification:

Consistency between model and specification

Assumes:

Model is appropriate

Specification is complete

Code verification:

Consistency between specification and implementation

Assumes:

Specification is appropriate

Implementation language is correctly defined

FDM - A Specification and Verification Methodology

Richard A. Kemmerer *
System Development Corporation
Santa Monica, California 90406

1. Introduction

System Development Corporation's Formal Development Methodology (FDM) is an integrated methodology for the design, specification, implementation, and verification of software. FDM enforces rigorous connections between successive stages of development. The FDM is used as follows:

1. The correctness requirements for the software are modeled.
2. A top-level design specification is written and verified to be consistent with the model.
3. The design specification is repeatedly refined to include more detail until a program design specification is derived.
4. The intermediate design specifications and the program design specification are verified as the refinement process is carried out.
5. An implementation is coded from the program design specification and this implementation is verified to be consistent with the program design specification.

By verifying that specifications are consistent with the model, design errors are detected immediately rather than during implementation verification.

A key point about the FDM is that all theorems to be proved about specifications and implementation are generated automatically by the verification system. In addition, development stages are integrated: the output of one stage is used as the input to the next; a user need not massage the data into the format needed for the next stage. Furthermore, since all tools run on the same machine, the output from one tool is written directly on a file used as input to another tool.

2. The Components of FDM

Four basic components comprise the FDM verification system. These are the Ina Jo specification language, the Ina Jo processor, the interactive theorem prover (ITP), and the verification condition generator (VCG). Each component

Richard Kemmerer is a consultant to System Development Corporation working on enhancements to the FDM. He is an Assistant Professor in the Computer Science Department at the University of California, Santa Barbara.

is discussed in detail in the following sections.

2.1. The Ina Jo Language

The Ina Jo language is a non-procedural assertion language that is an extension of first-order predicate calculus. The language assumes that the system is modeled as a state machine. Key elements of the language are types, constants, variables, definitions, initial conditions, criterion, constraints, transforms, modules, levels, and mappings. The following paragraphs contain examples of some of these elements. An Ina Jo Specification that contains these examples is presented in Appendix A.

Some examples of types are:

```
type element,  
    subject<element,  
    access = (read,write,append,exec),  
    accesses = set of access
```

The type element is an unspecified type and subject is an unspecified subtype of element. The only operation that is defined on unspecified types is equality. Access is an enumerated type with four possible values, and accesses is a set of type access. The only primitive types in Ina Jo are integer and boolean.

The initial condition is an assertion that must hold for the initial state of the system. The following initial condition specifies that initially no subject has access of any type to any object.

```
initial A"s:subject,o:object  
    (accesses_allowed(s,o) = empty)
```

The correctness requirements of the system are modeled in Ina Jo by the criteria. The criteria was originally a conjunction of assertions called criterion that specified what was a good state. These are often referred to as state invariants since they must hold for all states. In the process of specifying real systems it was found that it was often necessary to include restrictions on the relationship of one state to the next in the model. To meet this demand a constraint was added to the criteria. The constraint is an invariant about state transitions that compares the old and new states. Thus, although it is not in agreement with the English language an Ina Jo correctness criteria is made up of the conjunction of the individual criterion and the constraint. The following example of a criterion specifies that for all subjects s and objects o if s has write access to o, then the class of s is equal to the class of o and the category of s is equal to the category of o.

```

A" s:subject, o:object(
    write<:accesses_allowed(s,o)
    -> class(s)=class(o) & catg(s)=catg(o))

```

An Ina Jo transform is a state transition function it specifies what the values of the state variables will be after the state transition relative to what their values were before the transition took place.

Only a subset of the Ina Jo language has been presented here. A complete description of the language can be found in the Ina Jo Reference Manual [LSS 80] and in the tutorial overview [Egg 80].

2.2. The Ina Jo Processor

The Ina Jo processor reads specifications written in Ina Jo and produces theorems to be proved by the interactive theorem prover. Two types of theorems are generated by the processor: consistency theorems and correctness theorems. Consistency theorems guarantee that the effect of a transform is not false, that defined terms are well defined, that type restrictions are observed, and that mappings are consistent. These theorems are usually existentializations. For instance, if the effect part of a transform contains $N"x=x+1$ and $N"x=x$ then a theorem is generated stating that there exists an element of the type of x that satisfies these two conditions. Since this reduces to false the specification cannot be proved consistent.

A number of correctness theorems are generated by the Ina Jo processor. One states that the initial conditions satisfy all of the criterion. This guarantees that the system is initially in a good state. In addition, for each transform in the top-level specification a theorem is generated that guarantees that the transform satisfies the criteria. This theorem states that if the old state satisfies all of the criterion then the new state will also satisfy all of the criterion, and that the relationship between the old and new states satisfies the constraint. Since the initial state is shown to satisfy the criteria, and following any transform that starts in a state that satisfies the criteria the new state satisfies the criteria, by induction one can conclude that all states satisfy the criteria.

In addition to the theorems generated for the top-level specification, it is necessary to generate correctness theorems that guarantee that each lower-level specification correctly implements the corresponding higher-level transform with respect to the mappings.

Finally, it is possible to introduce transforms at the lower levels that do not correspond to any transform at the

level above; it is necessary to generate correctness theorems for these transforms that guarantee that they satisfy a mapping of the criteria.

In addition to generating consistency and correctness theorems the Ina Jo processor must generate entry and exit assertions for each of the high order language procedures that implements a transform in the program design specification. To do this the Ina Jo processor needs to know how the objects of the lowest level specification (program design specification) map on to objects in the high order language (HOL) implementation. This is provided by the implementation specification which is nothing more than these mappings. Thus, the Ina Jo processor for this step in the verification process accepts as input the program design specification and the implementation specification and outputs the entry and exit assertions for the HOL procedures that implement transforms of the program design specification.

2.3. The Interactive Theorem Prover

The interactive theorem prover (ITP) aids the user in documenting the proofs of long theorems. The ITP uses the principle of reductio ad absurdum (proof by contradiction). That is, the first step in the proof process is for the ITP to automatically assume the contrary and the user then proceeds to show that this assumption reduces to false.

The design of the ITP adheres to the following objectives: all proofs must be automatically checked for soundness, the user must be in complete control, the output must be in a format that can be audited, and the user must be relieved of typing voluminous amounts of information that can be typed by the theorem prover under user direction. The following paragraphs discuss how these design objectives have been met.

Each time the user directs the ITP to perform a step the ITP checks its knowledge base to see if the step is logically sound. If the step is not logically sound it will not be performed and the user will be notified.

The proofs are written in a human-readable form by adopting a Dewey Decimal like line numbering scheme that indicates the step sequence in the proof as well as the nesting level. That is, each time a new step of the proof is executed the last part of the line number is incremented by one. In addition, each time a theorem is needed to complete the proof the user states the theorem and the current line number has a decimal point and a one appended to it to arrive at the next line number. Thus, each decimal point indicates the nesting of theorems being proved. When the proof of a theorem is completed the last decimal point and

any numbers following it are removed. The proofs are also made more readable by appending English justifications to each proof step. For instance, when the ITP automatically assumes the contrary this step has "ASSUME" appended to it. Also, if a result of false is derived from contradicting statements at steps 11.3 and 11.12, then this step has "(11.3 11.12)CONTRADICTION" appended to it.

The ITP accomplishes automatic deductions by generating corollaries to proof steps. These corollaries are numbered with the proof step number followed by a hyphen and then an integer value (see example below). An example of when corollaries are generated is when the proof step is a conjunction of predicates and the ITP automatically splits these into the individual conjuncts each as a separate corollary. Although the ITP performs most deductive steps automatically, it never enters into lengthy excursions to heuristically discover deductions. For instance it never attempts substitutions unless the user requests a particular substitution.

To give an example of the numbering scheme and the proof by contradiction approach consider the following scenario. After executing proof step 99 the user realizes he would like to have a theorem to use in the proof; therefore, he states this theorem as step 100.

100 H1 & H2 & H3 -> C1 & C2

The theorem consists of three hypothesis H1, H2, and H3 and two conclusion C1 and C2. Since the ITP uses the method of proof by contradiction it automatically assumes the contrary. In addition since the proof of this theorem introduces a new level the next line number is the previous line number with ".1" appended to it. Thus, the next line is

100.1 H1 & H2 & H3 & (~C1 | ~C2)

Next the ITP automatically splits this conjunction getting the following four corollaries.

100.1-1 H1
100.1-2 H2
100.1-3 H3
100.1-4 ~C1 | ~C2

The user next proceeds to prove that ~C1 is false and that ~C2 is false which yields corollary 100.1-4 to be false, which reduces 100.1 to false, and thus proves the theorem stated at line 100.

A detailed discussion of the ITP can be found in the ITP User's Manual [Sch 80].

2.4. The Verification Condition Generator

For the verification process to be complete, it is necessary to perform code level proofs in addition to specification verifications. To meet this need a verification condition generator (VCG) for Modula is currently being built. The VCG accepts as input the asserted HOL code of the implementation and the entry and exit assertions output by the Ina Jo processor. The output of the VCG is the verification conditions (theorems) that assert that each subroutine satisfies its exit assertion assuming that its entry assertion holds at the point of invocation. The verification conditions output by the VCG are used as input to the ITP which is used to prove them.

3. Applications of the FDM

The FDM has been thoroughly tested on a variety of real-world problems. Most noteworthy of the systems to which the FDM tools have been applied include:

1. An operating system kernel for KVM/370
2. Three kernels for a secure network system
3. A capability based Secure Transaction Processing System (STPS)
4. A system for automating the periods processing for a large scientific processor using a Job Stream Separator (JSS) approach
5. A secure network front-end

For KVM the kernel as well as four trusted processes running on the kernel had top-level specifications written and verified. The top-level specifications are to be refined to lower level specifications which will also be verified.

The specifications for the second system were written by non-SDC personnel. These specifications included top-level specifications for three different kernels of which each node of the system was comprised. Each of the specifications was verified to be consistent with its correctness criteria.

For the STPS there were three levels of Ina Jo specification written of which the top two were verified to be consistent with the STPS correctness criteria.

There are presently two levels of specification written for the JSS. The top-level specification has been verified and the second level specification is in the process of being verified. The code for this system is being written in Modula, and the Modula VCG will be used to perform code level verification of the system.

The specification and verification of the secure

network front-end is also currently in process. This system includes an executive and twenty trusted processes. At the present time the top-level specification for the executive has been written and verified and the second level specification is being written. In addition top-level specifications for two of the trusted processes are being written. Parts of this system may be verified down to the code level.

4. Conclusions

The Formal Development Methodology is a specification and verification methodology that is well integrated and rigorous. FDM is capable of performing verification against a variety of correctness criteria without requiring any changes to the tools. The methodology has been successfully applied to a number of complex real-world systems. Although to date none of these verification efforts have been carried to code level, this will be done in the near future. FDM is a useful methodology for systems that warrant the cost of formal verification.

5. Acknowledgments

The principal designers and implementors of the FDM and its tools are John Scheid and Val Schorre. Also currently active in enhancements to the tools are Sue Landauer and Paul Eggert.

6. References

- [Egg 80] Eggert, Paul R., "Overview of the Ina Jo Specification Language," System Development Corporation document SP-4082, October 1980.
- [LSS 80] Locasso, R., J. Scheid, V. Schorre, and P. Eggert, "The Ina Jo Specification Language Reference Manual," System Development Corporation document TM-(L)-6021/001/00, June 1980.
- [Sch 80] Schorre, V., "The Interactive Theorem Prover (ITP) User's Manual," System Development Corporation document (in preparation).

Appendix A - A Specification Example

```
00010 $TITLE EXAMPLE
00020 SPECIFICATION EXAMPLE
00030 LEVEL TOP_LEVEL
00040
00050 TYPE ELEMENT,
00060 SUBJECT < ELEMENT,
00070 OBJECT < ELEMENT
00080
00090 TYPE ACCESS = (READ, WRITE, APPEND, EXEC),
00100 CLASSIFICATION,
00110 CATEGORY
00120
00130 TYPE CATEGORIES = SET OF CATEGORY,
00140 ACCESSES = SET OF ACCESS
00150
00160
00170 CONSTANT
00180 CLASS(ELEMENT):CLASSIFICATION,
00190 CATG(ELEMENT):CATEGORIES
00200
00210 CONSTANT
00220 OK_TO_WRITE(S:SUBJECT,O:OBJECT):BOOLEAN =
00230 CLASS(S) = CLASS(O)
00240 & CATG(S) = CATG(O)
00250
00260
00270 VARIABLE
00280 ACCESSES_ALLOWED(SUBJECT,OBJECT):ACCESSES
00290
00300
00310 INITIAL
00320 A*S:SUBJECT,O:OBJECT(ACCESSES_ALLOWED(S,O) = EMPTY)
00330
00340 CRITERION
00350 A*S:SUBJECT,O:OBJECT(
00360 ( WRITE <: ACCESSES_ALLOWED(S,O)
00370 -> CLASS(S) = CLASS(O) & CATG(S) = CATG(O))
00380 )
00390
00400 TRANSFORM GET_WRITE_ACCESS(S:SUBJECT,O:OBJECT) EXTERNAL
00410 EFFECT
00420 A*S1:SUBJECT,O1:OBJECT(
00430 N*ACCESSES_ALLOWED(S1,O1) =
00440 ( OK_TO_WRITE(S,O)
00450 & S1 = S
00460 & O1 = O =>
00470 ACCESSES_ALLOWED(S1,O1) || S*(WRITE)
00480 <> ACCESSES_ALLOWED(S1,O1))
00490 )
00500
00510 END TOP_LEVEL
00520
```

```

00530 LEVEL SECOND_LEVEL UNDER TOP_LEVEL
00540
00550 TYPE ELEMENT,
00560     SUBJECT1 < ELEMENT,
00570     OBJECT < ELEMENT,
00580     SUBJECT2 < OBJECT
00590
00600 TYPE
00610     ACCESS = (READ, WRITE, APPEND, EXEC),
00620     COM_ACCESS = (READ, WRITE),
00630     CLASSIFICATION = (UNCLASSIFIED, CONFIDENTIAL,
00640                     SECRET, TOP_SECRET),
00650     CATEGORY
00660
00670 TYPE CATEGORIES = SET OF CATEGORY,
00680     FILE_ACCESSSES = SET OF ACCESS,
00690     COM_ACCESSSES = SET OF COM_ACCESS
00700
00710
00720 CONSTANT
00730     CLASS(ELEMENT):CLASSIFICATION,
00740     CATG(ELEMENT):CATEGORIES
00750
00760 CONSTANT
00770     OK_TO_WRITE(S:SUBJECT1,O:OBJECT):BOOLEAN =
00780         CLASS(S) = CLASS(O)
00790         & CATG(S) = CATG(O)
00800
00810 VARIABLE
00820     ACCESSSES_GRANTED(SUBJECT1,OBJECT):FILE_ACCESSSES,
00830     COMMUNICATION_ACCESSSES(SUBJECT1,SUBJECT2):COM_ACCESSSES,
00840     ACTIVE_USER1(SUBJECT1):BOOLEAN,
00850     ACTIVE_USER2(SUBJECT2):BOOLEAN
00860
00870 INITIAL
00880     A^E1,E2:ELEMENT(
00890         ACCESSSES_GRANTED(E1,E2) = EMPTY
00900         & ( E^S1:SUBJECT1,S2:SUBJECT2(S1 = E1 & S2 = E2)
00910             -> COMMUNICATION_ACCESSSES(E1,E2) = EMPTY)
00920         )
00930     & A^S1:SUBJECT1(~ACTIVE_USER1(S1))
00940     & A^S2:SUBJECT2(~ACTIVE_USER2(S2))
00950
00960 TRANSFORM GRANT_SEND(S1:SUBJECT1,S2:SUBJECT2)
00970 EFFECT
00980     A^T1:SUBJECT1,T2:SUBJECT2(
00990         N^COMMUNICATION_ACCESSSES(T1,T2) =
01000         ( OK_TO_WRITE(S1,S2)
01010           & ACTIVE_USER1(S1)
01020           & ACTIVE_USER2(S2)
01030           & T1 = S1
01040           & T2 = S2 =>
01050             COMMUNICATION_ACCESSSES(T1,T2) || S^(
WRITE)
01060             <> COMMUNICATION_ACCESSSES(T1,T2))
01070         )
01080

```

```

01090  TRANSFORM GRANT_WRITE(S:SUBJECT1,0:OBJECT)
01100  EFFECT
01110      A*S1:SUBJECT1,01:OBJECT(
01120          N*ACCESSES_GRANTED(S1,01) =
01130              ( OK_TO_WRITE(S,0)
01140                  & A*S2:SUBJECT2(S2^=0)
01150                  & S1 = S
01160                  & 01 = 0          =>
01170                      ACCESSES_GRANTED(S1,01) || S*(WRITE)
01180              <>                      ACCESSES_GRANTED(S1,01))
01190          )
01200
01210  TRANSFORM LOGON(S:SUBJECT1) EXTERNAL
01220  EFFECT
01230      A*S1:SUBJECT1(
01240          N*ACTIVE_USER1(S1)=
01250              ( S1 = S          => TRUE
01260              <>                      ACTIVE_USER1(S1)
01270          )
01280      & A*S2:SUBJECT2(
01290          N*ACTIVE_USER2(S2)=
01300              ( S2 = S          => TRUE
01310              <>                      ACTIVE_USER2(S2)
01320          )
01330
01340  MAP
01350      ELEMENT == ELEMENT,
01360      SUBJECT == SUBJECT1,
01370      OBJECT == OBJECT,
01380      ACCESS == ACCESS,
01390      CLASSIFICATION == CLASSIFICATION,
01400      CATEGORY == CATEGORY,
01410      CATEGORIES == CATEGORIES,
01420      ACCESSES == FILE_ACCESSES,
01430      READ == READ,
01440      WRITE == WRITE,
01450      APPEND == APPEND,
01460      EXEC == EXEC,
01470
01480      CLASS(E) == CLASS(E),
01490      CATG(E) == CATG(E),
01500      OK_TO_WRITE(S,0) == OK_TO_WRITE(S,0),
01510
01520      ACCESSES_ALLOWED(S,0) ==
01530          (E*S2:SUBJECT2(0 = S2) =>
01540              COMMUNICATION_ACCESSES(S,0)
01550          <>                      ACCESSES_GRANTED(S,0)),
01560
01570      GET_WRITE_ACCESS(S,0) ==
01580          ( E*S2:SUBJECT2(S2 = 0)  =>
01590              GRANT_SEND(S,0)
01600              & NC*(ACCESSES_GRANTED)
01610          <>                      GRANT_WRITE(S,0)
01620              & NC*(COMMUNICATION_ACCESSES)
01630          )
01640
01650  END SECOND_LEVEL
01660
01670  END EXAMPLE

```

FDM

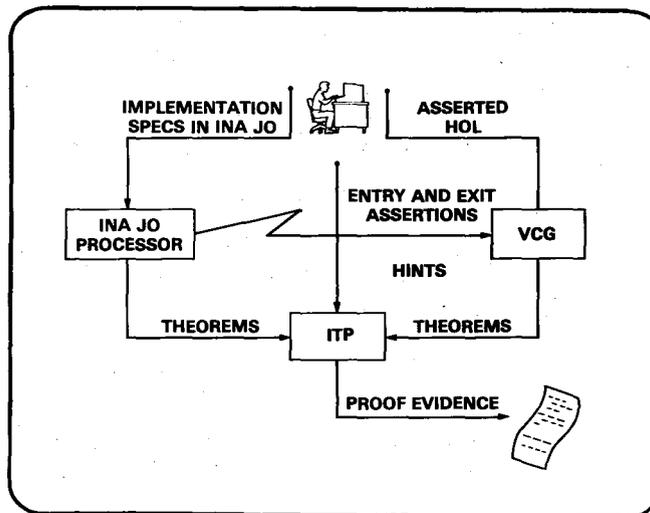
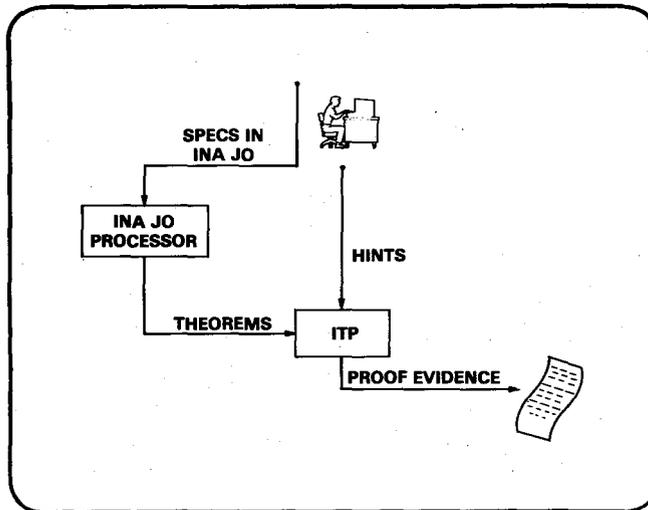
A FORMAL METHODOLOGY FOR SOFTWARE DEVELOPMENT

FDM

- INTEGRATED METHODOLOGY FOR DESIGN, SPECIFICATION, IMPLEMENTATION AND VERIFICATION OF SOFTWARE
- ENFORCES ESTABLISHMENT OF RIGOROUS CONNECTIONS BETWEEN SUCCESSIVE STAGES OF DEVELOPMENT
 - IDENTIFICATION AND MODELLING OF REQUIREMENTS
 - DESIGN SPECIFICATIONS
 - VERIFICATION OF SPECIFICATIONS
 - PROGRAM DESIGN SPECIFICATIONS
 - VERIFICATION OF IMPLEMENTATION

TOOLS OF FDM

- SPECIFICATION LANGUAGE (INA JO)
- LANGUAGE PROCESSOR
- INTERACTIVE THEOREM PROVER (ITP)
- VERIFICATION CONDITION GENERATOR (VCG)



INA JO LANGUAGE

- STATE MACHINE REPRESENTATION
- NON-PROCEDURAL
- ASSERTION LANGUAGE: EXTENSION OF FIRST-ORDER PREDICATE CALCULUS
- LANGUAGE ELEMENTS
 - TYPES
 - CONSTANTS
 - VARIABLES
 - DEFINITIONS
 - INITIAL CONDITIONS
 - CRITERION
 - CONSTRAINTS
 - TRANSFORMS
 - MODULES
 - LEVELS
 - MAPPINGS

- TYPE ELEMENT,
SUBJECT < ELEMENT,
OBJECT < ELEMENT
- TYPE ACCESS = (READ, WRITE, APPEND, EXEC),
ACCESSES = SET OF ACCESS
- TYPE TIME = INTEGER

- CONSTANT
CLASS (ELEMENT) : CLASSIFICATION
- VARIABLE
ACCESSES_ALLOWED (SUBJECT, OBJECT) : ACCESSES
- DEFINE
OK_TO_WRITE (S:SUBJECT, O:OBJECT) : BOOLEAN = =
CLASS(S) = CLASS(O)
& CATG(S) = CATG(O)

- INITIAL
A" S: SUBJECT, O: OBJECT
(ACCESSES_ALLOWED (S,O) = EMPTY)
- CRITERION
A" S: SUBJECT, O: OBJECT (
WRITE < : ACCESSES_ALLOWED (S,O)
→ CLASS(S) = CLASS(O) & CATG(S) = CATG(O))
- CONSTRAINT
N"TIME > TIME | N"TIME = 0 & TIME > 0

- **CRITERIA = CRITERION + CONSTRAINT**
- **CRITERION IS AN INVARIANT ABOUT STATES**
- **CONSTRAINT IS AN INVARIANT ABOUT STATE TRANSITIONS**

- **TRANSFORM GET_WRITE_ACCESS (S:SUBJECT, O:OBJECT)**

EXTERNAL

EFFECT

A'' S1: SUBJECT, O1: OBJECT (

N'' ACCESSES_ALLOWED (S1, O1) =

(OK_TO_WRITE (S,O)

& S1 = S

& O1 = O

=> ACCESSES_ALLOWED (S1, O1) || S'' (WRITE)

<> ACCESSES_ALLOWED (S,O)))

MAPPINGS

- **ALL TYPES, CONSTANTS, VARIABLES, AND EXTERNAL TRANSFORMS ARE MAPPED TO THE NEXT LOWER LEVEL**

- **E.G.,**

GET_WRITE_ACCESS (S,O) = =

(E'' S2: SUBJECT2 (S2 = O) =>

GRANT_SEND (S,O)

& NC'' (ACCESSES_GRANTED)

<> GRANT_WRITE (S,O)

& NC'' (COMMUNICATION_ACCESSES)

)

INA JO PROCESSOR

- **READS SPECIFICATIONS, INCLUDING CRITERIA AND MAPPINGS**
- **GENERATES CONSISTENCY AND CORRECTNESS THEOREMS**
- **GENERATES ENTRY AND EXIT ASSERTIONS FOR PROGRAM MODULES FROM IMPLEMENTATION LEVEL SPECIFICATION**

CONSISTENCY THEOREMS

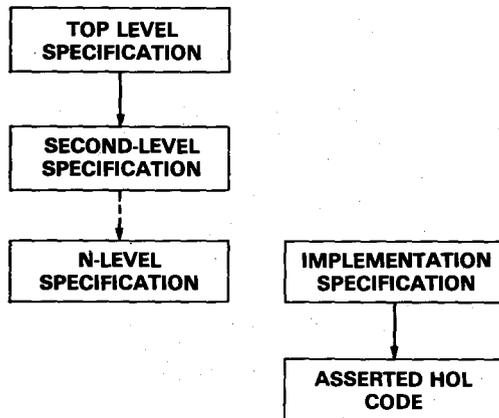
- **EFFECT OF TRANSFORM NOT "FALSE"**
- **DEFINED TERMS ARE WELL-DEFINED**
- **TYPE RESTRICTIONS ARE OBSERVED**
- **MAPPINGS ARE CONSISTENT**

CORRECTNESS THEOREMS

- **INITIAL CONDITIONS SATISFY CRITERIA**
- **TLS TRANSFORMS SATISFY CRITERIA**
- **LOWER-LEVEL TRANSFORMS CORRECTLY IMPLEMENT CORRESPONDING HIGHER-LEVEL TRANSFORMS WITH RESPECT TO MAPPINGS**
- **LOWER-LEVEL TRANSFORMS THAT DO NOT CORRESPOND TO HIGHER-LEVEL TRANSFORMS SATISFY A MAPPING OF THE CRITERIA**

TLS TRANSFORM SATISFIES CRITERIA

- **RELATIONSHIP BETWEEN OLD STATE AND NEW STATE SATISFIES CONSTRAINT**
- **IF OLD STATE SATISFIES CRITERION THEN SO DO NEW STATES**



INTERACTIVE THEOREM PROVER (ITP)

- **AIDS THE USER IN FINDING AND DOCUMENTING PROOFS OF LONG THEOREMS.**
- **USES THE PRINCIPLE OF REDUCTIO AD ABSURDUM.**

OBJECTIVES OF THE ITP

- ALL PROOFS MUST BE AUTOMATICALLY CHECKED FOR SOUNDNESS.
- THE USER MUST BE IN COMPLETE CONTROL.
- THE OUTPUT MUST BE IN A FORMAT THAT CAN BE AUDITED.
- THE USER MUST BE SAVED FROM VOLUMINOUS TYPING OF PROOFS.

ITP

- CHECKS ALL PROOF STEPS FOR LOGICAL SOUNDNESS
- WRITES PROOFS IN HUMAN-READABLE FORM
 - PROOFS ORGANIZED IN NESTED FASHION
 - LINE NUMBERS INDICATE STEP SEQUENCE AND NESTING LEVEL
 - ENGLISH JUSTIFICATION AUTOMATICALLY APPENDED TO EACH PROOF STEP
- PERFORMS MOST DEDUCTIVE STEPS AUTOMATICALLY, BUT NEVER ENTERS INTO LENGTHY EXCURSIONS TO HEURISTICALLY DISCOVER DEDUCTIONS
- ACCOMPLISHES AUTOMATIC DEDUCTIONS BY GENERATING COROLLARIES TO PROOF STEPS AS THEY ARE PRODUCED, E.G.,
 - SIMPLIFICATION
 - INSTANTIATION
 - AND SPLITTING

EXAMPLE

USER STATES THEOREM AT STEP 100

100 $H_1 \& H_2 \& H_3 \rightarrow C_1 \& C_2$

ITP ASSUMES THE CONTRARY

100.1 $H_1 \& H_2 \& H_3 \& (\neg C_1 \mid \neg C_2)$

ITP AUTOMATICALLY AND SPLITS

100.1-1 H_1

100.1-2 H_2

100.1-3 H_3

100.1-4 $\neg C_1 \mid \neg C_2$

ANNOTATION EXAMPLES

ASSUME

(15.1) 'AND SPLIT'

(40.8 40.10) SUBSTITUTION (40.10L)

(38.11-3 38.2-2) CONTRADICTION

(23.12.3) 'Q.E.D.'

VERIFICATION CONDITION GENERATOR (VCG)

- **ACCEPTS AS INPUT**
 - **HOL CODE**
 - **ENTRY AND EXIT ASSERTIONS FROM INA JO**
 - **ADDITIONAL ASSERTIONS IMBEDDED IN HOL CODE**
- **GENERATES VERIFICATION CONDITIONS THAT ASSERT THAT EACH SUBROUTINE SATISFIES ITS EXIT ASSERTION ASSUMING ENTRY ASSERTION HOLDS AT POINT OF INVOCATION**
- **VERIFICATION CONDITIONS THEN PROVED TO BE THEOREMS USING ITP**

APPLICATIONS OF FDM

- **OPERATING SYSTEM KERNEL FOR KVM/370**
- **KERNELS FOR A SECURE NETWORK SYSTEM**
- **CAPABILITY BASED SECURE TRANSACTION PROCESSING SYSTEM**
- **JOB STREAM SEPARATOR FOR AUTOMATING THE PERIODS PROCESSING FOR A LARGE SCIENTIFIC PROCESSOR**
- **SECURE NETWORK FRONT-END**

FUTURE DIRECTIONS

- **BETTER USER INTERFACE**
 - **CRT WITH EXTENDED SEARCH CAPABILITY**
 - **PROOF TREES**
- **DIRECT PROOF OPTION**
- **AUTOMATING STEPS THAT ARE ALWAYS PERFORMED BY THE USER OF ITP**

PRINCIPAL DESIGNERS

JOHN SCHEID

VAL SCHORRE

**BUILDING
VERIFIED SYSTEMS
WITH
GYPSY
DONALD I. GOOD
UNIVERSITY OF TEXAS**

**GYPSY
WHAT DOES IT DO?
HOW DOES IT WORK?
WHAT HAS BEEN DONE?
WHAT IS THE CURRENT STATUS?**

WHAT DOES GYPSY DO?

PURPOSE

THE PURPOSE OF GYPSY IS THE DEVELOPMENT OF VERY HIGHLY RELIABLE SOFTWARE SYSTEMS.

APPROACH

GYPSY IS A WELL-INTEGRATED SYSTEM OF METHODS, LANGUAGES, AND TOOLS FOR SPECIFYING, IMPLEMENTING, AND VERIFYING OPERATIONAL SOFTWARE SYSTEMS.

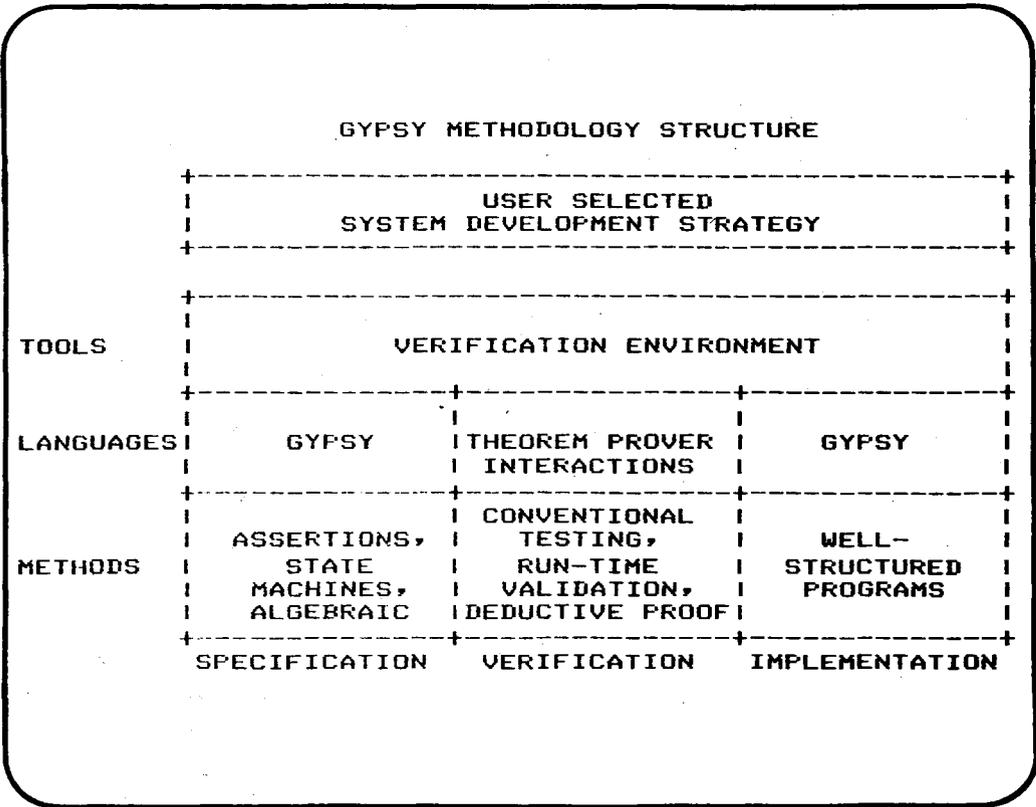
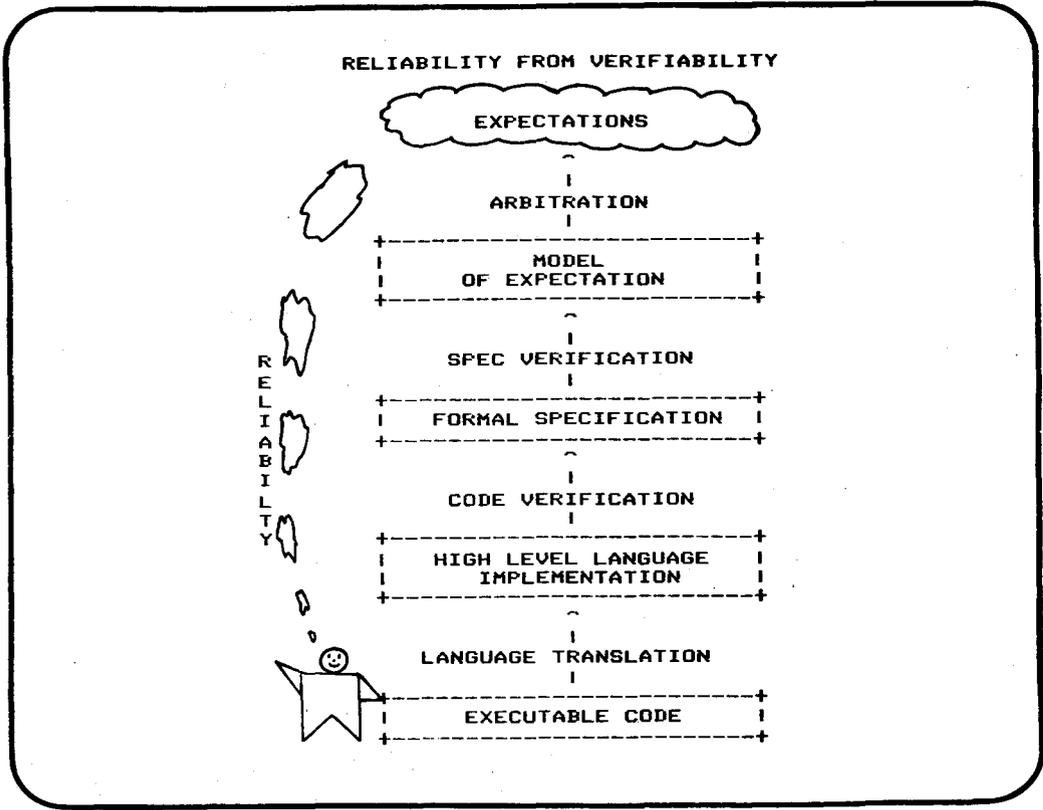
HOW DOES GYPSY WORK?

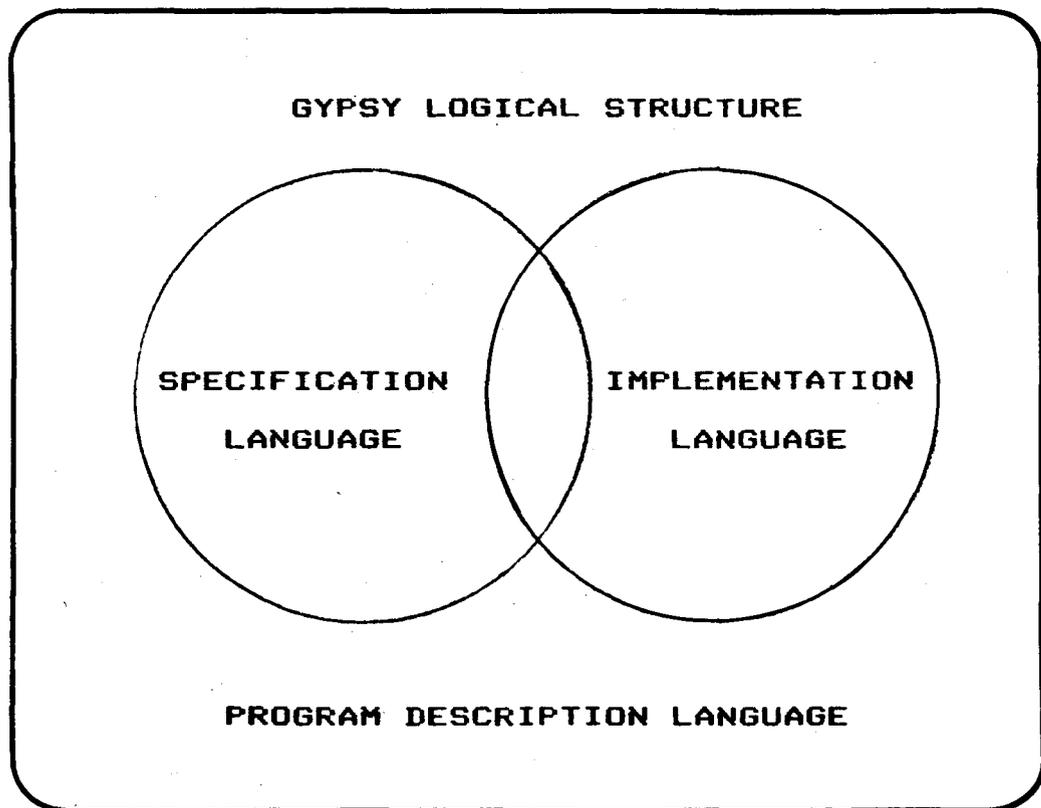
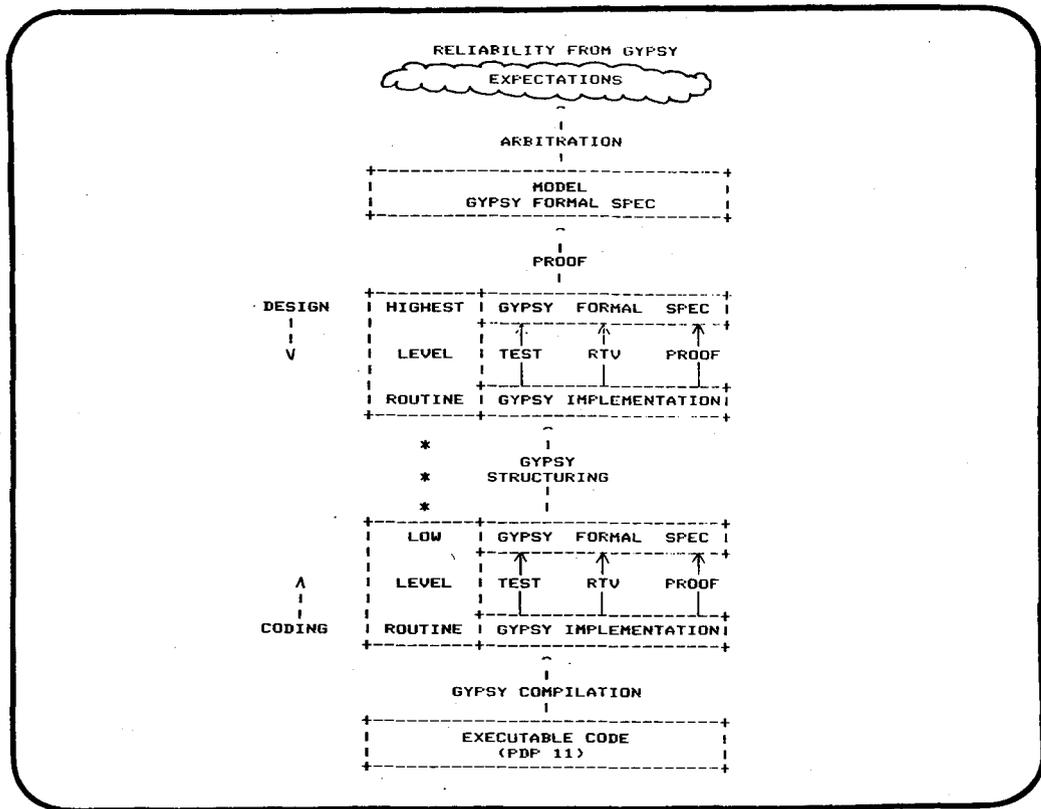
LANGUAGE

THE GYPSY LANGUAGE DESCRIBES ROUTINES THAT OPERATE ON OBJECTS. THE DESCRIPTION INCLUDES BOTH IMPLEMENTATION AND SPECIFICATION.

VERIFICATION ENVIRONMENT

THE VERIFICATION ENVIRONMENT IMPLEMENTS THE TOOLS NEEDED TO CONSTRUCT AND EXECUTE A SET OF VERIFIED GYPSY ROUTINES. THE ENVIRONMENT AMPLIFIES HUMAN CAPABILITY AND REDUCES PROBABILITY OF HUMAN ERROR.





GYPSY TEXTUAL STRUCTURE

SCOPE DEMO=
BEGIN

PROCEDURE P(VAR X: IN_BUFF) = ...;

FUNCTION F(N: INTEGER): INTEGER = ...;

TYPE HISTORY = SEQUENCE OF PACKET;

CONST HI = 256;

LEMMA MAKE_SECURE (A, B: HISTORY) = ...;

NAME {UNIT} U FROM {SCOPE} S;

END;

ROUTINES

PREDEFINED: FUNCTIONS FOR PREDEFINED
 TYPES

ASSIGNMENT

IF, CASE, LOOP, LEAVE, SIGNAL

MOVE, REMOVE

SEND, RECEIVE, GIVE

COBEGIN, AWAIT

USER DEFINED: FUNCTIONS, PROCEDURES

LOGICAL STRUCTURE OF ALL ROUTINES

WHAT?	EXTERNAL	INTERFACE SPEC
		FUNCTIONAL SPEC
HOW?	INTERNAL	LOCAL VARIABLES
		OPERATIONS AND SPECS

TEXTUAL STRUCTURE OF PROCEDURES

```

PROCEDURE DOWNGRADER (VAR H: IN_BUF; ...) ] INTERFACE
                                           ] SPEC
BEGIN
  BLOCK AUTHORIZED_DOWNGRADING (...); ] FUNCTIONAL
  ... ] SPEC
  VAR MESSAGE: TEXT; ] LOCAL
  ... ] VARS
  LOOP
  SPEC [ ASSERT OUTTO (L,MYID) ] OPERATIONS
        =AUTHORIZED_SEQ (INFROM (H,MYID)); ] AND
        RECEIVE MESSAGE ...; ] SPECS
  ...
  END;
END;

```

TEXTUAL STRUCTURE OF FUNCTIONS

```
FUNCTION F (N: INTEGER): INTEGER =  
BEGIN  
    ENTRY N>0;  
    EXIT RESULT = FACTORIAL (N);  
    VAR I: INTEGER := 1;  
    RESULT := 1;  
    LOOP  
        ...  
    END;  
END;
```

Diagram illustrating the textual structure of functions, with brackets on the right side grouping the code into sections:

- INTERFACE SPEC (bracketed next to the function signature)
- FUNCTIONAL SPEC (bracketed next to the ENTRY and EXIT statements)
- LOCAL VARS (bracketed next to the VAR statement)
- OPERATIONS AND SPECS (bracketed next to the LOOP and END statements)

SPECIFICATION FUNCTIONS

```
FUNCTION FACTORIAL (X: INTEGER): INTEGER =  
BEGIN  
    ENTRY X GE 0;  
    EXIT (ASSUME RESULT =  
        IF X = 0 THEN 1  
        ELSE X * FACTORIAL(X - 1) FI);  
END;
```

STATE TRANSITION SPECIFICATIONS

```

PROCEDURE SYSTEM (VAR S: SYS_OBJECTS) =
BEGIN
  EXIT ALLOWED_TRANSITION (S', S);
  PENDING;
END;

FUNCTION ALLOWED_TRANSITION (P, Q: SYS_OBJECTS)
: BOOLEAN =
BEGIN
  EXIT (ASSUME RESULT
  IFF IF IN_STATE_1 (P) THEN AFTER_1 (P,Q)
  ELSE IF IN_STATE_2 (P) THEN AFTER_2 (P, Q)
  ...
  ELSE FALSE FI...FI);
END;

LEMMA SECURITY_PRESERVED (P, Q: SYS_OBJECTS) =
IS_SECURE (P)
AND ALLOWED_TRANSITION (P, Q)
-> IS_SECURE (Q);

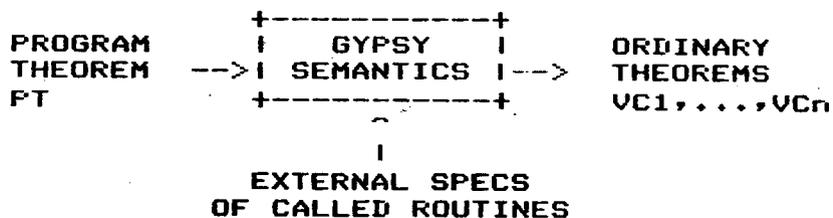
FUNCTION IS_SECURE (P: SYS_OBJECTS): BOOLEAN = ...;

```

PROVING ROUTINES

A ROUTINE TOGETHER WITH ITS SPECIFICATIONS IS A 'PROGRAM' THEOREM.

TRANSFORM



SUCH THAT

VC1 AND ... VCn --> PT

THEN, GIVEN LEMMAS L1, ..., Lm AS A BASIS,

PROVE

L1 AND ... Lm --> Vck FOR EACH k.

PROVING LEMMAS

A GYPSY LEMMA IS AN ORDINARY THEOREM OF PREDICATE CALCULUS.

EXAMPLE:

```
LEMMA AUTHORIZED_SUBSEQ (P, Q: HISTORY) =  
  P SUB Q ->  
  AUTHORIZED_SEQ (P) SUB AUTHORIZED_SEQ (Q);
```

PROVE A LEMMA L FROM OTHER LEMMAS L1, ..., Lk

```
L1 AND ... Lk -> L
```

DATA OBJECTS

GLOBAL CONSTANTS: CONST N = 4

FORMAL PARAMETERS: (VAR H: IN_BUFF; N: INTEGER)

LOCAL VARIABLES: VAR M: MESSAGE
 CONST P = 7

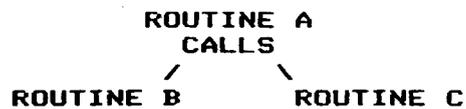
STRUCTURING

"SUMMARIZING: AS A SLOW WITTED HUMAN BEING I HAVE A VERY SMALL HEAD AND I HAD BETTER LEARN TO LIVE WITH IT AND TO RESPECT MY LIMITATIONS AND GIVE THEM FULL CREDIT, RATHER THAN TRY TO IGNORE THEM, FOR THE LATTER VAIN EFFORT WILL BE PUNISHED BY FAILURE."

EE.W. DIJKSTRA,
NOTES ON STRUCTURED
PROGRAMMING, 1972,
P33

GYPSY STRUCTURING

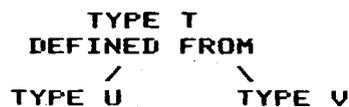
IMPLEMENTATION:



SPECIFICATION:



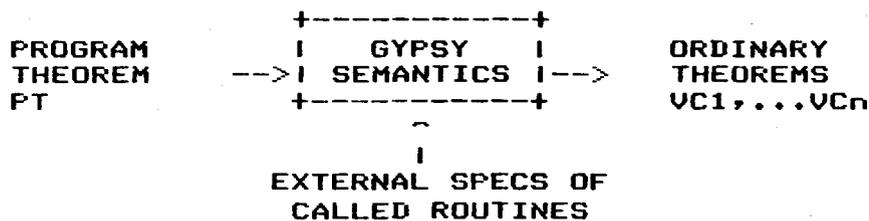
OBJECTS:



PROOF:



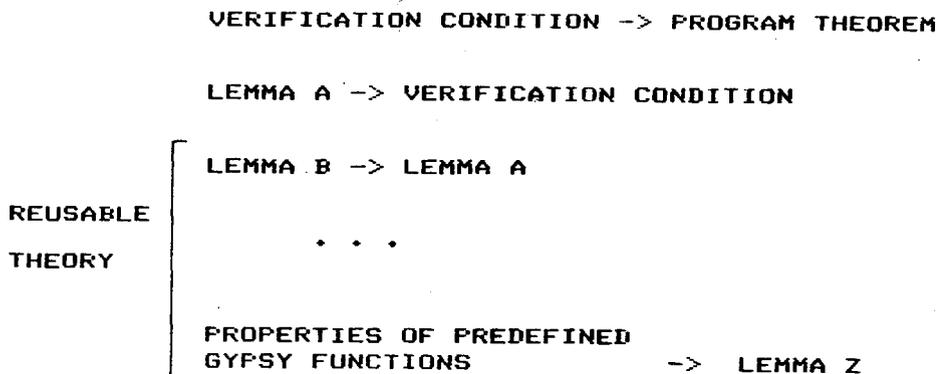
PROOF INDEPENDENCE



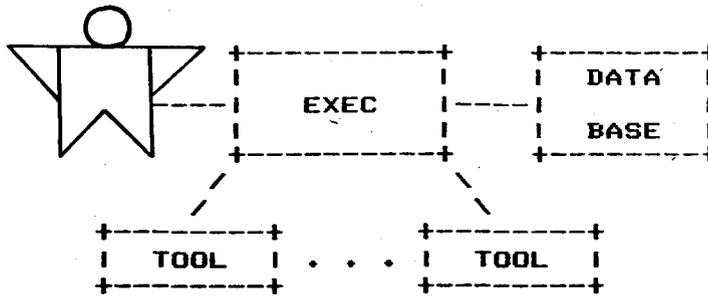
THE PROOF OF PT ASSUMES ONLY THAT CALLED ROUTINES CAN BE IMPLEMENTED TO MEET THEIR STATED EXTERNAL SPECS. THE PROOF IS INDEPENDENT OF ANY PARTICULAR IMPLEMENTATION.

THIS ALLOWS PROOF OF INDIVIDUAL ROUTINES TO BE DONE IN PARALLEL WITH ANY DESIRABLE ORDER OF DEVELOPMENT.

PROBLEM DOMAIN THEORIES



VERIFICATION ENVIRONMENT



TOOLS AVAILABLE:

GYPSY SYNTAX AND SEMANTIC ANALYZER,
SYNTAX DIRECTED EDITOR, VERIFICATION
CONDITION GENERATOR, INTERACTIVE THEOREM
PROVER, INTERPRETER, COMPILER, DATA BASE
DISPLAY, PROGRAM DEVELOPMENT MANAGER

UNDER DEVELOPMENT:

GYPSY TO BLISS TRANSLATOR, SPECIFICATION-
DRIVEN HIGH-LEVEL OPTIMIZER, CONVERSION
TO INTERLISP, EXPANSION OF DATA BASE
CAPACITY

TRIAL

APPLICATIONS

[WELLS, 76] NETWORK COMMUNICATION SYSTEM

- LAYER 1. 4-NODE MESSAGE SWITCHING NETWORK**
2. 4-NODE PACKETIZER/ASSEMBLER NETWORK
3. 5-NODE PACKET SWITCHING NETWORK

SPECIFICATION: 1500 LINES

IMPLEMENTATION: 1000 LINES

CONCURRENT PROCESSES: 16

VERIFICATION: MANUAL PROOFS OF CONCURRENCY

EXECUTABLE: NO

EFFORT: 1-2 WORK YEARS

[HORN, 77] SECURE INTERNETWORK

AN N-NODE NETWORK OF ACTUAL HOSTS WITH SENSITIVE INFORMATION COMMUNICATING VIA END-TO-END ENCRYPTION OVER AN UNSECURED INTERNETWORK THAT INCLUDES THE ARPANET.

SPECIFICATIONS: 372 LINES

IMPLEMENTATION: 10 LINES

CONCURRENT PROCESSES: UNSPECIFIED $N > 0$

**VERIFICATIONS: MANUAL,
35 MAJOR DEDUCTIVE STEPS,
20-40 PAGES**

EXECUTABLE: MODEL OF ACTUAL NETWORKS

EFFORT: .5-1 WORK YEAR

[MORICONI, 77] N X N MESSAGE SWITCHER

N CONCURRENT SWITCHER PROCESSES ROUTING
MESSAGES AMONG N USERS.

SPECIFICATIONS: 90 LINES

IMPLEMENTATION: 50 LINES

CONCURRENT PROCESSES: UNSPECIFIED $N > 0$

VERIFICATION: FULLY MECHANICAL AND
INCREMENTAL, 60 PAGES OF TRANSCRIPT.

EXECUTABLE: NO

EFFORT: 3-6 WORK MONTHS

[HAYNES AND NYBERG, 78] DISCRETE ADDRESS BEACON SYSTEM

SELECTED COLLISION AVOIDANCE ROUTINES FROM AN AIR
TRAFFIC CONTROL SYSTEM.

SPECIFICATIONS: 844 LINES (105 SPEC FUNCTIONS)

IMPLEMENTATION: 529 LINES (19 ROUTINES)

VERIFICATION: MECHANICALLY PROVED 30-40 OF 50 VCS.

EXECUTABLE: MODEL OF RUNNING FORTRAN IV PROGRAM

EFFORT: 1-2 WORK YEARS BY TEXAS INSTRUMENTS

[SMITH AND GOOD, 79] SIMPLE DISTRIBUTED GUARD

INTERACTIVELY MONITORS MESSAGE TRAFFIC BETWEEN
A HIGH SECURITY SYSTEM AND A LOW SYSTEM.
TERMINAL DRIVERS ARE PROVIDED TO SIMULATE HIGH
AND LOW SYSTEMS.

SPECIFICATIONS: 252 LINES

IMPLEMENTATION: 241 LINES

CONCURRENT PROCESSES: 15

VERIFICATION: MECHANICAL
32 PAGES OF FINAL PROOF
TRANSCRIPT

EXECUTABLE: ON PDP 11/03s

EFFORT: 2 WORK MONTHS

CURRENT STATUS

SPECIFICATION IMPLEMENTATION VERIFICATION METHODS	STABLE SINCE JAN 1979
GYPSY LANGUAGES	STABLE SINCE SEPT 1978
VERIFICATION ENVIRONMENT	IN EXPERIMENTAL USE. DEVELOPMENT AND MAINTENANCE IN PROGRESS.
DEVELOPMENT OF VERIFIED SYSTEM EXAMPLES	IN PROGRESS

ACKNOWLEDGMENTS

HDM

(Hierarchical Development Methodology)

An Approach to Designing Secure Systems and Proving
Them Correct

Karl Levitt
Computer Science Laboratory
SRI International
Menlo Park, CA

OUTLINE

- An Overview of HDM
- Writing "Good" Specifications in Special
- An Example of the Application of HDM -- PSOS (a "provably"
secure operating system)
- Formal Requirements for Secure Systems -- and how to prove
them
- HDM Tools
- Assessment of HDM
- Outstanding Problems

CREDITS

Creation of HDM and Special
Larry Robinson¹ (David Parnas)

HDM "Checking" Tools²
Olivier Roubine

Towards a Second Generation HDM
Brad Silverberg³, David Elliott, Joe Goguen

Formalization of HDM Subset -- and Theorem Proving
Bob Boyer, J. Moore

Design of PSOS
Peter Neumann, Larry Robinson, Rich Feiertag⁴

Multi-Level Security (MLS) Requirement
and Proof Tool
Rick Feiertag

Program Verification Tools
Dwight Hare, Mark Morkoki, Boyer, Moore

Specification of Concurrency
Les Lamport, Richard Schwartz, P. H. Melliar-Smith

1 Now at Ford Aerospace

2 Now at Honeywell

3 Now at Summit Systems

4 Now at Sytek

HDM is an Integrated Collection Of

- * Languages
- * Tools
- * Concepts
- * Guidelines

To Aid In Developing and Verifying Large Real-World Software Systems.

Developed at SRI From 1973 - Present

Distinguishing Characteristics of HDM.

- * Oriented Towards Real-World Solutions to Real-World Problems
- * Has a Formal Basis
- * Is Comprehensive
- * Is a Research Vehicle
- * Supports Verification
 - of design
 - of code

HDM Handles many of the "dirty" aspects of real-world systems, including

- * Resource Limitations
- * Resource Sharing
- * Side-Effects
- * Aliasing

Does not yet handle full concurrency

HDM is for use by the general community,
not just a sophisticated elite.

Still, learning HDM is a non-trivial task

A rigorous approach to software development
is intrinsically difficult

Applications of HDM

*PSOS --

- designed by SRI
- implementation underway at Ford Aerospace

*KSOS (at Ford and Honeywell) --

A Unix-Compatible O. S. supporting a multi-level
security policy

*SIFT --

A software implementation fault-
tolerant avionics computer. Production
and verification of sift is underway at SRI

HDM Structures at System Design

Vertical Structure

(Hierarchy of Abstract Machines --Dijkstra)
each level provides a set of facilities to the next higher level. The facilities at one level depend for implementation only on the facilities provided by the next lower level

The facilities provided by the top level are those available to the user

Horizontal Structure (provided by Modules)

Each module encapsulates closely related concepts, loosely coupled to other modules in the level

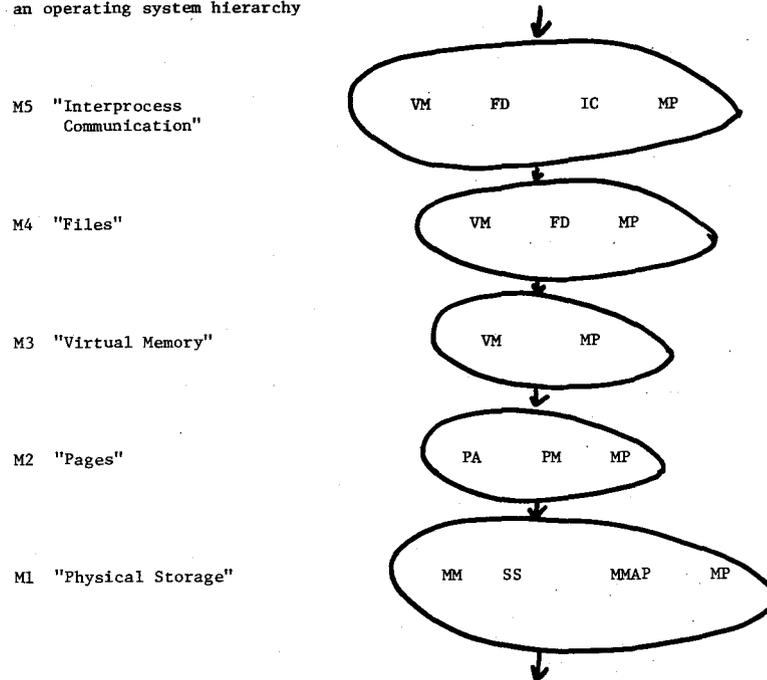
There are many examples of the abstract machine concept, e.g.:

- Families of Instruction Set Processors, e.g., IBM System/370
- Hierarchies of Communications Protocols
- Operating Systems (e.g., T.H.E., PSOS)

But,

The key is to formalize the concept

Some "key" levels in an operating system hierarchy



Key

VM Virtual Memory

FD File Directories

IC Interprocess Communications

MP Multiple Processes

PA Pages

PM Page Mapping

MM Main Memory

SS Secondary Storage

MMAP Memory Mapping

Figure 2: PSOS GENERIC DESIGN HIERARCHY

LEVEL	PSOS ABSTRACTION	PSOS LEVEL
F	USER ABSTRACTIONS	14-16
E	COMMUNITY ABSTRACTIONS	10-13
D	ABSTRACT OBJECT MANAGER	9
C	VIRTUAL RESOURCES	6-8
B	PHYSICAL RESOURCES	1-5
A	CAPABILITIES	0

Figure 1: PSOS DESIGN HIERARCHY

LEVEL	PSOS ABSTRACTION OR FUNCTION
16	USER REQUEST INTERPRETER *
15	USER ENVIRONMENTS AND NAME SPACES *
14	USER INPUT-OUTPUT *
13	PROCEDURE RECORDS *
12	USER PROCESSES * AND VISIBLE INPUT-OUTPUT *
11	CREATION AND DELETION OF USER OBJECTS *
10	DIRECTORIES (*)[C11]
9	EXTENDED TYPES (*)[C11]
8	SEGMENTATION AND WINDOWS (*)[C11]
7	PAGING [8]
6	SYSTEM PROCESSES AND INPUT-OUTPUT [12]
5	PRIMITIVE INPUT/OUTPUT [6]
4	ARITHMETIC AND OTHER BASIC OPERATIONS *
3	CLOCKS [6]
2	INTERRUPTS [6]
1	REGISTERS (*) AND ADDRESSABLE MEMORY [7]
0	CAPABILITIES *

* = MODULE FUNCTIONS VISIBLE AT USER INTERFACE.
 (*) = MODULE PARTIALLY VISIBLE AT USER INTERFACE.
 [I] = MODULE HIDDEN BY LEVEL I.
 [C11] = CREATION/DELETION ONLY HIDDEN BY LEVEL 11.

HDM also structures the development process -- into stages of development and verification.

The stages are:

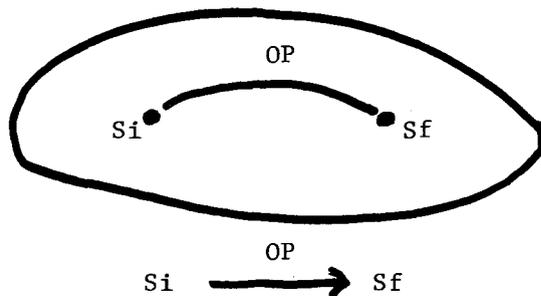
- | | | |
|------------------------|---|-------------|
| * Decomposition | } | Design |
| * Module Selection | | |
| * Module Specification | | |
| * Representation | } | Realization |
| * Implementation | | |

- Verification can be attempted as system develops
- Decisions are recorded as they are made
- Often, "important" decisions are made early and, hence, subject to early review (a system usually goes bad in design)

Motherhood: Recognize that backtracking and "crystal-ball gazing" are necessary. The "stages of HDM" are guidelines, not hard-and-fast rules

An abstract machine (or module) in HDM consists of:

1. A set of Internal Data Structures that defines its state
2. A set of operations that can access and modify the state



Realizing an abstract machine in terms of another

1. Data structure representation --

each upper-level state maps to a set of lower-level states, and distinct upper states must map to disjoint sets of lower states, i.e.,

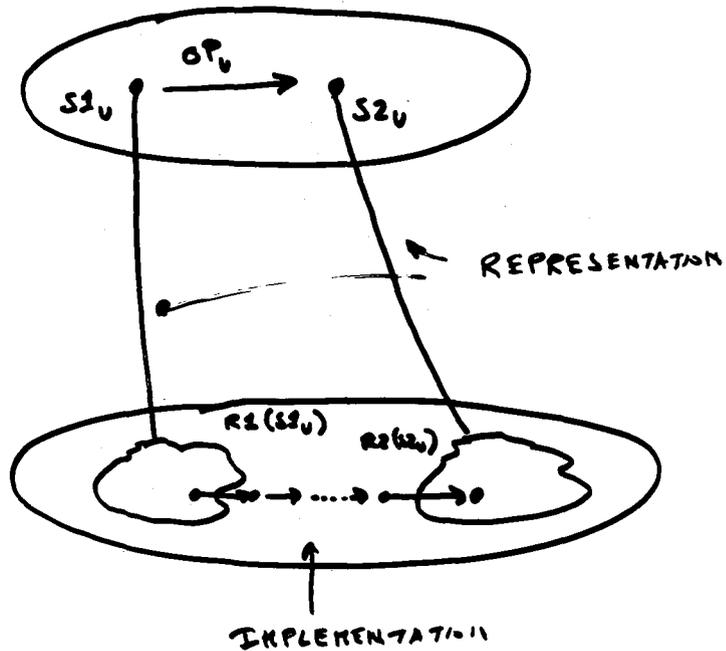
$$S1_u \neq S2_u \Rightarrow R(S1_u) \cap R(S2_u) = \emptyset$$

2. Operation Implementation --

Let operation OP_u take state $S1_u$ to $S2_u$,

$$S1_u \xrightarrow{OP_u} S2_u$$

An implementation of OP_u is correct if, when started from any state in $R(S1_u)$, it terminates in some state in $R(S2_u)$. (The well-known commutativity diagram illustrates this)



SPECIAL:

Special is HDM's module specification language.
A module specification specifies:

1. State-functions: functions that characterize the module's data structures, i.e., that determine its state.

The specification of a state-function provides its signature and constraints on its initial value.

2. Operations: The specification of an operation describes a state change and a returned value.

A state change is described by a predicate that nonprocedurally¹ relates the post-invocation values of the state-functions to their pre-invocation values.

The returned value is described in terms of constraints it must satisfy.

¹and non-deterministically

SYSTEM DESIGN WITH HDM & SPECIAL:

When given a problem statement, the first step is to formulate a model of a solution.

Generally, the more abstract the model at this point, the better. The process of hierarchical decomposition involves the formulating of successively more concrete models to implement the more abstract ones.

EXAMPLE:

Consider the problem of keeping word counts. The user is to be provided with the ability to:

1. Query the count for a given word.
2. Insert a word. If previously inserted, its count is incremented by one; if not, its count is set to one.
3. Delete a word occurrence.

Several alternative models are:

- a) An infinite mapping from words to integers, with the mapping initially everywhere undefined.

This can be pictured by an infinite (unordered) table:

?	3	12	...	?	...	15	...	1	...
a	b	c		aa		bar		foo	

The only defined words are "b", "c", "bar", & "foo" with counts 3, 12, 15, 1.

- b) An unsorted finite list of word, count pairs for defined words.

(⟨b,3⟩, ⟨bar, 15⟩, ⟨foo, 1⟩, ⟨c, 12⟩)

- c) Two integer-indexed arrays, one for defined words in sorted order, the other for counts

b	bar	c	foo	...
1	2	3	4	

3	15	12	1	...
1	2	3	4	

We'll choose alternative (a), the infinite mapping, since it is the most abstract.

In Special, it is specified:

```
VFUN word_store (word w) --> INTEGER count;  
  HIDDEN;  
  INITIALLY  
  FORALL word w1: word_store (w1) = UNDEFINED.
```

The query operation reads off word_store, trapping references to undefined words.

```
OVFUN Get_count (word w) -> INTEGER i;  
  EXCEPTIONS  
  undef: word_store(w) = UNDEFINED;  
  DERIVATION  
  word_store (w);
```

The operation `Insert_word` changes the state and returns as value the new count.

```
OVFUN Insert_word (word w) --> INTEGER cnt;  
  EXCEPTIONS  
    full: RESOURCE_ERROR;
```

EFFECTS

```
  IF word_store(w) = UNDEFINED  
    THEN 'word_store(w) = 1  
  ELSE 'word_store(w) = word_store(w) + 1;  
  cat = 'word_store(w);
```

IMPORTANT NOTE:

The EFFECTS section specifies an unordered conjunction of effects, and "=" denotes mathematical equality, not assignment.

The operation `Delete_word` deletes a word occurrence and returns the new count.

```
OVFUN Delete_word (word w) --> INTEGER cnt;
```

EXCEPTIONS

```
  undef: word_store(w) = UNDEFINED;
```

EFFECTS

```
  IF word_store(w) = 0 THEN  
    'word_store(w) = word_store(w) - 1  
  ELSE 'word_store(w) = 0;  
  
  'word_store(w) = cnt;
```

```

MODULE segment

TYPES
nat_number: { INTEGER j | j >= 0 };

PARAMETERS
INTEGER max_size;

FUNCTIONS

VFUN seg_val(nat_number i) -> INTEGER data;
HIDDEN;
INITIALLY
data = 7;

VFUN size() -> nat_number v;
RELATION
CARDINALITY({ nat_number i | seg_val(i) = 7 });

VFUN seg_read(nat_number loc) -> INTEGER data;
EXCEPTIONS
out_of_bounds: NOT(size() > loc);
RELATION
seg_val(loc);

OFUN rewrite(nat_number loc; INTEGER data);
EXCEPTIONS
out_of_bounds: NOT(size() > loc);
EFFECTS
'seg_val(loc) = data;

OFUN append(INTEGER data);
EXCEPTIONS
out_of_room: size() = max_size;
EFFECTS
'seg_val(size()) = data;

OFUN shrink();
EXCEPTIONS
empty: size() = 0;
EFFECTS
'seg_val(size() - 1) = 7;

END_MODULE

```

```

MODULE pages

TYPES

nat_number: { INTEGER j | j >= 0 };

PARAMETERS
INTEGER max_pages, page_size;

FUNCTIONS

VFUN page_val(nat_number k, loc) -> INTEGER data;
HIDDEN;
INITIALLY
data = 0;

VFUN number_of_pages() -> nat_number n;
HIDDEN;
INITIALLY
n = 0;

VFUN size_last_page() -> nat_number n;
HIDDEN;
INITIALLY
n = 0;

```

```

VFUN page_read(nat_number k, loc) -> INTEGER data;
EXCEPTIONS
  no_page: NOT(number_of_pages() > k);
  no_loc: IF k < number_of_pages()
    THEN loc >= page_size
    ELSE loc >= size_last_page();
DERIVATION
  page_val(k, loc);

OFUN page_write(nat_number k, loc; INTEGER data);
EXCEPTIONS
  no_page: NOT(number_of_pages() > k);
  no_loc: IF k < number_of_pages()
    THEN loc >= page_size
    ELSE loc >= size_last_page();
EFFECTS
  'page_val(k, loc) = data;

OFUN new_page();
EXCEPTIONS
  no_more_pages: number_of_pages() = max_pages;
EFFECTS
  'number_of_pages() = number_of_pages() + 1;

OFUN resize_last_page(nat_number new_length);
EXCEPTIONS
  no_last_page: number_of_pages() = 0;
  full: new_length > page_size;
EFFECTS
  'size_last_page() = new_length;
  FORALL nat_number n | n
    INSET { new_length - 1 ..
size_last_page() - 1 } :
    'page_val(number_of_pages(), n) = 0;

OFUN deallocate_last_page();
EXCEPTIONS
  no_last_page: number_of_pages() = 0;
EFFECTS
  'number_of_pages() = number_of_pages() - 1;
  'size_last_page() = page_size;
  FORALL nat_number n: 'page_val(number_of_pages(), n) = 0;

END_MODULE

```

MAP segment TO pages;

TYPES

```
nat_number: { INTEGER n | n >= 0 };
```

EXTERNALREFS

```
FROM segment:
INTEGER max_size;
VFUN seg_val(nat_number n) -> INTEGER data;
```

```
FROM pages:
INTEGER max_pages, page_size;
VFUN page_val(nat_number k, loc) -> INTEGER data;
VFUN number_of_pages() -> nat_number n;
VFUN size_last_page() -> nat_number n;
```

MAPPINGS

```
max_size: max_pages * page_size;
```

```
seg_val(nat_number loc):
  IF loc
    <= page_size *(number_of_pages() - 1) + size_last_page() - 1
    THEN page_val(INTPART(loc / page_size),
      FRACTPART(loc / page_size))
    ELSE ?;
```

END_MAP

A REQUIREMENT STATEMENT FOR A SYSTEM
(Not Adequately Expressible in Special)

An abstract statement of what the system does. Generally, a requirement expresses a subset of the information contained in the specification and requires

- * Expression of "information flow"
- * Expression of the effect of sequences of operations
- * Second order logic

The top-level specification of a system can ..

TH principle -- be verified with respect to its requirement

REQUIREMENT STATEMENT FOR MULTI-LEVEL SECURITY

Level = \langle Classification, Category_set \rangle

Classification is an element of a totally ordered set

For two levels

L1 = \langle CL1, CAT 1 \rangle

L2 = \langle CL2, CAT 2 \rangle

L1 \succeq L2

=

CL1 \succeq CL2

and

CAT 1 \succeq CAT 2

Example

Classifications:

Unclassified, Confidential, Secret, Top Secret

Categories

Atomic, Nato

Information can flow from L2 to L1

If and only if $L1 \geq L2$

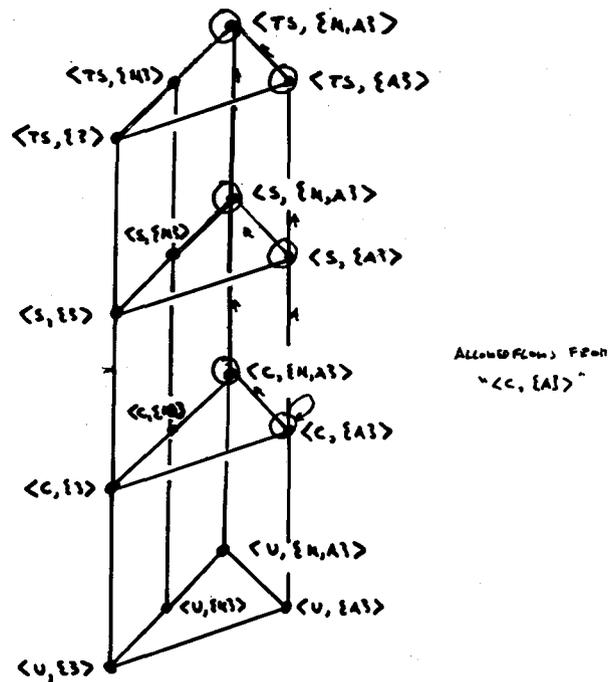
This model is flawed since:

- All information will eventually reach the highest security level
- Information at a high security level can be "destroyed" by low security level information

Nevertheless --

This model is widely used as the basis for secure systems, e.g., KSOS

MLS LATTICE



Proving that a top-level special specification is multi-level secure is conceptually very easy

- * Write the specifications such that a security level is associated with each data structure (V. function).
- * Show that according to the specs, the new value of a data structure at Level L is dependent only on the old values of data structures at L_i , $L_i \leq L$.

The identification of dependencies is complicated by the "syntactic sugar" and "real-world" features of special -- but very doable.

```
MODULE virtual_memory
  PARAMETERS
    INTEGER max_seg_no, max_seg_index;

  EXTERNALREFS
    FROM security:
    security_level: DESIGNATOR;
    BOOLEAN lteq(security_level l1, l2);

  FUNCTIONS
    VFUN contents(INTEGER segno, index; security_level sl)
      -> INTEGER c;
    HIDDEN;
    INITIALLY
      c = ?;

    VFUN read(INTEGER segno, index; security_level sl)
      [security_level pl]
      -> INTEGER c;
    EXCEPTIONS
      segno < 0 OR segno > max_seg_no;
      contents(segno, index, sl) = "?";
      ~lteq(sl, pl);
    DERIVATION
      contents(segno, index, sl);

    OFUN write(INTEGER segno, index, c; security_level sl)
      [security_level pl];
    EXCEPTIONS
      segno < 0 OR segno > max_seg_no;
      index < 0 OR index > max_seg_index;
      ~lteq(pl, sl);
    EFFECTS
      'contents(segno, index, sl) = c;
      FORALL INTEGER i | i >= 0 AND i < index
        AND contents(segno, i, sl) = ?;
      'contents(segno, i, sl) = 0;

  END_MODULE
```

HDM Tools:

1. Specifications checkers (completed)
2. Multilevel security verifier (completed)
3. Modula verification system (completed)
4. Pascal verification system (in progress)

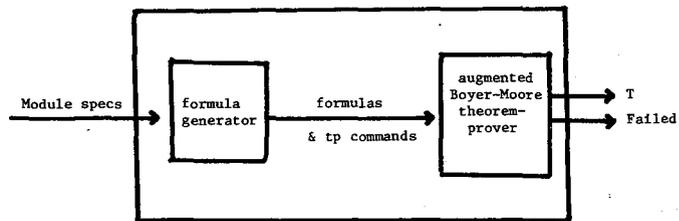
All in Interlisp and available for public use.

1. Spec checkers

- * Usual parsers, type checkers, and pretty-printers for Special, HSL, & ILPL.
- * Various external consistency criteria also checked.
- * Limited in scope, but heavily used.
- * Support small amount of version control.

2. Multilevel security verifier

Basic multilevel security property: whenever information flows from one entity to another, the security level of the recipient is at least as high as the sender.



Multilevel Security Verifier
Crich Feiertag -- Now at Sytek)

Validity of generated formulas \supset multilevel security.

Tool is conservative, i.e., may not be able to demonstrate mls for some secure specs, but never the other way.

Has been used extensively by SRI & non-SRI people, and has exposed many previously unknown security violations.

Theorem-proving is completely automatic.

Formulas to be proved usually easy but numerous.

3. MODULA VERIFICATION ENVIRONMENT

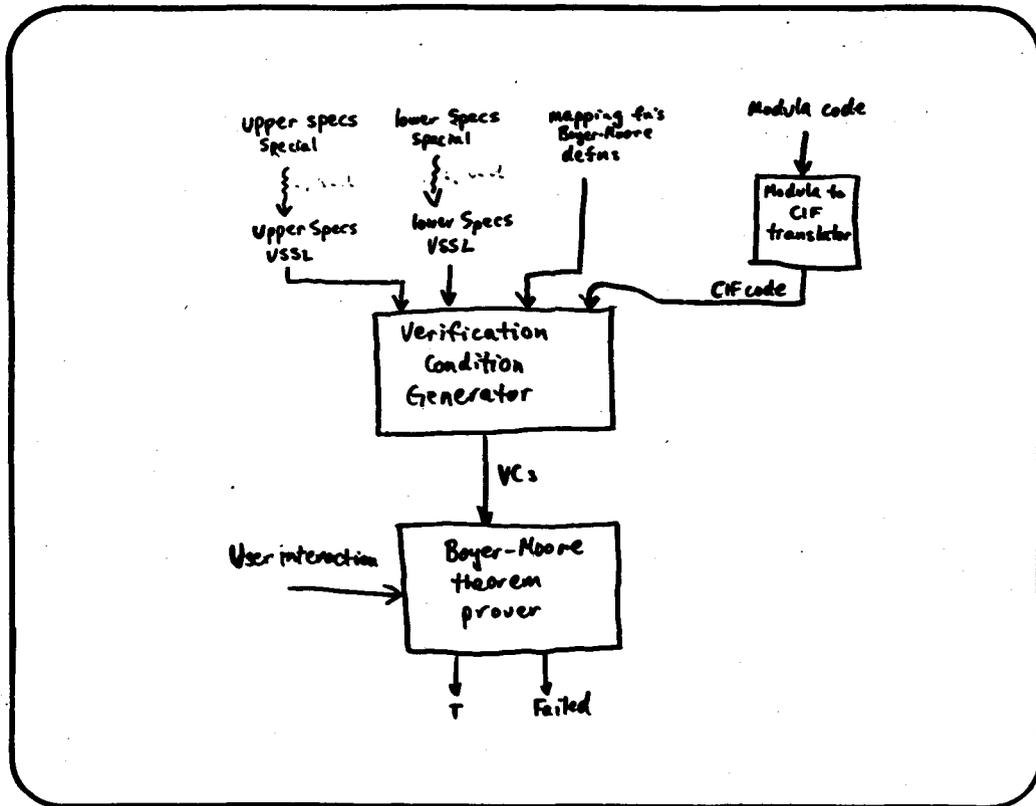
Developed for verification of Modula code in Ford Aerospace's KSOS implementation.

Based directly on the Boyer-Moore formalization of HDM.

- * Specs are written in Special variant VSSL (a.k.a. "the formalized subset").

USSL is a cleaned-up, formally-defined version of Special. Assertion level consists of expressions in B-M theory. Concrete syntax is Lisp-like, internal-form like (e.g., more like a linearized abstract syntax).

- * Implementation language supported by the B-M formalization is the assembly-like language (CIF (like ILPL))

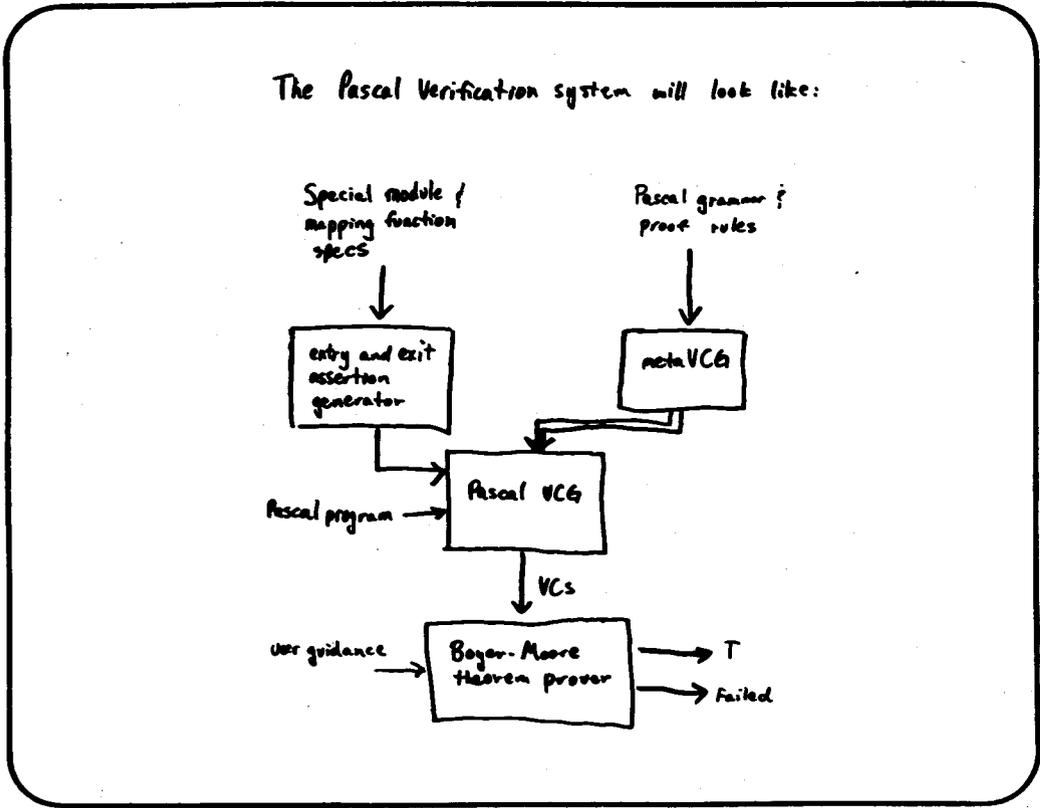
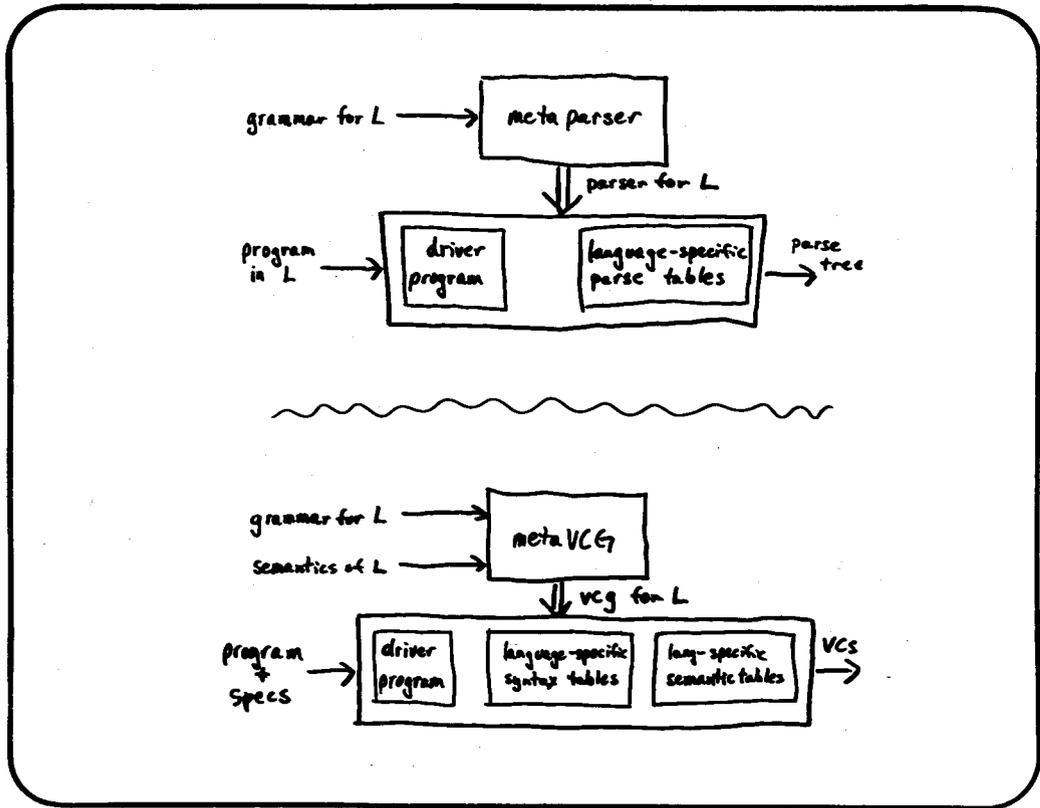


4. Pascal Verification System

Currently under development for proof of SIFT.
Deals directly with Special specs and Pascal code
(not USSL & CIF).

A novel component is the meta-verification condition generator (meta VCG).

The meta VCG processes formal semantic descriptions along the lines of the way meta-parsers (i.e., parser generators) process formal syntax descriptions.



EXPERIENCE WITH SPECIAL:

While well-conceived, Special has flaws

- * The concrete syntax is too often awkward and unpredictable (the syntax-checker gets used a lot!)
- * The provided language structures CO-, V-, and OV-functions) do not correspond directly with the structures of the underlying model (state-fn's & op's). A great source of confusion.
- * Some constructs contain "dark corners" with semantics not easily deducible from principles.
- * Other constructs not as general as they could (or should) be.
- * The type system should be integrated better, with more modern abstract data type facility.

NEVERTHELESS ---

- * Special has been (and is being) used extensively and productively in the design of numerous systems.
- * While module specification in Special is harder than it should be, it's not hard to learn one's way "around" the problems. Effort is well worth it.
- * The "formalized subset" is extremely clean and does not suffer from these flaws.

WORK IS IN PROGRESS TO DEVELOP A SUCCESSOR TO SPECIAL.

AN ASSESSMENT OF HDM:

HDM has been most successful as a design tool.

Most users see HDM as just Special. That is, they use HDM primarily for Module design & specification. The specs capture design decisions and serve as a reference for discussing alternatives.

Users are typically interested in verifying some properties of the design, so appreciate the rigor Special provides.

Current HDM activity is creating a "second generation" methodology, taking into consideration user experiences and recent research, particularly in the area of data type specification.

Our appraisal has affirmed the appropriateness of the state-machine approach to specification -- most often, it is easier to use than the algebraic approach, especially when:

- new concepts are being specified
(i.e., things other than stacks, queues, etc.)

especially if those concepts are more "process" oriented, as opposed to data oriented.

- difficult "real-world" features must be specified, including side-effects, aliasing, etc.

People tend to think in models.

GOALS

1. Routine, production-quality proofs of
 - a. Program correctness wrt specifications
 - b. Specification properties
2. Strong alternative & complement to program testing
3. Stimulus to mathematical basis for
 - a. software reliability
 - b. programming methodology
4. Ultimately, certification use in highly critical software
 - a. Nuclear Reactors
 - b. Avionics Systems
 - c. Secure Systems
 - i. Electronic Funds Transfer
 - ii. Operating
 - iii. Military
 - d. Protocols in Message/Data Systems
 - i. Electronic Mail
 - ii. Electronic Funds Transfer
 - iii. Distributed Systems

AFFIRM's PARADIGM

1. Abstract Data Types
 - a. Algebraic specification (Guttag)
 - b. User-defined in programs
2. Inductive Assertion Method for Programs
 - a. Assertions on Loops, Entry/Exit
 - b. Turn programs into Verification Conditions
3. Interactive Theorem Prover/Checker
 - a. User gives strategy and directions
 - b. System does book-keeping, formula manipulation

ALGEBRAIC SPECIFICATIONS

1. Abstract Data Types

- a. Set
- b. Queue
- c. BinaryTree
- d. Sequence

2. Parts of a Type

- a. interfaces of operations, strongly typed
- b. axioms defining operations
- c. schema for induction on the type

3. Operations are

- a. Constructors - Other operations defined over constructors
- b. Extenders (Modifiers)
- c. Selectors
- d. Predicates

type *QueueOfElemType*;

declare *q, q1, q2, qq*: *QueueOfElemType*;

declare *i, i1, i2, it*: *ElemType*;

interfaces *Constructors* *NewQueueOfElemType, q Add i*,
Extenders *Remove(q), Append(q1, q2), que(i)*
: *QueueOfElemType*;

interfaces *Selectors* *Front(q), Back(q)*: *ElemType*;

interfaces *Induction(q)*,
Predicate i in q: *Boolean*;

axioms *Axioms for Equality*

q = q == TRUE,
q Add i = NewQueueOfElemType == FALSE,
NewQueueOfElemType = q Add i == FALSE,
q1 Add i1 = q2 Add i2 == ((q1=q2) and (i1=i2));

axioms *Remove(NewQueueOfElemType) == NewQueueOfElemType.*
Remove(q Add i) == if q = NewQueueOfElemType
then q
else Remove(q) Add i;

axioms *Append(q, NewQueueOfElemType) == q,*
Append(q, q1 Add i1) == Append(q, q1) Add i1;

axiom *que(i) == NewQueueOfElemType Add i;*

axiom *Front(q Add i) == if q = NewQueueOfElemType*
then i
else Front(q);

axiom Back(q Add i) == i;

axioms i in NewQueueOfElemType == FALSE,
i in (q Add i1) == (i in q or (i=i1));

schema Induction(q)
== cases(Prop(NewQueueOfElemType),
all qq, ii (IH(qq) imp Prop(qq Add ii)));
end {QueueOfElemType} ;

THEOREM PROVER: MECHANICAL

- Rewrite Rule Orientation

* Axioms $lhs = rhs$ become rules $lhs \rightarrow rhs$

* Properties of good rules:

- Finite Termination
- Unique Termination -- Knuth-Bendix algorithm
- Sufficient completeness

- Natural Logic

* Combine with conditional expressions for logic

b and c \rightarrow if b then c else FALSE

b imp c \rightarrow if b then c else TRUE

if (if b then c else d) then e else f \rightarrow
if b then (if c then e else f)
else (if d then e else f)

* Also quantifiers *some* and *forall*

- Recursive function definitions (an escape mechanism from otherwise infinite rewrite rules)

Examples

Notation

```
define splitat(q,i)== i in q imp  
    some q1,q2 (q=Append(q1 Add i, q2));
```

Recursive Function

```
define MakeQueue(e,n)==  
    if n<=0 then NewQueueOfElemType  
    else MakeQueue(e, n-1) Add e;
```

THEOREM PROVER: HUMAN

Proof Structure

- * Nodes: propositions
- * Arcs: names of subgoals

- * Movement around tree via cursor
 - up, down - to retrace steps
 - retry, resume - current theorem
 - next - to "natural" successor
 - named node or arc

Name, annotate, print status and theorems

EXAMPLE PROOF TREE

*QueueSplit is: not (q = NewQueueOfElemType)
 imp Append(que(Front(q)), Remove(q)) = q

proof tree:

4: QueueSplit
 employ Induction(q)
 NewQueueOfElemType:
 Immediate
6: Add:
 2 cases
8: 3 invoke first IH
10: 4 replace qq'
10:-> (proven)

Proof Commands

<i>try prop</i>	sets up a goal
<i>apply prop</i>	use prop as a lemma
<i>invoke def</i>	invoke a definition
<i>employ Induction(v)</i>	use a schema
<i>suppose prop</i>	divide with prop and \sim prop
<i>replace</i>	use equalities

USER HABITABILITY

Proving is hard - the system should help, not hinder

User Interface Features

1. Spelling correction
2. User profile
3. Command abort, fix, undo, redo
4. Recursive Execs

System Interfaces

1. Transcript of sessions
2. Output through formatter to variable font device
3. Automatic loading of needed types
4. Easy access to editors

EXAMPLES

Data Types

Queue
Set
Sequence жжжж
Circle
Binary Tree
Array

Small Examples

Interpolation Search
Root Finding (numerical analysis)

Large Examples

Delta - 1000 line BLISS module for file updating
Fully specified
Partially proved

Communication Protocols
Alternating Bit
3 way handshake (TCP)

Specification

Toy Security Kernel

EXPERIENCE

Easy to learn, knowing literature and logic
Several "external" users (protocols)

Error-prone users
Using commands
Getting lost
Stating theorems and lemmas wrong

Proofs are
Simple, well-structured at end
Messy, long in middle
Crudely planned at start
Easier to find than theorems ****

Paradigm good
Proving must be interactive

Rewriting rules are effective, natural

Data abstraction methodology
Now widely known
Extendible - transition systems

User interface is critical to productivity

Resource demands are bottle neck -

CURRENT STATE OF AFFIRM

Released for wider use over ArpaNet
December 1979

REFERENCE LIBRARY

Reference Manual
User's guide
Type Library
Annotated Transcripts
Collected Papers

PROTOTYPE FOR EXPERIMENTATION

Variety of users
Variety of applications

CONTINUED EVOLUTION

More theory of rewrite rules
Better interface, display capabilities
Integration with testing
Methodology for errors, exceptions
Support for proof persistence
Larger, stable library

An Overview of Software Testing

Mary Jo Reece
MITRE Corporation

Outline

What is Software Testing?
Why is Software Testing Important?
**Where does Software Testing Fit into the
Software Life Cycle?**
How is Software Testing Conducted?
Summary

What is Software Testing?

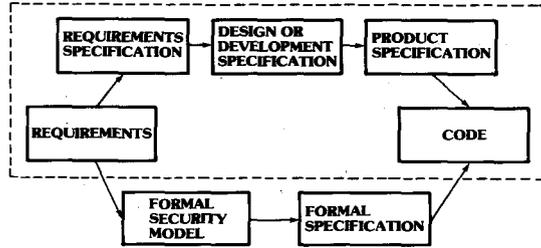
Why is Software Testing Important?

Why is Software Testing Important?

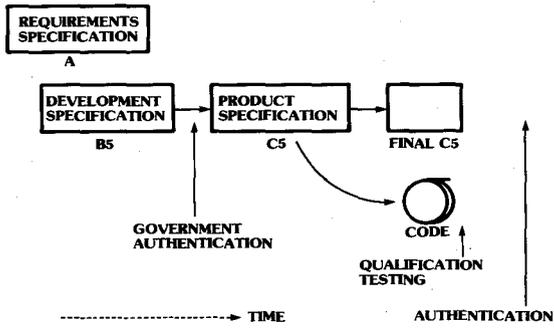
	Software Effort		
	Analysis & Design	Coding & Auditing	Test
SAGE	39%	14%	47%
Gemini	36%	17%	47%
O/S 360	33%	17%	50%

Where does Software Testing Fit into the Software Life Cycle?

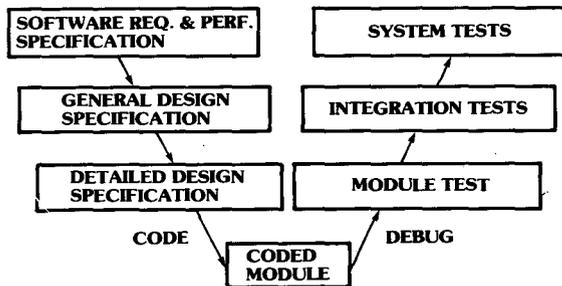
Software Development Approaches



Where does Software Testing Fit Into the Software Life Cycle?



Relationship of Development to Test Activities



**How is Software Testing
Conducted?**

**How is Software Testing
Conducted?**

Overall Software Testing Activity

- Test planning
- Test case design
- Test execution
- Evaluation of test results

**How is Software Testing
Conducted?**

Test Case Design

- Test plans
- Test procedures
- Test reports

How is Software Testing Conducted?

Software Testing Approaches

Module tests

What are they?

Why start at this level?

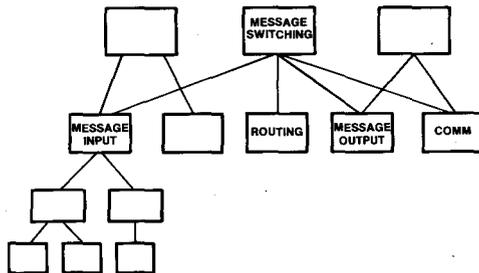
Integration tests

What do they do?

How do they differ from module testing?

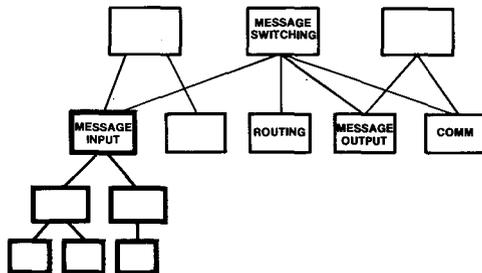
How is Software Testing Conducted?

Explicit vs. Implicit Testing



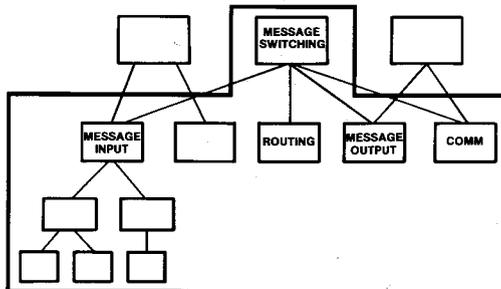
How is Software Testing Conducted?

Explicit vs. Implicit Testing



How is Software Testing Conducted?

Explicit vs. Implicit Testing



How is Software Testing Conducted?

Implicit Testing

Exercising software without knowledge of structure

Based entirely on external inputs

Cannot control software variables

Difficult to isolate source of any failures

Requires entire software structure

Summary

Summary

Testing does not introduce quality into the product per se — it only provides a measure of the existing quality level and may identify the extent and location of the defects.

**UPDATE ON THE
KERNELIZED SECURE OPERATING SYSTEM
(KSOS)**

John Nagle

UPDATE ON KSOS - OVERVIEW

- Project goals and their realization
- Problems along the way
- Insights into trusted computing

FLASH!

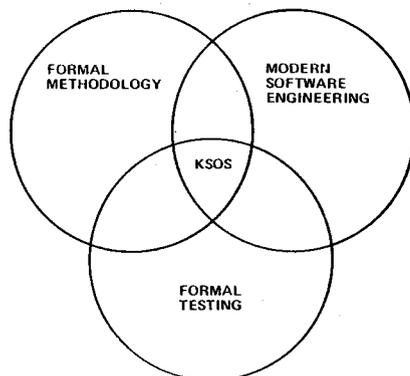
**SHIPPED TO ALPHA TEST SITE
ON 11 SEP 80!**

PROJECT GOALS

KSOS REQUIREMENTS SUMMARY

- **Provable security: based on security Kernel and trusted processes**
- **UNIX compatibility**
- **Efficiency comparable with UNIX**
- **Administrative support features**
- **General purpose Kernel**
 - **Multiple machines**
 - **Emulators for other operating systems**
 - **Non-UNIX applications**

KSOS SECURITY ASSURANCE



UNIX COMPATIBILITY

- Functional compatibility - very close to UNIX
- Performance of Alpha release 4x to 8x UNIX
 - Costs of security, mostly structural
 - Overhead of kernel/emulator structure
 - UNIX maturity
 - Reduced possibilities for global optimization

BROAD APPLICABILITY

- Support turn-key operation
 - Need for trusted support tools
 - Reduce known vulnerabilities requiring a "GURU" for repair
 - Eliminate "Super-User" by providing encapsulated utilities
- Reduce UNIX specific aspects of kernel
 - Flat file system
 - Rich inter-process communications
 - General process creation support

PROBLEMS ALONG THE WAY

MAINTAINING CONSISTENCY

Problem

- How to maintain consistency between the multiple independent representations of a system component

Solution

- Extensive use of on-line configuration management tools
- Management discipline prudently applied
- Independent test team
- Formal testing

MULTIPLE LANGUAGE SUPPORT

Problem

- Seven different languages used for various aspects of the project. All required modification and support.

Solution

- Hire multi-lingual staff
- Encourage ADA
- Need more research in integrated software development environments

MODULA

Problem

- Significant re-work of compiler was required.

Solution

- ADA?

MODULA (Continued)

Problem

- Strongly modular languages discourage highly efficient structures, or incur substantial overhead

Solution

- Additional research in compilers
- Better machine support

BENEFITS OF MODULA

- Strong typing
- Language-generic multi-programming
- Enforced modularity

FORMAL TOOLS

Problem

- Limitations of existing formal specification languages

Solution

- More research, particularly in integrated environments

MATHEMATICAL MODEL LIMITATIONS

Problem

- Bell-Lapadula model too restricted

Solution

- Research in models of security

INSIGHTS INTO TRUSTED COMPUTING

HINDSIGHT - THINGS THAT WORKED

- Success of disciplined methodology
- Value of formal specifications for unexpected purposes
- Integrated development environment worked well
- Personnel accepted formal methods easily
- Although occasionally annoying, MIL-SPEC documentation was useful
- Having a model to work against very helpful

HINDSIGHT - WHAT MIGHT HAVE BEEN DONE BETTER

- Better integration of segment and file systems
- More insight into consistency between multiple representations
- Better implementation language
- Simpler secure path mechanism
- Alternate Emulator structure

INSIGHTS INTO TCB DESIGN AND IMPLEMENTATION

- It can be done!
- Need for consistency between different languages, care in their use
- Utility and benefits of formal specifications
- Code proofs are not yet practical except for demonstrations. However, being ready to do them is of great benefit.
- Need for additional tools and concepts

ASSURANCE PRACTICES IN KVM/370

**MARVIN SCHAEFER
SYSTEM DEVELOPMENT CORPORATION
SANTA MONICA, CALIFORNIA**

ASSURANCES FOR ACCREDITATION

- **SECURITY EVIDENCE**
 - **HARDWARE ADEQUACY**
 - **SECURITY ANALYSIS**
 - **FORMAL SPECIFICATION/VERIFICATION**
 - **SOFTWARE ENGINEERING PRACTICES**
 - **TESTING METHODOLOGY**
 - **DOCUMENTATION**
 - **PEER REVIEW**

PREHISTORY OF THE CONCEPT

- **REFERENCE MONITOR DEFINED**
 - **ANDERSON, ET AL**
- **VIRTUAL MACHINE MONITOR STUDIES**
 - **POPEK, WEISSMAN, BELADY**
- **VM/370 IMPLEMENTED**
 - **REFERENCE MONITOR IS EMULATOR**
 - **CP IS CP/67 ON BETTER HARDWARE**
 - **3 STATES FROM 2**
 - **SEPARATE ADDRESS SPACES**
 - **SMALL, SIMPLE, CONSISTENT**
 - **EVEN IMPLEMENTS S/370 SECURITY FLAWS**

EARLY HISTORY

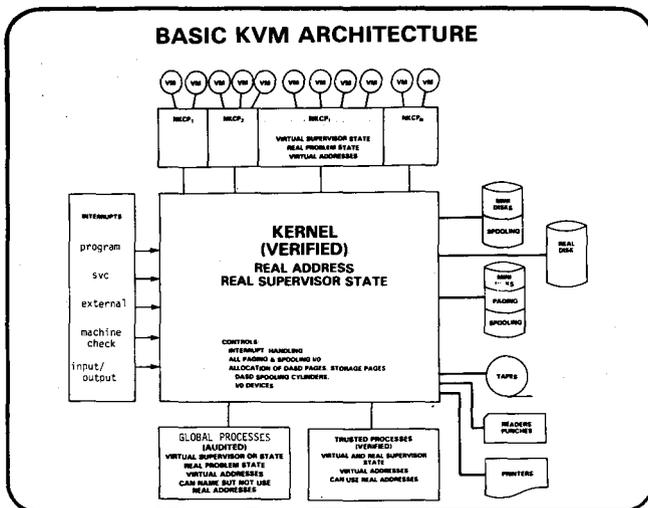
- **PENETRATION STUDY — SDC/IBM**
- **"HARDENING" EFFORTS**
 - **YORKTOWN HEIGHTS**
 - **APARS DEMANDED**
 - **OTHER PROPOSALS**

KVM SECURITY RETROFIT

- **MINIMAL REWRITE OF CODE**
- **VERIFIABILITY ALL THE WAY TO THE CODE**
 - **"PARNAS" SPECIFICATION**
 - **FORMAL VERIFIED SPECIFICATION**
 - **EXTENDED SECURITY ANALYSES**
 - **EUCLID, VERIFIED IMPLEMENTATION**
- **CONTINUING PEER REVIEW**
 - **ARPA KVM REVIEW COMMITTEE**
 - **ARPA SECURITY WORKING GROUP**
 - **IBM**
- **PERFORMANCE AND MEASUREMENT GOALS**

ARCHITECTURAL INFLUENCES

- **KERNELIZED DESIGN**
 - **UCLA SECURE UNIX™**
 - **MITRE 11/45 KERNEL**
 - **AFDSC SECURE MULTICS**
 - **MIT PROJECT GUARDIAN**
 - **SRI PSOS**
- **HIERARCHICAL DATA TYPE MONITORS**
 - **HOARE, BRINCH HANSON**
 - **JANSON, REED**



CONTROVERSIES AND CONUNDRUMS

- **TRUSTED PROCESS**
 - POLICY ENFORCEMENT IN KERNEL
 - POLICY INTERPRETATION IN TRUSTED PROCESSES
- **TRUSTED PROCESSOR AND PERIPHERALS**
 - CONTROL UNITS
 - UNTRUSTED DEVICES
- **CONFINEMENT AND SCHEDULERS**
 - WHAT COUNTS FOR CORRECTNESS?
 - WHAT CAN BE VIRTUALIZED?
 - SECURITY/PERFORMANCE TRADEOFFS

ABSTRACT SPECIFICATIONS

- **IMPRECISE INFORMAL SPECS**
 - "PARNAS" FORMAT
 - ENGLISH AND PSEUDO CODE STRUCTURE
- **IMPRECISE FORMAL SPECIFICATIONS**
 - TIMELESSNESS
 - NON-PROCEDURAL PROCEDURALITY
 - TIMEOUTS
 - CAPABILITY FAULTS
 - CONTEXT RESTORATION
 - ABEND

SEARCH FOR SUITABLE, VERIFIABLE HOL

- EUCLID'S DEMISE
- PASCAL'S INEFFICIENCIES AND DATA-STRUCTURE INADEQUACIES
- PL/I'S SUPPORT PACKAGE
- FREGE'S KARMA
- JOVIAL COMPROMISE

CODING FROM FORMAL SPECIFICATIONS

- INFINITE SETS BECOME FINITE TABLES
CONSIDERATIONS
 - HOW FINITE?
 - HOW SPARSE?
 - HOW ACCESSED?
 - HOW FREQUENTLY?
- FAITHFUL IMPLEMENTATION OF \exists , \forall
CONSIDERATION
 - IS IT A SPECIFICATION "FICTION"
- LEGALITY-CHECKING

REVISED SPECIFICATION/VERIFICATION THRUST

- ORIGINAL SPECS
 - COMPLETED 1978
 - NEVER VERIFIED
- REVISED, VERIFIED TOP LEVEL SPECS (1980)
 - DERIVED FROM CODE, IMPLEMENTORS
 - ARCHITECTURAL MODIFICATION/SIMPLIFICATION
- SECOND-LEVEL SPECIFICATIONS (1980)
 - CORRELATION REVIEWED WITH IMPLEMENTORS
 - MAPPINGS COMPLETED BETWEEN LEVELS

CODING

- TWO PARALLEL EFFORTS
 - NKCP-KERNEL INTERFACE MACROS
 - NKCP MODS PERFORMED VIA CMS EXECS
 - KERNEL IMPLEMENTED BOTTOM-UP
 - SUB KERNEL
 - TRUSTED PROCESSES STUBBED

TESTING

- KVM DEVELOPED & TESTED UNDER VM/370
 - HEAVY USE OF CMS- AND CP- TEST ENVIRONMENTS
 - ADSTOP
 - PER TRACE
 - MACHINE CONFIGURATION
- KERNEL "UNIT" TESTING
 - DRIVER IS PSEUDO NKCP
 - GROWN OVER TIME
 - SELECTABLE KERNEL CALL TEST CASES
 - LEGAL AS ILLEGAL PARAMETERS USED
- KERNEL "INTEGRATION" TESTING
 - DRIVER IS NKCP OR NKCPs
 - VMs USED TO DRIVE NKCPs

TESTING SYNCHRONY AND ASYNCHRONY

- STRICT SYNCHRONY
 - KERNEL, 1 NKCP, 1 VM
- A SYNCHRONOUS NKCP
 - KERNEL, 1 NKCP, 2 VMs
- ASYNCHRONOUS KERNEL
 - KERNEL, 2 NKCPs, 1 VM EACH
- TOTAL ASYNCHRONY
 - KERNEL, 2 NKCPs, 2 VMs PER NKCP

FIELD TEST

- **INITIAL TESTS TO BEGIN JANUARY 1981**
 - SDC IBM 4331
 - ARMY ITEL AS/5
 - NAVY AMDAHL V/7
 - AIR FORCE IBM 3031/4341
- **PERFORMANCE MEASUREMENT AND TUNING**
- **FUNCTIONALITY TESTING**
- **SECURITY INTERFACE EVALUATION/FEEDBACK**
- **SECURITY TESTING**

IN RETROSPECT

- **INSUFFICIENT DETAILED DOCUMENTATION**
- **JOVIAL WAS NOT OPTIMAL CHOICE**
 - NOT MAINTAINED VM/370 COMPILER
 - ORIGINAL KVM CONVENTIONS EXCEEDED MANY COMPILER CAPABILITIES
 - LACK OF MODERN LANGUAGE FEATURES
- **PEER REVIEW SHOULD BE FREQUENT**
- **WAS RETROFIT SUCH A GOOD IDEA?**
- **STAFF SIZE SHOULD HAVE BEEN INCREASED EARLIER**
- **STAFF SHOULD HAVE HAD ACCESS TO A LOCAL COMPUTER**

KERNELIZED SECURE OPERATING SYSTEM

(KSOS-6)

CHARLES H. BONNEAU
HONEYWELL

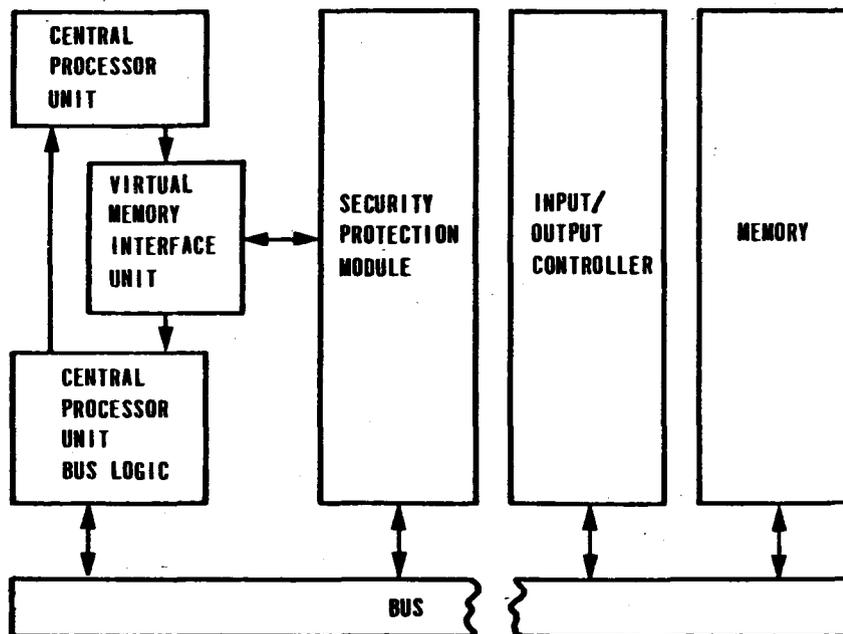
TOPICS

- PROJECT OBJECTIVES
- HARDWARE DESIGN OVERVIEW
- SOFTWARE DESIGN OVERVIEW
- ASSURANCE TECHNIQUES

PROJECT OBJECTIVES

- DEVELOP ADD-ON HARDWARE TO COMMERCIAL MACHINE WHICH MAKES IT EASIER TO BUILD SECURE SYSTEMS
- DEVELOP TCB SOFTWARE
 - ENFORCE DoD SECURITY POLICY
 - FORMALLY PROVABLE
 - SUPPORT UNIX + OTHER APPLICATIONS

SPM + LEVEL 6 MINICOMPUTER = SCOMP



SECURITY PROTECTION MODULE FEATURES

- FAST PROCESS SWITCHING
 - PROCESS DESCRIPTOR TREE DEFINITION VIA DESCRIPTOR BASE ROOT
 - AUTO LOAD OF DESCRIPTORS
- NO CPU TO MEMORY MEDIATION OVERHEAD AFTER INITIAL ACCESS
- 1-3 LEVEL MEMORY DESCRIPTOR SYSTEM
 - R, W, E CONTROL AT ANY LEVEL
 - SEGMENTS: 2K WORDS (512)
 - PAGES: 128 WORDS
- I/O MEDIATION
 - CPU TO DEVICE
 - DEVICE TO MEMORY
- MULTICS-LIKE RING STRUCTURE
 - 2 PRIVILEGED, 2 NON-PRIVILEGED RINGS
 - READ, WRITE, EXECUTE, AND CALL BRACKETS
 - RING CROSSING SUPPORT INSTRUCTIONS
- PAGE FAULT RECOVERY SUPPORT

KSOS-6 SOFTWARE

- SECURITY KERNEL
- TRUSTED SOFTWARE
- UNIX EMULATOR

KERNEL DESIGN OVERVIEW

- NON-FILESYSTEM IO OUTSIDE KERNEL
- FILES CONSTRUCTED EXTERNALLY USING SEGMENTS
- DEMAND PAGING VIRTUAL MEMORY
- NON-DISCRETIONARY ACCESS CONTROL - BELL AND LAPADULA
 - PRIVILEGE
 - ACCESS ATTRIBUTES NOT FIXED
- DISCRETIONARY ACCESS CONTROL
 - UNIX R, W, E FOR OWNER, GROUP, OTHER
 - RING BRACKETS FOR OWNER, GROUP, OTHER
 - SUBTYPES
- KERNEL OBJECTS
 - PROCESSES
 - SEGMENTS
 - DEVICES

KSOS-6 ARCHITECTURE

- ADDRESS SPACE PARTITIONING
 - MEMORY
 - SEG 0-95: DISTRIBUTED KERNEL
 - SEG 96-127: LOCAL KERNEL
 - SEG 128-511: USER
 - DEVICE
 - DEV 0-31: DISTRIBUTED KERNEL
 - DEV 32-511: USER
- RING STRUCTURE
 - RING 0 : KERNEL
 - RING 2 : UNIX EMULATOR
TRUSTED S/W
 - RING 3 : USER APPLICATIONS

VISIBLE FUNCTIONS

PROCESSES

- CREATE_PROCESS
- INVOKE_PROCESS
- RELEASE_PROCESS
- GET_PROCESS_ACCESS
- SET_PROCESS_ACCESS
- GET_PROCESS_STATUS
- SET_PROCESS_STATUS
- SET_PROCESS_SUBTYPES
- RECEIVE_MESSAGE
- SEND_MESSAGE
- INTERRUPT_RETURN
- GET_SYSTEM_PARAMETERS
- SHUTDOWN

SEGMENTS

- CREATE_SEGMENT
- DELETE_SEGMENT
- GET_SEGMENT_ACCESS
- SET_SEGMENT_ACCESS
- GET_SEGMENT_STATUS
- SET_SEGMENT_STATUS
- MAP_SEGMENT
- UNMAP_SEGMENT
- WIRE_SEGMENT
- UNWIRE_SEGMENT
- SYNC_SEGMENT

DEVICES

- CREATE_DEVICE
- REMOVE_DEVICE
- GET_DEVICE_ACCESS
- SET_DEVICE_ACCESS
- GET_DEVICE_STATUS
- SET_DEVICE_STATUS
- MAP_DEVICE
- UNMAP_DEVICE
- SECURE_TERMINAL_LOCK
- SECURE_TERMINAL_UNLOCK
- MOUNT
- UNMOUNT
- READ_SYSTEM_CLOCK
- SET_SYSTEM_CLOCK

TRUSTED SOFTWARE

● OPERATIONS SERVICES

- SECURE STARTUP
- OPERATOR INTERFACE
- SECURE LOADER(S)
- SHUTDOWN

● USER SERVICES

- SECURE INITIATOR
- SECURE SERVER
- LOGIN
- SET USER ACCESS LEVEL
- SET FILE ACCESS LEVEL
- LOGOUT

TRUSTED SOFTWARE (CONT)

- MAINTENANCE SERVICES

- MAKE FILE SYSTEM
- SEGMENT DUMP
- SAVE/RESTORE FILESYSTEM
- FILESYSTEM CONSISTENCY CHECK

ASSURANCE TECHNIQUES

- HARDWARE

- DESIGN VERIFICATION
 - TESTING USED TO VERIFY DESIGN
 - ANALYSIS USED TO VERIFY COMPLETENESS OF TESTING
- FAILURE INDUCED SECURITY COMPROMISE
 - ESTABLISH PROBABILITIES THAT FAILURE WILL RESULT IN COMPROMISE
 - IDENTIFY FUNCTIONS THAT REQUIRE RUNNING PERIODIC "HEALTH CHECKS"
- HARDWARE "GATES" INCLUDED IN FORMAL TOP-LEVEL SPEC

ASSURANCE TECHNIQUES

- SOFTWARE
 - SPECIFICATIONS
 - FORMAL TOP-LEVEL SPEC
 - B5 DESIGN SPEC
 - C5 DESIGN SPEC
 - IMPLEMENTATION
 - VERIFIABLE LANGUAGE - UCLA PASCAL
 - 10K SOURCE LINES
 - DESIGN REVIEWS
 - INFORMAL VERIFICATION BY CORRESPONDENCE THROUGH IMPLEMENTATION
 - FORMAL VERIFICATION OF SYSTEM DESIGN
 - SRI HIERARCHICAL DEVELOPMENT METHODOLOGY (HDM)
 - ILLUSTRATIVE PROOF OF IMPLEMENTATION
 - TEST --- TEST

KERNEL VERIFICATION STATUS/RESULTS

- PROOF OF DESIGN ALMOST COMPLETE
 - 1 MODULE REMAINS
- FALSE THEOREMS EXIST
 - RESOURCE EXHAUSTION
 - TRANQUILITY PRINCIPLE VIOLATIONS
 - EXCEPTION REPORTING ON WRITE-UPS
- DIFFERENCES FROM IMPLEMENTATION
 - PRIVILEGE IS REMOVED
- TOOLS
 - IMPROVED
 - ISOLATING REASONS FOR FALSE THEOREMS IS TEDIOUS

SCOMP TLS

<u>LEVEL</u>	<u>MODULE</u>	<u>NO. OF FUNCTIONS</u>
13	PROCESS_VIRTUAL_PROCESSES	11
12	PROCESS_VIRTUAL_DEVICES	15
11	PROCESS_VIRTUAL_SEGMENTS	15
10	INTERPROCESS_COMMUNICATION	3
9	PROCESS_OPERATORS	10
8	SEGMENTS	15
7	MOUNTABLE_FILESYSTEMS	11
6	DEVICES	26
5	PROCESS_STATES	13
4	SUBTYPE_CONTROL	5
3	OBJECT_ACCESS_CONTROL	16
2	PRIVILEGE_CONTROL	3
1	OBJECT_NAMES	3
0	CLOCK	5
		<hr/>
		151

- APPROX 4000 LINES OF SPECIAL
- 50 VISIBLE FUNCTIONS - 12 HARDWARE GATES
38 SOFTWARE GATES