

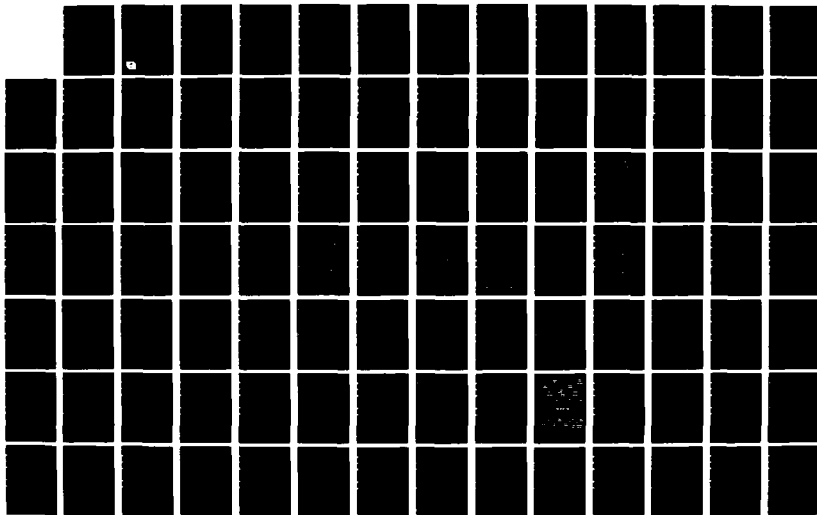
AD-A194 355

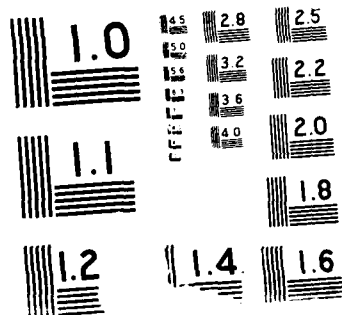
PROCEEDINGS OF THE STRATEGIC DEFENSE INITIATIVE
ORGANIZATION (SDIO) TOOL (U) INSTITUTE FOR DEFENSE
ANALYSES ALEXANDRIA VA D HEVSTEK 04 MAR 87 IDA-M-308
IDA/HQ-87-32084 NDA903-84-C-0031 F/G 12/5

1/3

UNCLASSIFIED

NL





DIC FILE COPY

UNCLASSIFIED

Copy 13 of 29 copies

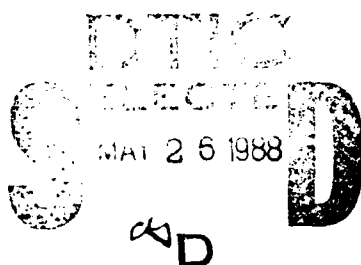
(2)

IDA MEMORANDUM REPORT M-308

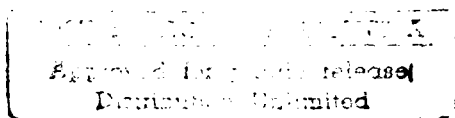
AD-A194 355

PROCEEDINGS OF THE STRATEGIC DEFENSE
INITIATIVE ORGANIZATION (SDIO) TOOL FAIR
JANUARY 26-27, 1987
ALEXANDRIA, VIRGINIA

Deborah Heystek, *Editor*



March 1987



Prepared for
Strategic Defense Initiative Organization



INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

UNCLASSIFIED

IDA Log No. HQ 87-32084

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, or (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Papers

Papers normally address relatively restricted technical or policy issues. They communicate the results of special analyses, interim reports or phases of a task, ad hoc or quick reaction work. Papers are reviewed to ensure that they meet standards similar to those expected of refereed papers in professional journals.

Memorandum Reports

IDA Memorandum Reports are used for the convenience of the sponsors or the analysts to record substantive work done in quick reaction studies and major interactive technical support activities; to make available preliminary and tentative results of analyses or of working group and panel activities; to forward information that is essentially unanalyzed and unevaluated; or to make a record of conferences, meetings, or briefings, or of data developed in the course of an investigation. Review of Memorandum Reports is suited to their content and intended use.

The results of IDA work are also conveyed by briefings and informal memoranda to sponsors and others designated by the sponsors, when appropriate.

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

This Memorandum Report is published in order to make available the material it contains for the use and convenience of interested parties. The material has not necessarily been completely evaluated and analyzed, nor subjected to IDA review.

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE

AD-A194355

1a REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS		
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release - distribution unlimited		
2b DECLASSIFICATION/DOWNGRADING SCHEDULE					
4 PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Memorandum M-308			5 MONITORING ORGANIZATION REPORT NUMBER(S)		
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses		6b OFFICE SYMBOL IDA	7a NAME OF MONITORING ORGANIZATION DOD - IDA Management Office		
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311			7b ADDRESS (City, State, and Zip Code) 1801 N. Beauregard Street Alexandria, Virginia 22311		
8a NAME OF FUNDING/SPONSORING ORGANIZATION Strategic Defense Initiative Organization		8b OFFICE SYMBOL (if applicable) SDIO	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031		
8c ADDRESS (City, State, and Zip Code) SDIO/PI Room 1E149 Pentagon, Washington D.C. 20301-7100			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO. T-R5-422
11 TITLE (Include Security Classification) Proceedings of the Strategic Defense Initiative Organization (SDIO) Tool Fair, January 26-27, 1987 (U)					
12 PERSONAL AUTHOR(S) Editor - Deborah Heystek					
13a TYPE OF REPORT Final		13b TIME COVERED FROM _____ TO _____		14 DATE OF REPORT (Year, Month, Day) 4 March 1987	
15 PAGE COUNT 207					
16 SUPPLEMENTARY NOTATION					
17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	CBD announcement, SDIO Tool Fair, Software Technology Integration Plan, software environments/tools, SA/PDL - an Ada-based Process Description Language, SDIO Program Office, System Design Languages/Methodologies.		
19 ABSTRACT (Continue on reverse if necessary and identify by block number) This IDA Memorandum M-308 has been prepared to disseminate information provided to IDA by participants at the SDIO Tool Fair (January 26-27, 1987). The objective of the task order responsible for this Memorandum is to generate a list of software tools/environments to be acquired and evaluated. The SDIO Tool Fair is a "starting point" for the evaluation of such off-the-shelf products, which will support the development, generation, simulation, or evaluation of systems described by the SA/PDL (an Ada-based Process Description Language).					
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a NAME OF RESPONSIBLE INDIVIDUAL Dr. Cathy Jo Linn, IDA			22b TELEPHONE (Include Area Code) (703) 824-5520		22c OFFICE SYMBOL IDA/CSED

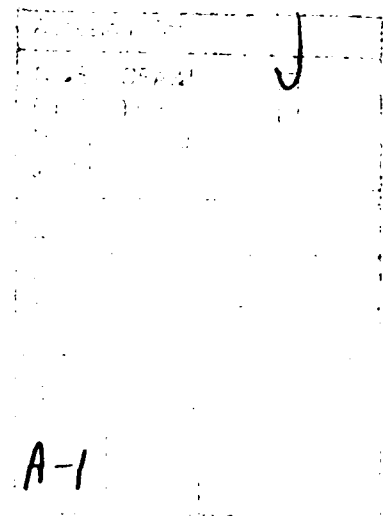
UNCLASSIFIED

IDA MEMORANDUM REPORT M-308

PROCEEDINGS OF THE STRATEGIC DEFENSE
INITIATIVE ORGANIZATION (SDIO) TOOL FAIR
JANUARY 26-27, 1987
ALEXANDRIA, VIRGINIA

Deborah Heystek, *Editor*

March 1987



INSTITUTE FOR DEFENSE ANALYSES

Contact MDA 903 84 C 0031
Task T-R5-422

UNCLASSIFIED

Table of Contents

Preface.....	iii
Acknowledgments.....	v
Introduction.....	vii
CBD Announcement.....	ix
Responses to the SDIO Tool Fair Questionnaire Appear in the Following Order:	
CADRE Technologies, Inc.....	1
Teledyne-Brown Engineering.....	12
Advanced Systems Architectures.....	25
Jodfrey Associates, Inc.....	59
Westinghouse Electric Corporation.....	67
TASC - The Analytic Sciences Corporation.....	85
Syscon Corporation.....	95
SofTech.....	111
Associative Design Technology, Inc.....	119
AD CAD, Inc.....	131
Research Triangle Institute.....	141
TRW at U.S. Army Strategic Defense Command.....	151
Intermetrics, Inc.....	173
Integrated Systems, Inc.....	185
Software Products and Services, Inc.....	195

Preface

The purpose of IDA Memorandum Report M-308, *Proceedings of the Strategic Initiative Organization (SDIO) Tool Fair*, is to make available information provided to IDA by the SDIO Tool Fair participants. This information was used during the selection process which lead to the Tool Fair.

The importance of this document is based on fulfilling the objective of Task Order MDA903 84 C 0031: T-R5-422, SDIO Software Technology Integration Plan, which is to "generate a list of software environments/tools to be acquired and evaluated." M-308 will be used as a starting point in the formal evaluation of "off the shelf" products that support the development, generation, simulation, or evaluation of systems described in the Ada programming language and SA/PDL, an Ada-based Process Description Language. As a memorandum Report, M-308 is directed towards the SDIO Program Office.

Acknowledgments

The editor is indebted to the participants of the SDIO Tool Fair for their cooperation in this effort, and to all the IDA support staff who have helped during this phase of the project.

Introduction

This document contains responses by Strategic Defense Initiative Organization (SDIO) Tool Fair participants to questions posed by The Institute for Defense Analyses (IDA).

IDA received a request from the SDIO to review Automated System Design Languages/Methodologies (SDLs). The goal was to evaluate these products and determine their applicability to the task of specifying system and BM/C3 architectures.

A Tool Fair was organized to familiarize IDA, SDIO, SDIO contractors, and other interested parties with the tools and environments currently available. To generate a list of products for participation in the Tool Fair, IDA published a Sources Sought announcement in the Commerce Business Daily (CBD) on November 11, 1986. (A copy of that announcement appears on the following page.) A number of responses to the CBD announcement were received, and from those a group of tools was selected for inclusion in the SDIO Tool Fair held January 26-27, 1987. IDA recognizes that (1) not every tool vendor was aware of the CBD announcement. Thus, it is possible that some very good tools were not represented at the Tool Fair, and (2) the selection of participants in the Tool Fair was based entirely on information provided to IDA by the tool vendors.

The next step in this evaluation of tools and environments for the SDIO is a more thorough and formal analysis of not only the products represented at the Tool Fair, but also any applicable tools and environments that may have been overlooked during the preliminary evaluation.

Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311
ATTN: Deborah Heystek

SDI SYSTEM DESIGN TOOL FAIR

The Strategic Defense Initiative Organization (SDIO) has specified an Ada Process Description Language (PDL) as a required formal presentation for descriptions of system and BM/C3 architectures. This PDL is to provide the basis for design refinement and enhancement -- a typical role for existing Program Design Languages. It is also, however, to be used to produce input to system and subsystem simulations in an automated manner.

The PDL is a representation formalism only. It has an advantage in that it does not require the use of any specific design methodology. It has disadvantages in that it is not "user friendly" (for non-Ada programmers) and since it is methodology independent, does not provide a variety of tools to assist in producing and visualizing designs.

The SDIO is interested in identifying and evaluating "higher level" System Design languages/Methodologies (SDLs). It is undertaking this effort as an initial step to deciding on the utility of specifying a standard SDL or of establishing requirements for SDLs without specifying a single formalism. To this end, it is requesting organizations that have developed an appropriate (see below) automated system design language/methodology to provide information about this language/methodology to the SDIO. It is planned that the developers of SDLs that are judged to be of potential interest to the SDIO will be invited to make a presentation in mid December to describe, analyze, and document a sample proposed SDI architecture component.

Organizations should submit a "Fact Sheet" of no more than 4 pages to describe the characteristics of their SDL. Supplementary materials (e.g., including manuals, description of formal languages) are welcome, but the Fact Sheet must be self-contained.

This fact sheet must specify the systems ability to

1. Produce Ada PDL as output
2. Represent large (over 10,000 components) systems
3. Represent both logical and physical systems
4. Accept graphical input and manipulation and
5. Run on a widely available operating system.

Planned future capabilities must be differentiated from current system capabilities.

CADRE Technologies, Inc.
222 Richmond Street
Providence, RI 02903

CADRE Technologies, Inc.

1. Describe how your system supports early detection of inconsistencies, closure and errors.

Teamwork provides semantic and syntactic error checking. Inconsistency checking and closure on analysis and design models.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

Teamwork allows the user to capture progress metrics during all phases of a project. It does this in two main ways: by maintaining versions and status information on each object that is edited.

Teamwork maintains status information of all model diagrams created in the system. This information includes who changed what object and when it was changed. There is also a place for the user to annotate each object to capture the progress as seen by the user. A status report tool is provided to list this information for all objects in a given model.

Sixteen versions of each object that is edited are maintained by Teamwork. The number of versions of an object can be considered a progress metric. Also, using Teamwork/Access users can extract many useful metrics that are captured as a natural by-product of using the tool. Indications on size and complexity of models are easily obtained. Measures of completeness of a model are also easy to obtain (i.e.) how many functions have associated data dictionary or specs that have been defined. These indicators can be useful in estimation activities or as progress indicators.

Cadre has put Bang Metric hooks into Teamwork for users interested in looking at relative complexity as a predictive aid. GEC, in the UK, is currently interfacing their GECONO (CoCoMo) tool to teamwork using the Teamwork/Access facilities.

3. Describe how your system supports documentation, program management and control.

Analysis and design models, and parts of these models, can be printed directly from Teamwork. Printers of both dot-matrix and laser formats are supported by the system. Teamwork uses Postscript and IMPRESS formats for both objects and bitmaps. Diagrams and text created in teamwork are easily included in documentation preparation packages. Teamwork currently

supports interfaces to three popular packages, Interleaf's TPS and WPS, Context's DOC, and Unilogic's Scribe. Many Teamwork users are currently using these facilities to easily create DOD-STD-2167 or similar documents.

Diagrams and text created in teamwork are also easily included in configuration management systems, such as DSEE (Apollo), CMS (DEC), and SCCS (UNIX), to facilitate project management and control. Teamwork customers are using these tools to Base-line specs and designs and maintain related source code files.

4. Describe how your system supports real time design.

Teamwork/RT provides complete automated support of the Lear Seigler/Boeing real-time analysis methodology. This methodology semantically combines finite state machine modeling (showing the control aspects of a system) with data flow diagram modeling (showing the processing and data flow aspects of a system, using the Yourdon/DeMarco methodology), providing two separate but necessary views of the system under analysis. The Teamwork Real Time capability is fully integrated with Teamwork's analysis and design tools and also provide extensive automated checking facilities.

5. Describe how your system supports concurrency, parallelism.

The Teamwork/RT methodology has the capability of showing concurrent as well as parallel systems through the use of Process Activation tables, State Transition Diagrams, and State Event matrices. These diagrams show the dynamic execution of systems and can model concurrent task instantiations.

6. Is your system constrained to a particular implementation language (Ada)?

Teamwork is not constrained to a particular implementation language. It is quite suitable for Ada.

7. Does you system produce Ada PDL?

Ada PDL can be semantically included and related to diagrams in analysis and design models where appropriate. Specific type definitions can be embedded in data definitions created in the Module Specifications inside Teamwork.

8. Describe how your system supports life cycle Intraphase & Interphase communications.

Intraphase Communication: Teamwork takes full advantage of workstation technology. Teamwork provides concurrency control for the project database, that allows multiple users (each on their own workstation) to effectively share the same project data base.

This enables several users to work in parallel on 'fleshing out' system specifications without danger or corrupting each others work. Team members can 'see' others work as it progresses and can check for consistency, either their own work only, or across the whole model as it is built. In addition the project database can be located anywhere within the network and may also be located on large backend servers (such as large VAXes) for effective support of large development teams without sacrificing the power and performance of dedicated workstations.

Interphase Communication: All Teamwork tools share a common project data base, SA structured analysis, RT real time analysis, IM information modeling, and SD structured design. This allows for very effective transitions across analysis and design phases of the project. For the phases of the life cycle not directly supported by Teamwork, the Teamwork/Access product can be used to extract information from the teamwork project data base and communicate that information to other tools. As indicated earlier Teamwork has successfully been integrated with document production tools, code development and analysis tools, configuration management tools and project management tools.

9. Is your system automated, executable, compilable?

Automated - Teamwork is an automated system for producing analysis and design specifications. The tools automate the entry, editing, storage and retrieval of related diagrams and text that comprise system models. These models can be thought of as "compilable" for the automatic semantic, syntactic and consistency checks that can be run. In addition portions of models that contain compilable code or pseudo-code (e.g., Ada or Ada PDL) are retrievable through teamwork/Access, and directly compilable.

10. Describe the graphics support for your system.

Teamwork uses high-resolution bit-mapped displays with a menu-driven interface. The menus are context-specific, and are in both pull-down and pop-up formats. Dot-matrix and laser printers output formats are supported through the use of Postscript and IMPRESS standard print languages.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics
- Macros
- Data flows
- Control flows

Prototyping is a way to quickly assess the correctness of system requirements or design decisions, by implementing a portion of the system and evaluating its performance or correctness. Teamwork supports prototyping by facilitating the entry and checking of partial models. This supports prototyping, because the user doesn't have to complete the model - he can take pieces of model down to executable code. The Teamwork consistency checker ensures that the piece of the model is semantically and syntactically correct.

Structured analysis and design techniques by themselves greatly facilitate the creation of software systems that lend themselves to reusability. It is also possible (with Teamwork/Access) to link executable code with analysis and design diagrams. By maintaining this linkage to backend librarian or configuration management systems users can greatly increase not only the design but also the maintenance of reusable code.

Teamwork supports unbounded levels of decomposition of processes as well as data.

Diagrams in Teamwork/SD can be annotated to describe packaging.

The methodologies supported by Teamwork, described elsewhere, directly support analysis and design abstraction.

Typing information can be easily included in Teamwork data definitions, and easily extracted with Teamwork/Access.

Teamwork supports evolutionary development for both individual diagram and text objects, as well as life cycle evolution. Sixteen versions of each diagram and text object are maintained automatically by Teamwork, allowing the evolutionary development of these objects. The shared common data base between all Teamwork products allows the evolutionary life cycle development from Analysis to Design to proceed without the need for re-keying the information.

Generics are supported through the use of the information hiding and the abstraction of data and processes. Also, Structure Chart diagrams in Teamwork/SD can be enhanced with annotations to show generics.

The Structure Chart of Teamwork/SD has a macro module symbol that the consistency checker understands.

The Yourdon/DeMarco structured analysis methodology supported by Teamwork includes data flows.

The Lear Siegler/Ward & Mellor real-time analysis methodology supported by teamwork includes control flows.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

Teamwork has embedded in it knowledge of Chen entity-relationship diagrams for information modelling, the Yourdon/DeMarco Structured Analysis methodology, the Yourdon/Constantine/Page-Jones Structured Design methodology, and the Lear Siegler/Ward & Mellor Real-Time Analysis methodology.

This knowledge is embedded in the language-sensitive graphic editors, as well as the consistency checker. Since checking is only performed on demand, the user may depart from these paradigms if he so chooses.

13. Describe the external tools with which your system interfaces (tool compatibility).

Teamwork objects can be retrieved in an ASCII format with the Teamwork/Access tool, and hence be interfaced with any tool that understands ASCII. Teamwork has already been integrated with documentation systems, Ada development systems, project management systems, and configuration management systems.

Interfaces have been built for the following tools:

- TPS/WPS from Interleaf
- Scribe from Unilogic
- DOC from Context
- R1000 Development Environment from Rational
- DSEE configuration management tool from Apollo .

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

The Yourdon/DeMarco Structured Analysis methodology automated by Teamwork/SA supports hierarchical decomposition and flow direction. In Teamwork, the user is not forced to take any specific approach, such as topdown. He can start anywhere in the model that makes sense and proceed from there. The system understands flow direction (as well as flow decomposition), and the consistency checker can ensure that input and output flows balance across levels of the hierarchy. Teamwork also has a "collapse" function that allows for easy repartitioning of levels of diagrams.

Designer creativity is supported through the user interface and versioning capabilities of Teamwork. The graphics are done on a high-resolution bit-mapped display, with very fast response time for the creation and editing of diagrams and text. This fosters designer creativity through exploratory analysis and design. It reduces operator fatigue, and the designer is less likely to lose a train of thought while waiting for a slow-responding system. The versioning capability of Teamwork makes it easy to draw diagrams iteratively, and evaluate them as the design is being built. It is easy to redraw and re-edit diagrams and text.

15. Is your system supported by formal syntax & semantics? Describe briefly.

The system is supported by formal syntax and semantics. See question 1.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

Hardware costs Workstation prices can vary based on vendor or volume discounts, use \$15K per workstation as an average price today. We expect prices to continue to decline to well below this average within the 87 timeframe.

Software costs \$16K list price per single seat of Teamwork, volume discounts and site licenses are available, large customers average under \$10K per seat for Teamwork licenses.

Training costs A range of training is available. Charges vary based on type of training. Contact Cadre with specific needs.

Maintenance costs \$12% annually, hardware and software.

17. Indicate the hostability (measure of degree of portability) of your system.

Teamwork is implemented in C and has proven to be portable across a wide range of workstation platforms. Teamwork is currently available on the following hardware platforms:

Apollo (All models, under Aegis or Domain/IX)

Sun (All models, under Unix 4.2/3)

DEC (All VAXstations, under VMS)

IBM (PC-RT, under AIX-UNIX)

Hewlett-Packard (9000 series, under HP-UNIX)

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

The structured analysis methodology supported by Teamwork is not specific in the abstractions/representations that can be described. In terms of modelling software, hardware, or people. The methodology provides a universal abstraction, showing transformations on data, without caring what the data is.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

The Teamwork product line supports standard methodologies where appropriate. The standard methodologies that Cadre has adapted for analysis, real-time analysis, and design (described above) describe, in an unambiguous graphical language, analysis and design specifications. Beyond that, the Teamwork philosophy is to provide a general purpose tool, and not to restrict an organization's development methodology, and to support a variety of approaches. Teamwork allows the adoption to other, non-supported formal structured analysis and structured design methodologies. Rigor is maintained by consistency checking and syntactic analysis.

Teamwork is flexible enough (through teamwork/Access) to allow communication with project control and project management methodologies without forcing any specific ones.

20. Describe how your system supports a team development approach. (Number of stations/users).

Teamwork provides shared access across a network* to the project data base. It maintains concurrency and locking for multiple users sharing the same data base. The number of stations/users is a limitation of the hardware platform and operating system, not the Teamwork software.

21. Describe how your system supports design trade-offs.

Designer creativity is supported through the user interface and versioning capabilities of Teamwork. The graphics are done on a high-resolution bit-mapped display, with very fast response time for the creation and editing of diagrams and text. This fosters designer creativity through exploratory analysis and design. The versioning capability of Teamwork makes it easy to draw diagrams iteratively, and evaluate them as the design is being built. It is easy to redraw and re-edit diagrams and text.

22. Indicate the range of problems to which the system can be applied.

The range of problems to which Teamwork can be applied include: embedded systems, real-time process control, information processing.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

Mr. Gary De Gregorio
Motorola
1303 East Algonquin Rd.
Schaumburg, IL 60196

Mr. Bob Elston
Engineering Manager
Boeing
P.O. Box 3999
Seattle, WA 98124

Mr. Dan Pawl
Manager of Software Development
Lear Siegler

Mr. Tom Weaver
McDonnell Douglas
Nuclear Technology Programs
Astronautics Division
St. Louis, MS 63166

Teledyne-Brown Engineering
West Oaks Executive Park
3700 Pender Drive
Fairfax, VA 22030

Teledyne-Brown Engineering

1. Describe how your system supports early detection of inconsistencies, closure and errors.

The Systems Engineering approach, together with the concept of using an automated paradigm (i.e., that defined by Balzer, Cheatham and Green, Kestrel Institute), dictates that automated tools be employed early in the life cycle (during the requirements and design phases) to maintain control and discipline on design rather than on the code and programming phase. The formal syntax of the Input/Output Requirements Language (IORL), a systems design language, enables an automated compilation and analysis activity to be performed early on the ensuing preliminary design prior to implementation, design finalization and code commitment. This automated compilation and analysis activity enables early detection of errors in a cost effective manner prior to commitment of long-term critical resources. This approach thus enables closure, consistency and completeness checks on the design.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

Once a design is synthesized using IORL, specific and accurate transformation ratios exist that enable the estimation of IORL design pages from requirements specification pages and subsequently to pages of software code design. A correlation can be made from A type, B type and C type specifications (i.e., from requirements/performance specifications to functional/allocated design specifications to product specifications). Present data available is consistent with and supports the ratios used for cost estimation relative to design and code.

3. Describe how your system supports documentation, program management and control.

Using TAGS technology initially during system design provides automatic documentation enabling integrity and visibility of design by both technical and management communities. The control and unique data base used to access information by development team members also ensures that information passed to other program management tools external to TAGS (e.g., PROMIS (PROMIS is a registered trademark), a PC based program management set of tools, resident as complimentary applications software) is also consistent and complete.

4. Describe how your system supports real time design.

See answer to question 5.

5. Describe how your system supports concurrency, parallelism.

The formal syntax of the systems design language (SDL) - IORL, enables it to precisely and accurately support (1) real-time design, (2) concurrency and (3) parallelism either independently or jointly as is commonly found in systems that contain functions supporting sensor processing, command and control, detection and identification, process control, network routing and control. Processing symbols such as fan-in-AND, fan-out-AND, fan-in-OR, fan-out-OR, the controlled-AND symbols, also part of IORL, support 1, 2 and 3 above. The controlled-AND symbol is also used to support the real-world computer interrupt, where parallel processors are operating, performing independent functions or operations.

6. Is your system constrained to a particular implementation language (Ada)?

TAGS technology permits designs developed to be implemented in any language (e.g., FORTRAN, PASCAL, C or Ada). However, it is focused on Ada and its automatic generation because of the benefits to be accrued in communicating information between hardware design (using Ada based VHDL) and software design (using Ada as both an implementation language and a PDL), and across various life-cycle phases (e.g., code and preliminary design phase).

7. Does you system produce Ada PDL?

Yes, Ada PDL (similar to that defined by the IEEE Ada PDL Guidelines and IDA Paper P-1983) is produced; however, the lack of uniform and well defined PDL standards in the workplace requires further dialogue within the present Ada and non-Ada user community. The Ada produced by TAGS is fully compatible and compliant with MIL-STD-1815A, the Ada Language Reference Manual. Furthermore, the involvement of the professional standards organizations, such as IEEE, as well as DoD, will be required to clearly define the specifics of what an acceptable or usable Ada PDL is to embody. The SDL Ada Process Description Language document proposed represents a viable noteworthy initiative.

8. Describe how your system supports life cycle Intraphase & Interphase communications.

The fact that TAGS technology can be used in the requirements, design, coding, testing and maintenance phases of the life-cycle clearly enables interphase communications. TAGS ability to support preliminary, formal and critical design activities enables viable intraphase support, for example.

9. Is your system automated, executable, compilable?

The automated nature of TAGS applications software packages on an engineering workstation enables executable and compilable tasks to be effected. Specifically, the diagnostic analyses, for example, enables automatic compilation of the resulting design for completeness, closure and consistency executable in a real-time mode or in a background mode.

10. Describe the graphics support for your system.

TAGS technology contains strong components intended to assist the user in visualizing such design concepts as parallelism and concurrency. The IORL design symbology graphics, intended to assist in the synthesis of design and architecture, are a part of the formal syntax that is executed and compiled. The ICON's and keyboard aids presented on the screen during the edit mode are intended to assist the user in capturing design embellishment rapidly and effectively. The electronic mouse, electronically displayed switches and multiple windowing capability are intended to facilitate access, review and edit functions in a timely manner.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Generics
- Evolutionary development
- Macros
- Data flows
- Control flows

The TAGS simulation compiler enables the generation of both early and rapid prototypes. Initial designs can be rapidly prototyped by identifying macro or generalized timing requirements. As design and timing refinements are made or subsequently established and identified, a more complete higher confidence level prototype can be generated.

The TAGS Schematic Block Diagrams (SBDs) support both information hiding and the packaging concept as identified in MIL-STD-1815A by enabling the identification and isolation of independent packages and processes. Once these entities are identified, they can then be subsequently embellished upon at a later time but within the context of the characteristics or parameters initially established, if known. SBDs enable the identification and compartmentalization of functionality at one level, while the related implementation and elaboration can be affected independently and subsequently at a lower level by another individual.

The hierarchical relationships and nature of IORL supports the various levels and concept of abstraction required to represent and manage design and software. The different parameter tables contained within the language (i.e., IOPT's, IPT's, PPT's), provide the capability to define various data types. Scalars, Strings, Vectors, Matrices, Logicals and Sets are a partial list of types that can be represented. These IORL features, SBDs, parameter tables, predefined processes and others also support the definition and declaration of generics as well. (See the IORL language reference manual for further elaboration).

TAGS technology is particularly effective in supporting evolutionary development by providing the capability to coexist with different, immature and incomplete design baselines. The tool suite provides the capability to individually customize and complete subsystems and components without perturbing completed design modules. The configuration management applications package enables the maintenance and integrity of multiple system baselines and iterations. Transitioning from one design baseline to another is facilitated.

Another of the IORL formally defined symbols is the Macro. It shows all of the fundamental input/output events or represents a combination of fundamental I/O events defined by a single input or output symbol. By condensing I/O information in this manner, the amount of detail in a diagram or design is reduced significantly and becomes more manageable. Macros are used primarily to represent the complete transmission of data, from initiation of contact between two sides of an interface to acceptance of the data received. They may include the line protocol used in transmitting the data. (See the IORL language manual for further elaboration).

The schematic block diagrams of IORL enable a precise, formal, complete and compilable representation of data flows with associated functionality and architectural representations. The SBDs also include all internal and external data and parameter passing with the associated structure. Another compilable representation, the data structured diagram (DSD) provides a graphic representation of system data. The tabular DSD describes physical data organization and storage allocation. The form DSD specifies a menu or report format (i.e., a graphical format statement). The picture DSD displays a drawing as a reflection of data functions. These features also allow the generation of an automatic data dictionary.

The Input/Output Relationship and Timing Diagram (IORTD), a part of the formal syntax, is used to represent overall control flow for associated data flow diagrams. TAGS allows immediate and automatic access and comparison of all system data and control flows to provide design visibility.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

The Balzer, Cheatham, and Green automated paradigms as described in **Computer**, November 1983, IEEE publication, are embodied within TAGS technology as well as TBE's own. These paradigms stress early detection of problems (i.e., in the requirements and design phases), and the ability to generate high confidence prototypes via automated tools coupled with prototype and design repeatability for consistency. These paradigms stress documentation of design as a natural by-product to enable design and requirements maintenance versus the present day approach to systems development of code maintenance. The latter is undesirable because of its inherent characteristic to destroy structure and insight as a function of maintenance and time.

These paradigms also enable classical and evolutionary development to coexist with prototyping initiatives that provide synergism within the life-cycle, enable better hardware/software tradeoffs, and reduce the amount of time (i.e., life-cycle) required for system realization.

13. Describe the external tools with which your system interfaces (tool compatibility).

A number of external tools (i.e., developed by other vendors) can be interfaced and integrated with the TAGS tool suite. A postscript interface exists that can directly interface with such tools as the Mentor Graphics Company's *Context*. The MIL-STD-1815A *Language Reference Manual*

adhered to shall enable code produced by TAGS to be compatible with any other Ada DoD validated software development environment. A number of other proprietary TAGS products are imminent and under development that will enable a variety of other tools to be directly interfaced with the applications packages. Current efforts are underway to integrate TAGS technology with specific requirements tools of NASTEC Corporation. It is TBE policy to interface with existing viable tools that extend TAGS utilitarian value and extensibility rather than build its own. This policy enables the company to focus on unique tool development activities to further increase the viability of TAGS technology. Another anticipated standard, to enable further tool extensibility, is the VHDL hardware description language (VHDL) interface completion.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

The TAGS formal development methodology is based on sound systems engineering principles that can be characterized by four basic activities:

- Conceptualization
- Definition
- Analysis
- Allocations

The language enables hierarchical decomposition and provides for the initial identification of a system in its environment with all associated interfaces (top-down).

However, since TAGS technology is intended to provide design freedom and innovativeness on the part of a designer or engineer, lateral *departures* and a bottoms-up approach are not precluded where user experience and expertise can be relied upon. Thus, control flows at any level can be undertaken with *departures* into the world of data flows or vice-versa without adverse limitations or design consequences. The technology enables functionality to co-exist with architectural representations and design formalisms in an efficient and *integrated manner*.

15. Is your system supported by formal syntax & semantics? Describe briefly.

The formal syntax and language embedded within TAGS technology is the First Order Requirements Language.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

A single and complete TAGS stand-alone node (software) with all thirteen design and configuration management tools costs \$16,250. However, individual user requirements may dictate fewer tools that can range from a low of \$2,500 to a high of \$5,000. Network (8 nodes or more) licenses qualify for discounts of up to 50%. The simulation compiler (prototype generator and timing analyzer) costs \$12,500. Training is included with the purchase of a single system for two individuals. The TAGS complete and basic course is 40 hours long.

The engineering workstation, an Apollo DN-3000 is approximately \$10,000. Hardware and software maintenance for a single node is approximately 10% of purchase price. All hardware and software can be purchased separately or as a turn-key system.

17. Indicate the hostability (measure of degree of portability) of your system.

The present hosts of TAGS technology are the Apollo computers and the DEC VAX (VAX is a registered trademark of the Digital Equipment Corporation) series. Additional hosting is underway and will be made available subject to nondisclosure agreements.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

The systems engineering methodology embodied within TAGS technology enables it to also support software and hardware design as part of the natural process of synthesis required to build systems. Thus TAGS technology and tools are the same ones used to support the hardware and software engineering activities. The strength of the approach in developing an integrated development environment, is severalfold: it permits viable hardware/software tradeoffs and timing analysis; insures good communications, information exchange and consistent documentation between hardware and software; fosters uniform training and tool innovation between the various life cycle design and development members; insures continued life-cycle tool usage from requirements and design through test and maintenance.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

The intent of TAGS technology is to support and embody all of the referenced methodologies. However, it is recognized that to have a complete methodology that accomplishes all is difficult and does not currently exist in a mature and well documented form. Nonetheless, Teledyne Brown Engineering feels that the TAGS technology approach taken currently satisfies several of these and complements a number of others.

The technology currently provides a development, design and programming (partial) methodology. It is also compatible with several structured and object oriented approaches (i.e., enables one to design and develop with TAGS and implement code traditionally at the user's discretion). However, invocation of automatic code generators and prototype analyzers can be expected to emphasize design areas previously ignored or deemphasized (e.g., code and algorithm optimization, design for reusability a priori).

Presently a number of software experts at Auburn University, Jersey City State College, and Georgia Institute of Technology are presently researching and evaluation innovative programming methodologies and TAGS in their efforts to increase software productivity.

Additionally, a number of companies that have integrated TAGS into their environments (i.e., as the basic design engine and data base) are developing project control and management components in efforts to establish an overall methodology that satisfies the criteria for completeness and extensiveness of methodology. Teledyne Brown Engineering is one of these companies. By integrating existing management and project control tools with the present TAGS environment, TBE feels it can identify an existing environment that can be used to support systems of the magnitude that SDIO and NASA are undertaking. These integrated "complete" environments can then be used to develop more mature intelligent and enhanced tools.

**20. Describe how your system supports a team development approach.
(Number of stations/users).**

By taking advantage of the networking capabilities offered by vendors (e.g., Apollo's Domain network approach), workstations can be linked together to support large design teams. The TAGS technology and software does not contain limitations on the number of users. Some present TAGS users have networks in excess of 50 workstations. Typical network rings are configured in multiples of eight workstations. Individual company requirements and the need for sensitive compartmented information have been found to dictate network sizings rather than technology constraints at this time.

21. Describe how your system supports design trade-offs.

See question 18 response.

The technology with the simulation compiler, can be used to generate prototypes in support of specific performance requirements. Less than satisfactory results require only that the user isolate modules or subsystems that require further optimization or timing changes. Once localized changes are made, the prototype can then be simulated again and new output data analyzed. The latter is an iterative process that enables timely and efficient hardware/software tradeoffs to be effected.

22. Indicate the range of problems to which the system can be applied.

TAGS technology has been employed across a wide range of systems and problems. Systems that include electronic warfare sensors, command and control, aircraft, communications, logistics support, process control, weapons real-time algorithms, management information systems, banking, architectural, DoD and non-DoD systems have been designed using TAGS technology. The systems engineering approach contained within TAGS is a broad-based methodology that can be applied across a wide range of problems. Specific TAGS applications of interest to the SDIO office are the Airborne Optical Adjunct (AOA), Satellite Integration Experiment (SIE), and National Test Bed (Rockwell team). Several thousand FORL design pages currently exist on these systems.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

Boeing Aerospace Company
1001 Southwest, 41st Street
Renton, WA 98055

Boeing Aerospace
P.O. Box 3999
Seattle, WA 98124
Bob Welling - (206) 764-0604

Sperry Corporation
Mall Station 104
Great Neck, NY 11020
Charles Pace - (516) 574-9261

Siemens
D AP 52
Otto - Hahn - Ring 6
P.O. box 830951
D-8000 Munich 83
Federal Republic of Germany
Peter Hoyer - 49 89 636-46065

MSI, Inc.
600 Maryland Avenue, SW
Suite 695
Washington, DC 20024
Jack Ridgway - (202) 554-6161

Jersey City State College
2039 Kennedy Boulevard
Jersey City, NJ 07305
Dr. Phillip Caverly - (201) 547-3291

Advanced Systems Architectures
Johnson House
73-79 Park Street.
Camberley.
Surrey. GU15 3PE.
U. K.

Advanced System Architectures

1. Describe how your system supports early detection of inconsistencies, closure and errors.

Auto-G supports detection of inconsistencies, closure and errors in the following ways:

1.1 By Visual Inspection Of Auto-G Documents.

The strict formality of the G/T notation guarantees that the contents of each Auto-G document can be unambiguously interpreted. If system requirements are expressed using Auto-G, the risk of a designer misinterpreting these requirements is minimized; if a system design is expressed using Auto-G, the functionality is immediately visible. Any discrepancy and inconsistency between requirements and design is therefore immediately apparent.

1.2 By Creating Only Syntactically Correct Documents.

Any attempt to violate the syntax rules is an indication of some inconsistency. Using the graphical man-machine interface, the user cannot create invalid syntax; using the textual interface, syntax is checked on request, and invalid syntax is not stored as a valid document.

1.3 By Semantic Checking.

Auto-G includes an automatic checking function which can be used to check the validity and completeness of a specification or design document. This function performs a series of semantic checks which together constitute a complete static analysis; when the checker is run it produces a detailed report on all functional aspects of the document(s) being checked.

The same constructs are used both to specify and to design the system. The checking function warns of any redundancy in the design - i.e. whenever variables are not accessed, procedures are not called, variables procedures and templates are not used, input parameters are never read or output parameters never assigned to. In addition, the checker warns of possible cases of unintended interference between functions which have an unspecified sequence of execution.

A specification is checked to ensure that there are no loose ends - i.e. that all the inputs and outputs to the system are related to objects in its environment.

A design is also checked to ensure that there are no loose ends - e.g. that the behavior of every function is consistent with its interface definition. The checker detects any processing that can never be executed because in practice it is unreachable.

As well as its semantic checks, the checker warns of the use of any legally valid design that could nonetheless compromise the reliability of a system e.g. the use of data shared by concurrent processes.

1.4 By Dynamic Analysis Using Auto-X.

To prove the completeness of a requirement, a specification, or a design, it is not sufficient merely to use a formal structured and quantified approach. Dynamic analysis and prototyping are increasingly recognized as important additional aids toward assuring the consistency and correctness of the evolving schema. Auto-X is the tool which provides this support for Auto-G users. The objective of this tool is to prove that the concept, specification or design is workable in the dynamic sense and to provide the necessary parameters, if required, for sizing the target system.

The functional specification of a system is a definition of its functional components and of the interaction that takes place between them. The Functional Exerciser (Auto-X) is designed to operate upon functional specifications and system designs structured in this way and which have been formally described using Auto-G. Auto-X functions with systems which have been defined in this way, by intercepting the signals (messages) that have to be passed between such objects. It is thus used in a similar manner to the circuit designer's oscilloscope, relying upon the information passing across an interface to verify, and characterize, the operation of a functional element.

Using Auto-X, functional specifications can be modeled: in this way, the implications of a complex set of requirements may be demonstrated to the customer who can then confirm that the specification is complete.

A consolidated Auto-G requirements specification, in effect, represents a formal environment model against which the associated system design may be checked for consistency. Throughout the design stage, Auto-X may be used to model the design itself: the design model interacts with the specification, and in this way the suitability of the design can be confirmed. If the design is not completed to the lowest level of detail, the design can still be

modeled using the Auto-X but the Auto-G checker identifies that there is further design work to be done.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

The question is answered in Section 19c.

3. Describe how your system supports documentation, program management and control.

The purpose of documentation is to record information about a system in an orderly manner in order that people may understand the system either to build it, to use it or to maintain it. The outputs from Auto-G (either graphical or textual) are an easy-to-understand formal representation of either the specification of the system or of the system itself: quite simply, Auto-G systems are largely self-documenting. The Auto-G database has also been linked to other tools including one which generates documentation to DoD standards e.g. DOD-STD-2167. Documentation produced directly from Auto-G does not eliminate the need for user manuals and hardware maintenance documentation. In addition, it is useful to record the critical decisions that affected the design: informal comments within the design tend to amplify WHAT is done rather than WHY.

Information stored by Auto-G is stored in Auto-G documents. Within the Auto-G database, each document has its own physical file. These documents can be arranged in a hierarchy to reflect the breakdown of the design phase into a network of design activities. Each design activity can then be worked on by an individual or by a small team working closely together. Estimating how much effort is required for each design activity can only be done empirically: with a set of effort estimates, critical path analysis and resource scheduling could easily be done based upon the network described by Auto-G.

Auto-G can be linked to another tool, or can be fully integrated with a project support environment (PSE) which provides configuration control of Auto-G documents and of other documentation.

4. Describe how your system supports real time design.

Auto-G supports a semantic model which is capable of representing any functional concept, however complex, in a way which is clear and unambiguous. The formal G/T notation is able to describe completely all functional aspects of a real-time system namely:

System Structure

- hierarchy
- Independent processes
- common procedures
- data areas

Process/Procedure Behavior

- local data and data flow
- Internal control flow
- transitions between states
- processing algorithms

Communication Between Functions

- messages sent and received
- external data accessed
- common functions utilized
- detailed protocol descriptions

Configuration Management

- modular construction and test
- parallel design activities
- system engineering

Other features that make Auto-G particularly appropriate for applications to complex real-time computing systems include the formal representation of time, and full support for generic design templates.

5. Describe how your system supports concurrency, parallelism.

5.1 Auto-G Supports Concurrency.

The Auto-G semantic model postulates the whole of a system and its environment as a set of independent concurrent processes that, fundamentally, operate asynchronously. They interact/communicate only through the passing of messages (called 'signals'). Thus Figure 5.1 represents a system which contains two concurrent processes, and which interacts with two types of process in its environment.

When processes communicate, it is not necessary for the sender and the receiver to be synchronized: the sender can dispatch a message which may not be attended to by the receiver until some later time. Where functionally required, synchronization between two processes is represented by a protocol involving an exchange of messages. The G/T notation therefore supports precisely the behavior of processes in the real world.

5.2 Auto-G Supports Parallelism.

The Auto-G semantic model describes the behavior of a process as sequences of actions which may be triggered by various conditions or events. The G/T notation allows the designer to distinguish between actions that must be performed in a definite order (Figure 5.2) and groups of actions that need not be performed in any particular order provided that all are completed before the next action is started (see Figure 5.3). A description of behavior

EQ/v1

Environment Overview of a Display Driver System which receives display messages from an external source and displays the contents of these messages on a display. The level of displayed information is controlled by means of control messages sent to the system by users of the system.

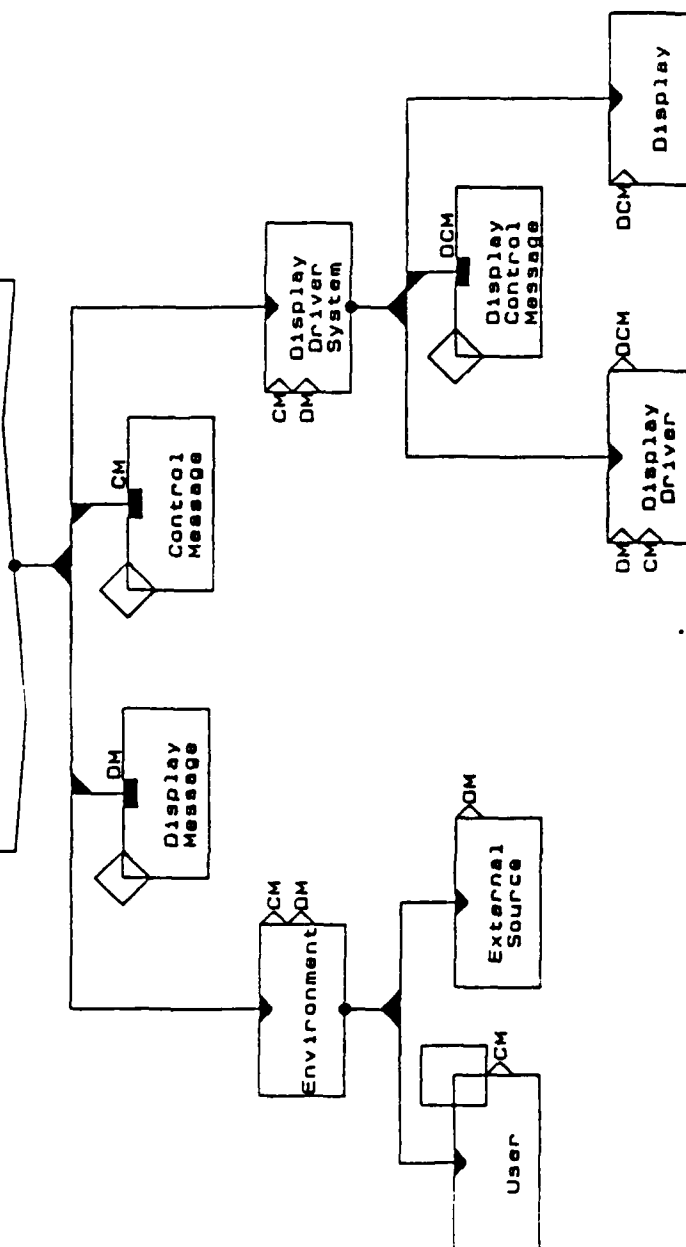


Figure 5.1 : CONCURRENT PROCESSES
COMMUNICATING VIA SIGNALS

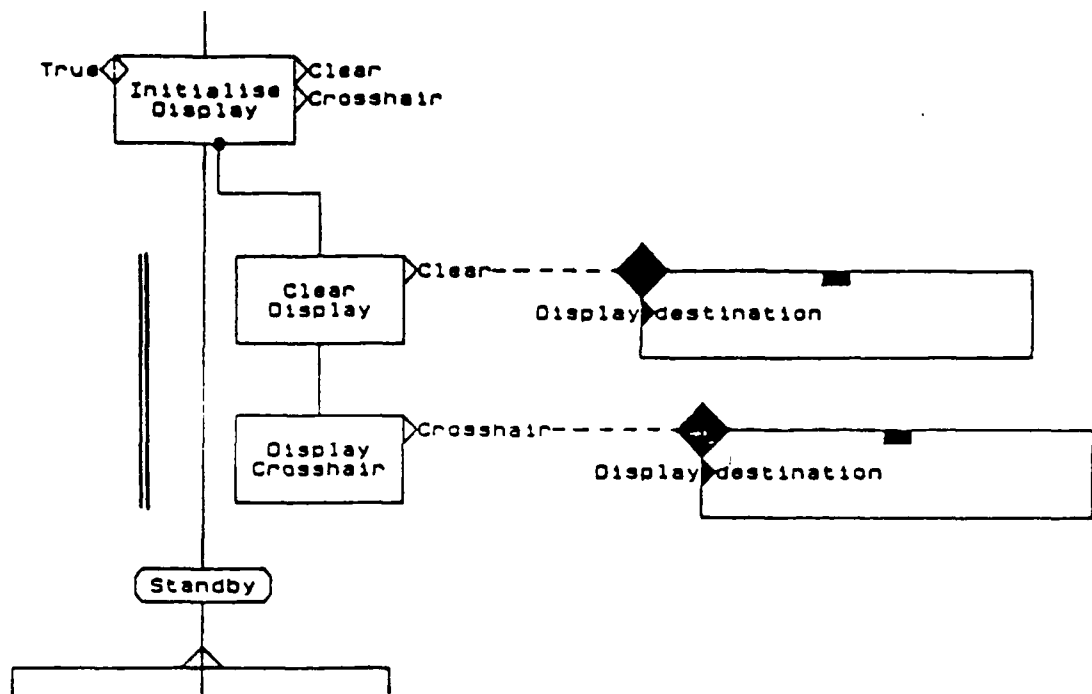


Figure 5.2 : ORDERED ACTIONS

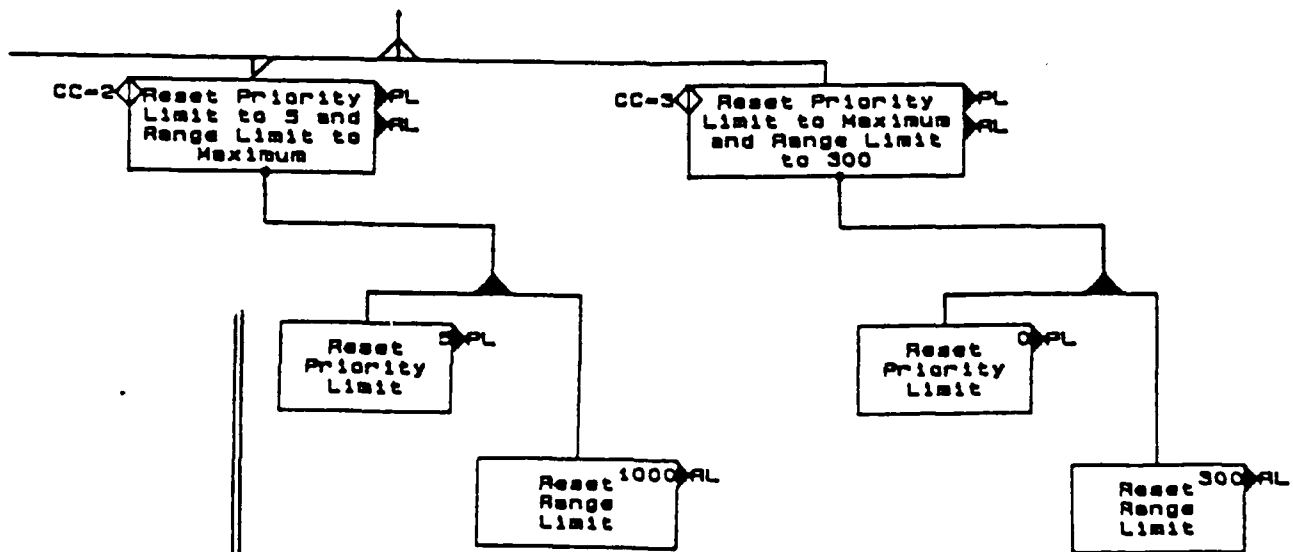


Figure 5.3 : UNORDERED ACTIONS

which uses the concept of unordered actions identifies those parts of the design where parallel processing could give a faster execution.

6. Is your system constrained to a particular implementation language (Ada)?

No. As a free-standing computer aided design (CAD) tool, Auto-G can be used to produce a proven design that can be implemented manually in any language suitable for real-time systems.

The Auto-G database has been linked to other tools for use during the implementation phase: one such tool is a text editor which guides programmers coding in JOVLAL; another tool generates test data automatically using as a basis the interface descriptions within the database.

If a toolset Code Generator is used, then it is possible to generate object code directly without the use of any implementation language.

7. Does your system produce Ada PDL?

Auto-G already produces various codes directly from a design (e.g., Ada, C); studies have shown that other codes (e.g. occam) could be produced relatively easily.

The G/T notation permits assertions about the design to be embedded within the design, just as Ada PDL is embedded within an Ada program. Auto-G does not at present produce Ada PDL, but it could be extended to do so if required.

8. Describe how your system supports life cycle intraphase & interphase communications.

One of the major headaches in systems development can be the use of many notations at different phases of the life cycle. Whenever information has to be transformed from one notation into another, there is the danger of the loss or distortion of that information, particularly when either of the notations isn't formal. Auto-G has a complete formal notation that can be used at every stage of the system development. The benefits of this are three-fold: firstly, transformation errors are eliminated; secondly, the use of the same notation throughout the life cycle makes it much easier to compare the outputs of different phases; thirdly, the use of a single notation within a project reduces the learning curve for the development personnel.

Figure 8.1 shows the differences between the successive phase of the life cycle. The following paragraphs describe these phases and show how Auto-G facilitates the transition from one phase to the next.

Auto-G naturally obliges the analyst to take a structured approach to the understanding and expression of a requirement. He begins by using Auto-G to capture information about the system as it is seen from a number of different viewpoints. Initially, what is known about the system from one viewpoint may be imprecise or ambiguous; Auto-G identifies these uncertain areas so that the analyst can request clarification. Although the information captured by Auto-G is clear and unambiguous, it is likely that different viewpoints will be inconsistent with each other. The Auto-G notation makes it easy to identify where the requirements of different viewpoints overlap, and where one viewpoint disagrees with another.

The requirements capture phase is followed by an analysis and consolidation phase. During this phase, the analyst pieces together the picture from the different viewpoints, and refers to the users to resolve the inconsistencies that are present. Auto-G enables the results to be checked automatically for consistency, and Auto-X enables the implications of the results to be understood and agreed with the users. The output from this phase is an agreed complete formal description of the interface between the system and its environment, and an environment model that represents the performance criteria for the system to be accepted.

Future releases of Auto-G will provide automated assistance to the process of identifying mutually inconsistent viewpoints and of consolidating them into a consistent picture.

A system may be doing many things at once; in the design phase, a technique of functional decomposition is used to identify functions that cannot be performed concurrently. Auto-G enables the designer to represent explicitly the fundamental timing requirements and to arrive at a plausible design. The match of the design against the specification can be done by visual inspection - this is straightforward because both the specification and the design are expressed in the same notation. The detailed behavior is also checked periodically against the specification using Auto-X; when the detailed design is complete, the outputs from Auto-X will provide an assessment of the computing resources required by the system. The output from this phase is a design which is as good as can possibly be achieved on the basis of a full dynamic analysis.

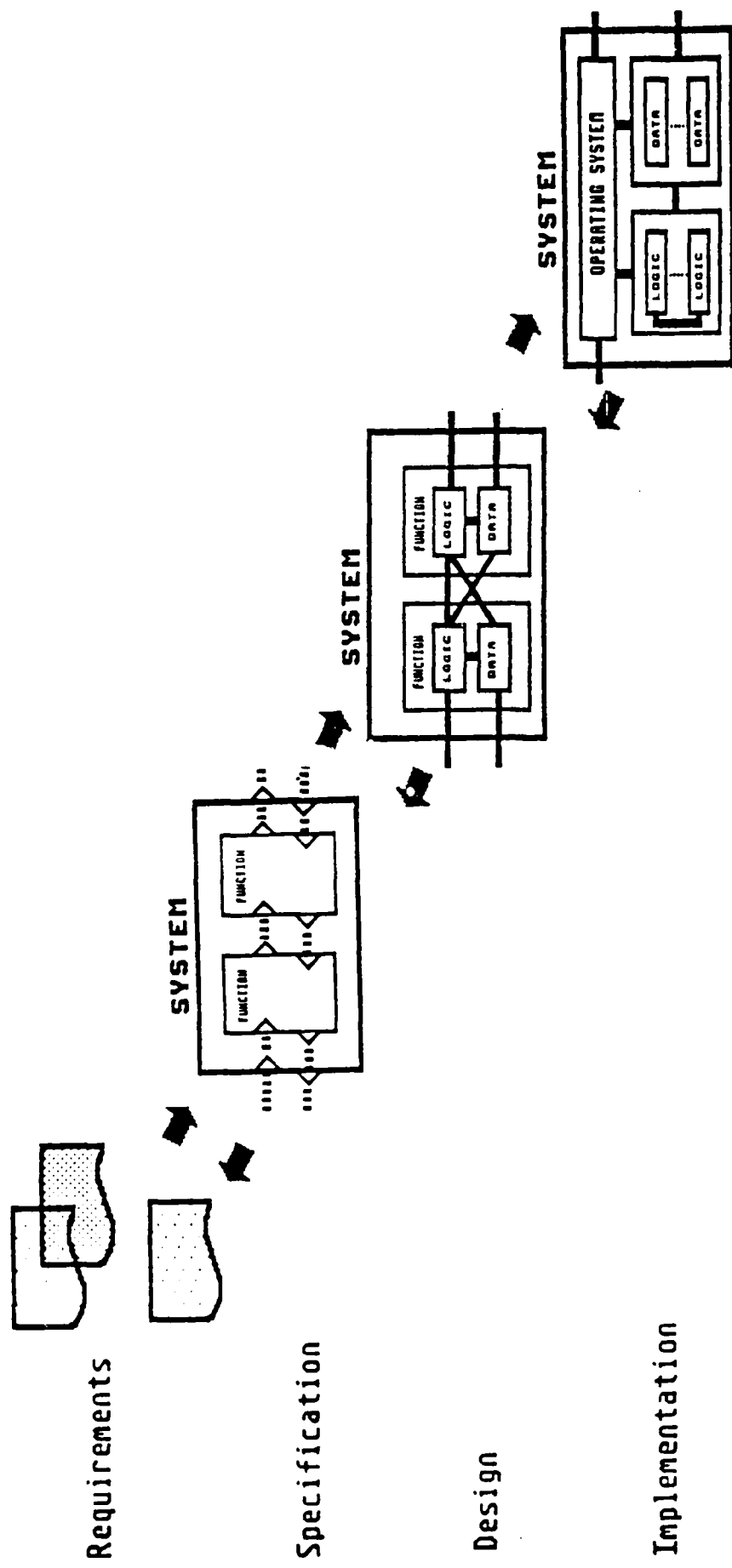


Figure 8.1 : PHASES IN THE SYSTEM LIFE CYCLE

Using information provided by Auto-X, a suitable architecture can be proposed for the target system, and this architecture can be described using the G/T notation. It is intended to provide assistance for this in future releases of Auto-G. Once the architecture is adequately described, code generation for the system can be made fully automatic. Alternatively, the detailed design may be given to a team of programmers for implementation using a manual coding phase.

Coding is followed by testing in the traditional manner, except that the acceptance criteria are formally defined as part of the requirement. If the target processor selected for implementation is the Sofchip Processor developed by Advanced System Architectures, then Auto-X is used as the symbolic debugger during the testing phase.

9. Is your system automated, executable, compilable?

9.1 Auto-G is Automated.

The Auto-G workstation is an intelligent tool which enables the user to develop a specification or a design within the rules of the Auto-G semantic model. The graphics interface to Auto-G uses a syntax-driven menu, which changes as the user modifies his document - it is impossible to create a document that is syntactically incorrect. The textual interface performs a complete syntax check on a document each time it is returned to the database. In addition to these syntax checks, the Auto-G checking function performs an automatic check on the semantics of any Auto-G document(s), and on their consistency.

9.2 Auto-G is Executable.

Auto-X enables any specification or design to be checked dynamically. Auto-X uses an executable code derived directly and automatically from Auto-G documents.

9.3 Auto-G is Compilable.

A number of Code Generators will be available during 1987 for use with the Auto-G toolset. These Code Generators will produce either an object code for a particular target processor, or a portable High Order Language which can be compiled to a number of processors. A Code Generator for ASA's Sofchip Processor (a dynamic parallel processor) is nearly complete; this will be followed by Code Generators producing program text in both Ada and C.

10. Describe the graphics support for your system.

Auto-G provides two interchangeable and complementary man-machine interfaces - both a textual and a graphical interface.

The graphical interface is used to produce documents in the form of G diagrams. Auto-G uses a syntax-driven menu, which changes as the user modifies his diagrams - It is impossible to create a diagram that is syntactically incorrect. Auto-G includes a powerful set of editing commands for the fast, accurate production of diagrams. These may be entered from the keyboard or selected from the on-screen menu. Prompts inform the user when the system is waiting for further input e.g. to complete an operation or for the next command.

Facilities include: Change, Move, Copy, Delete, Hide, and Reveal one or more items; Explode, Implode, Insert, reposition, Scale Up or Down, and Select an item.

Commands which assist the designer to gain a clearer picture of the context of a particular part of a G diagram include Zoom In, Zoom Out, Explode, Implode, hide, and Reveal. All of these may temporarily (or permanently) adjust the current 'view' of an item in relation to its components or its surroundings.

A most important aspect of Auto-G's editing facilities is the ability to select, display, manipulate and store meaningful and consistent subsets of documents. These are referred to as 'views', and can be used for various purposes. It is possible to have many views of the same document, and to produce view diagrams in hardcopy for use in presentations and in printed documentation. A view can provide a summary of all or part of a design, uncluttered by detail or lower levels of abstraction. The most important benefit of views is that they can be used to communicate ideas and information very effectively.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics

- Macros
- Data flows
- Control flows

Auto-X is a prototyping tool with a difference: the prototype is one and the same as the design being developed. Using Auto-X, the working prototype is not thrown away - it is progressively refined until it becomes the complete detailed design.

Using Auto-G, a system and its operating environment are described as a set of asynchronous functions (objects) coupled together via simple unidirectional communication links (signals). Auto-X is designed to operate upon system designs structured in this way which have been formally described using Auto-G.

Auto-X is designed so that it can be used at every level of the design process. Auto-X contains facilities for incorporating assumptions and estimates about the behavior of parts of the system, and the external world. At any stage, Auto-X can be used to confirm the matching between the system design and its specification, and to produce reliable estimates of the system performance. As the design of the system is developed, the assumptions are replaced by design detail, and estimates based upon outputs from Auto-X are progressively refined and improved.

Auto-X provides facilities to describe the dynamic properties of the functional objects and the way they communicate. The user may specify time delays caused by processing within a function, and also delays associated with inter-function signal transfer. The user may set up signal transfers between functions in which valid random values are generated for data fields. In addition, the user may load the system in a predetermined manner by specifying signal traffic distributions.

Once these parameters have been specified for the set of system functions of interest (and for their external environment), dynamic exercising of the system, at the level specified, can begin. Auto-X logs events within the system that can be used for dynamic flow analysis. Many different options are available to the user to allow him to verify, modify, and metricate his evolving system. These facilities allow him to gather statistics that can be used to scale the real system and to infer its performance. In addition, Auto-X provides facilities that allow the user to interact with the system at all levels.

Once a design has been completed down to the lowest level the effect of Auto-X closely parallels the operation of the conventional debugger. Deadlocks and race hazards in a structure can be clearly identified at an early stage in the design cycle.

Once a test has been set up, it can be run any number of times. It is therefore possible to set up acceptance tests for any part of a system (up to and including the entire system); these tests can be re-run whenever any change is made which affects that part of the system.

The G/T notation encourages designers to develop generalized solutions to system problems. These generalized designs may then be re-used efficiently in different circumstances.

There are two aspects of the G/T notation that are connected with design reusability: generics and macros.

The G/T notation supports several kinds of information hiding.

From an operational point of view, the notation supports object-oriented, hierarchically structured systems. To describe systems of this sort, the notation supports scope and visibility rules that can be used to prevent a functional object from referring to or accessing an entity defined inside another object.

From a development viewpoint, the graphical man-machine interface of Auto-G supports a different kind of information hiding through its 'view' facility. The view facility enables the user to hide or reveal any item within a G diagram or to alter its relative size and position. View information can be stored within the database and a user can maintain many views concurrently of a single Auto-G document; Auto-G guarantees that all the views are consistent with the underlying design. Figures 11c.1, 11c.2 and 11c.3 are all views of a design that is much too large to be shown clearly on an A4 page. The view facility is far more than a means of hiding information - it is also a powerful means of communicating information.

Using Auto-G, it is easy to describe any system as a set of modules, each described within a separate Auto-G design document. For every document, the user supplies a name and a set of version descriptors that together form an unique document identifier.

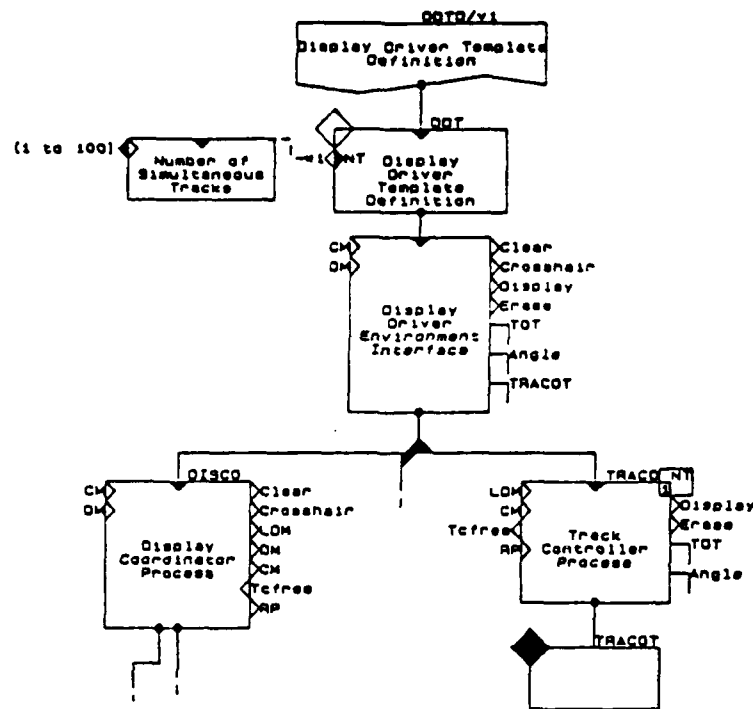


Figure 11c.1 : THREE VIEWS OF THE SAME DESIGN

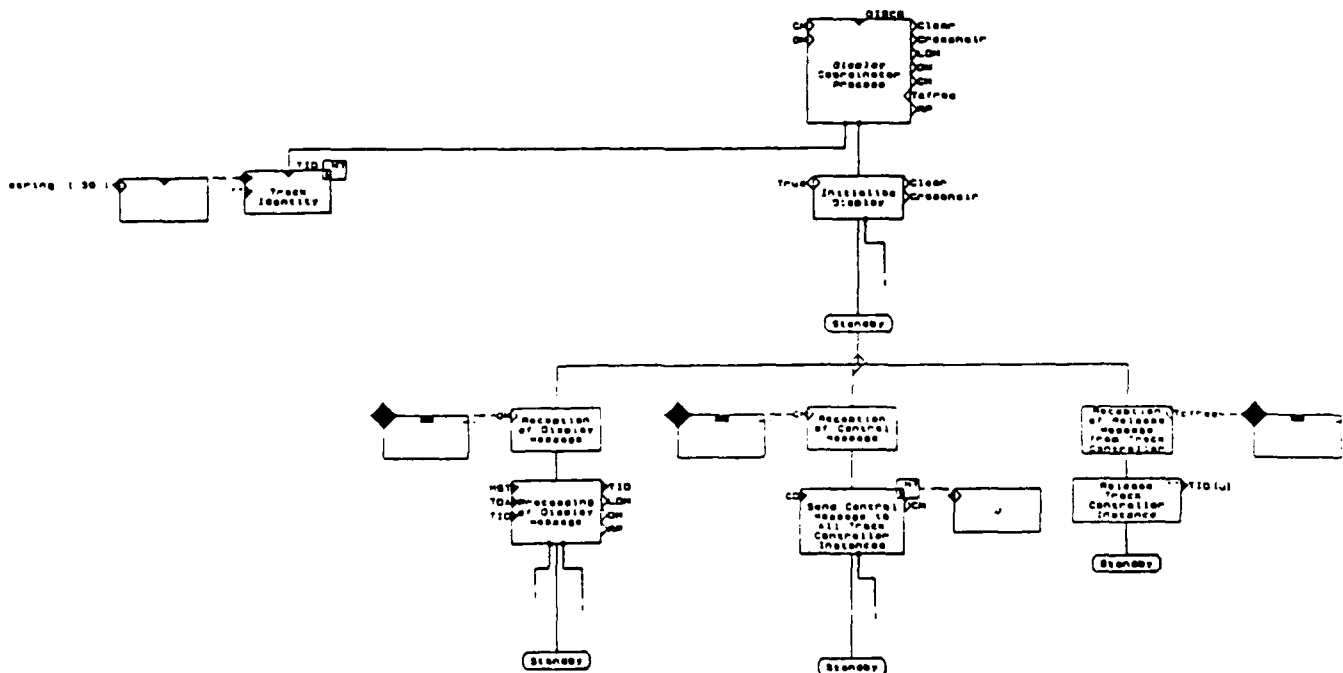
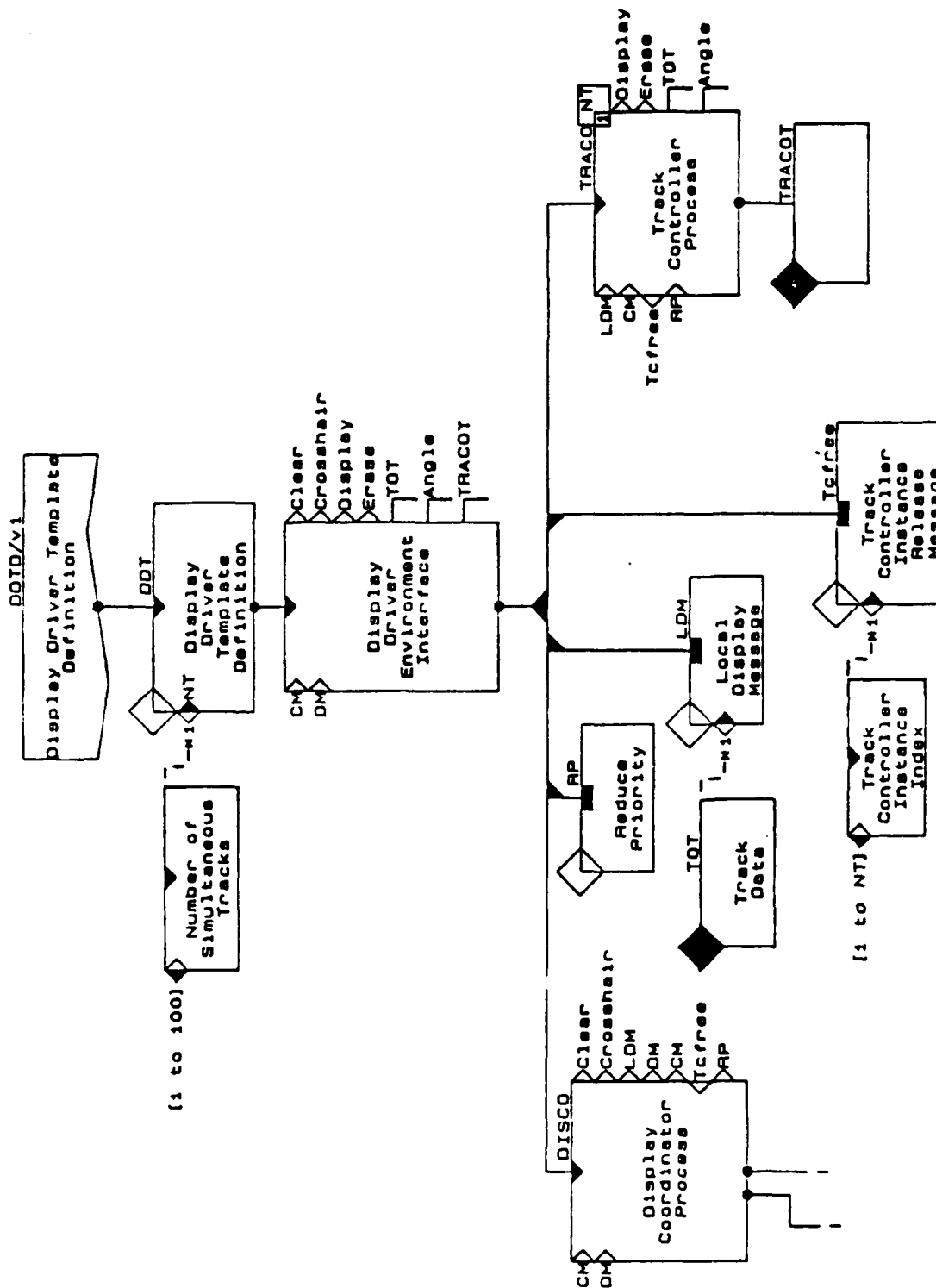


Figure 11c.2 : THREE VIEWS OF THE SAME DESIGN

Figure 11c.3 : THREE VIEWS OF THE SAME DESIGN



A package of modules may be defined by creating an Auto-G 'configuration document' which lists the names and versions of all the components: the package could be a complete system, or a group of functions within a system. Modules may be used in many different systems simply by identifying them in the appropriate configuration documents. When it is necessary to change a system by altering one of its modules, the module document is given new version descriptors - this makes it impossible for changes to be made inadvertently to the original system.

Auto-G also supports the concept of packaging through the use of generic design templates.

Auto-G provides total support for the concept of abstraction. Using a top-down approach to specification or design, the user may begin with very general ideas which he can then decompose into lower levels of greater detail. The process of decomposition may be applied to both the structure of the system and to the behavior of its components. Auto-G allows the designer to use as many levels of abstraction as he wishes until he reaches the 'atomic' structural or behavioral statements. Figures 11e.1, 11e.2 and 11e.3 show the progressive decomposition of a simple system as seen through the graphical man-machine interface.

Every object in the G/T notation has a type which determines what operations can be performed on that object.

Every non-data objects have a type; for example, an object of type 'signal' may be sent or received, an object of type 'independent function' may be given as the destination of a signal.

For data objects, the G/T notation has many built-in types (e.g. integer, set, character-string) out of which it is possible to define complex record types.

The notation enables the user to define his own types as a matter of convenience. For instance, a single type could be defined to represent a record structure with many fields of differing types.

The user can define his own 'new' types which are distinct from all other types: new types can be used to prevent design errors such as can occur through the use of the wrong units in an expression of quantity. For instance, by defining 'new' types for minutes and seconds, any inadvertent assignment of minutes to a variable of type seconds may be detected automatically.

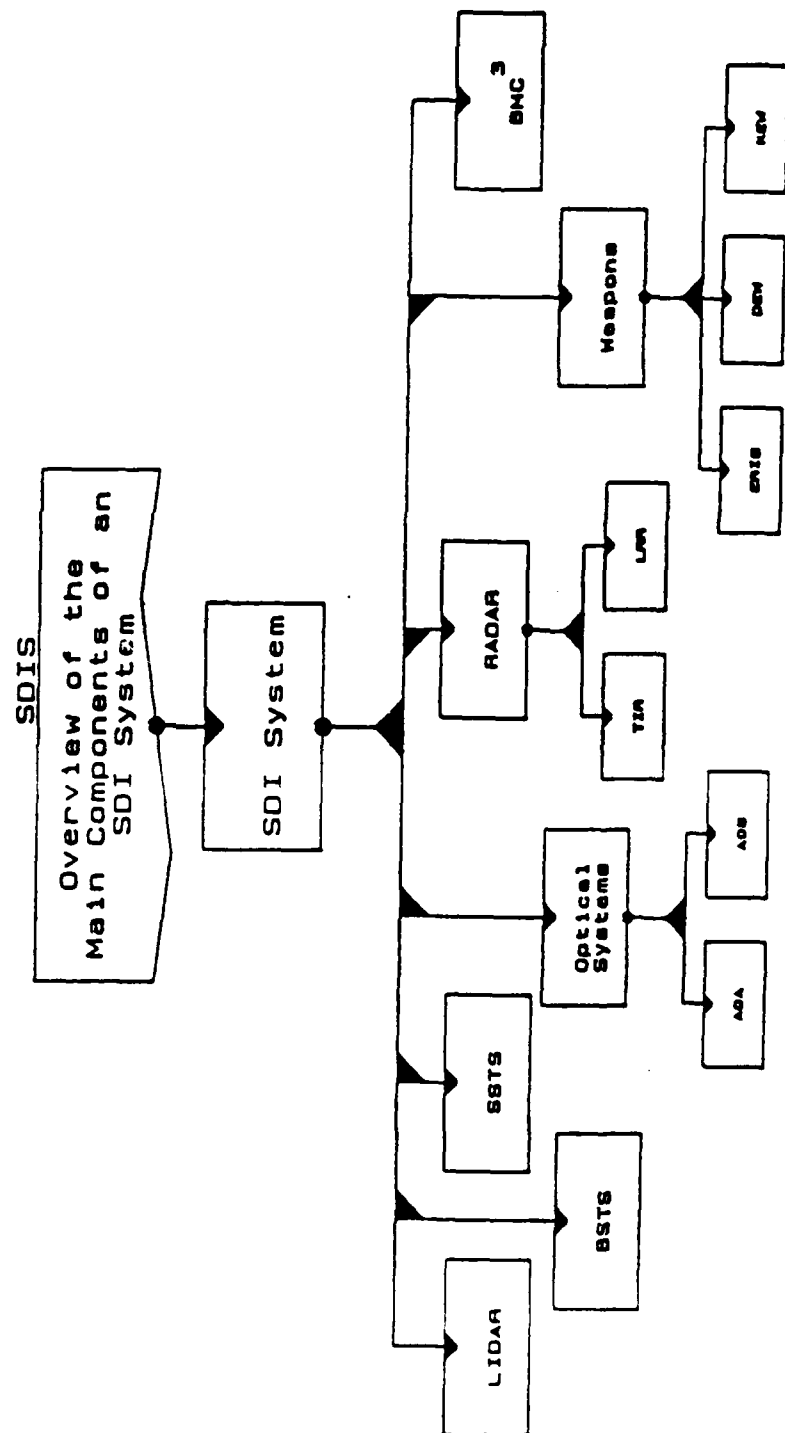


Figure 11e.1 : THREE LEVELS OF ABSTRACTION

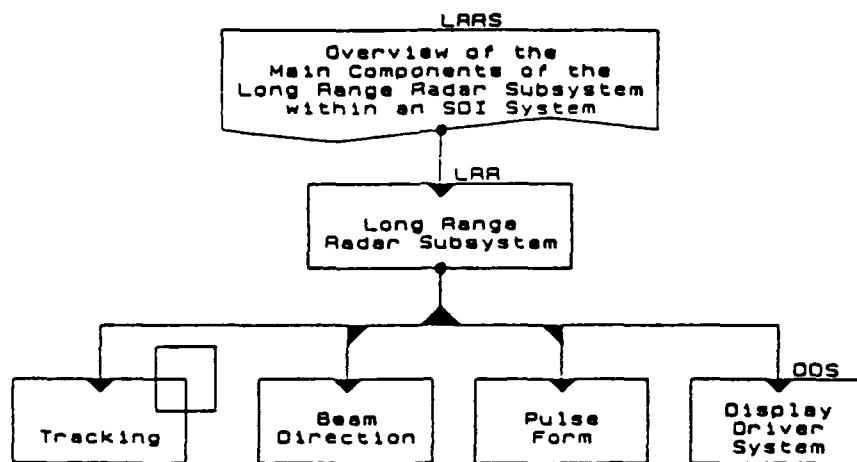


Figure 11e.2 : THREE LEVELS OF ABSTRACTION

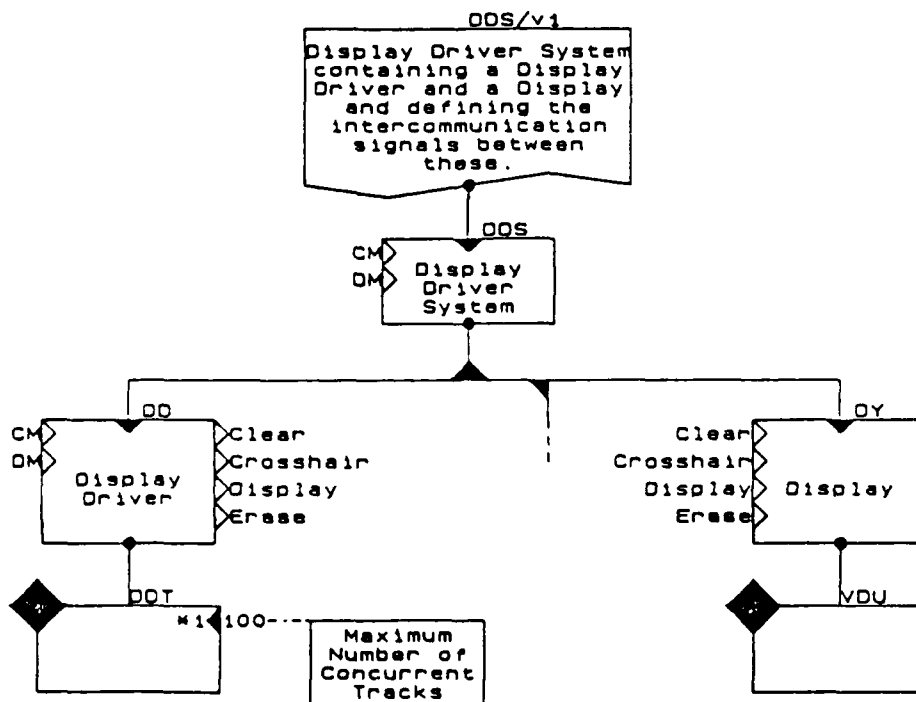


Figure 11e.3 : THREE LEVELS OF ABSTRACTION

Type checking is done whenever possible throughout the Auto-G toolset. A major part of the work of automatic checking within Auto-G is to ensure that an object's type is compatible with its usage.

Every Auto-G document has a user-supplied name and set of version descriptors that together form a unique document identifier. This labelling of documents is suitable for enforcing configuration management and build state controls during the development of systems.

A package of modules may be defined by creating an Auto-G 'configuration document' which lists the names and versions of all the components. When a module is changed, it is given new version descriptors. To produce a package that includes the changed module, a new configuration document must be created giving the new version identifiers.

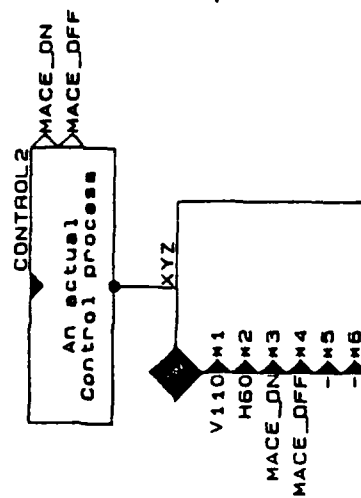
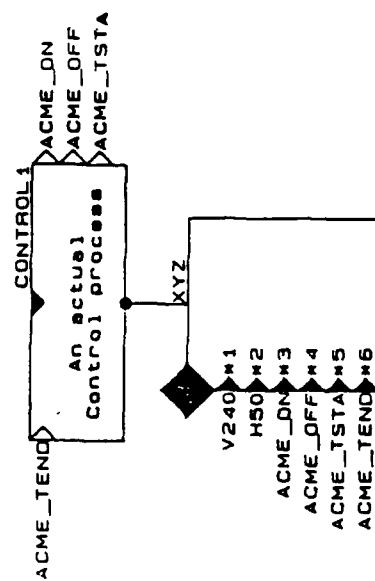
The G/T notation fully supports the concept of generics. Using the notation, it is possible to design generalized solutions to problems as 'design templates'. These templates can be used for creating real objects within a system. Values and identifiers used within the design of a template may be supplied by parameters so that their application can be as general as possible. The notation also supports generic operators and procedures.

Figures 11h.2 and 11h.3 show the use of a template within a system. In the examples, the template is a pattern for a function that controls electrical equipment. The template has six parameters, namely the voltage and frequency at which the equipment runs, the format of the M_ON signal, the format of the M_OFF signal and the formats of signals (TEST_START and TEST_END) used to TEST the equipment.

The equipment in Figure 11h.3 does not have a test function and so the TEST_START and TEST_END parameters are not used; those parts of the template that relate to self-test are redundant in this case. When a template is used, the Auto-G checker identifies any redundant design, removes it and checks what remains. Only the non-redundant parts of the design are used for automatic code generation.

The G/T notation supports the concept of meaningful text substitution macros. Using the notation, it is possible to define convenient synonyms for any name or expression. Auto-G checks that every macro is correctly defined, and that each use matches the definition.

Figure 1h.3 : USE OF A TEMPLATE



The G/T notation allows the designer to describe data structures of arbitrary complexity. Data may be global to the system, or may be local and private to part of the system. The characteristics of a variable can be declared precisely i.e. its type, its initial value, whether it is read/write or read only and what functions may access it.

Normally, data is carried between functional objects by signal. Functional objects may also read and write to external data if this is consistent with the way the data has been declared.

For any function, Auto-G allows the designer to define its communications interface - that is, the external data that is accessed, and the types of signal sent and received. Interface definition may be done at any level of abstraction (as shown in Figure 11j.1), and the consistency of the interface definitions at every level is checked automatically by Auto-G. The communication subsymbols make explicit the use and flow of data within the system. The flow of data can be made even clearer by joining the 'send' action in one function to the 'receive' action in another (as shown in Figure 11j.2).

The use of the G/T notation alone cannot adequately describe data flows because the system consists of concurrent functions. This is because a static analysis cannot determine what data values are passed around when several functions access the same variable. One of the outputs from Auto-X is information on how data is used moment by moment: from this information, Auto-X generates a trace listing showing the actual data flows within the system.

Auto-G allows the behavior of each concurrent function to be described in terms of a state transition network i.e. a set of discrete states and a set of transitions between those states, where transition is a sequence of actions triggered by some condition or event. The graphical representation of a network is very similar to a traditional flowchart.

The G/T notation permits design at any level of abstraction, and what is described as an action at one level may be decomposed into a network. In this way, the notation permits one network to be nested within another network. A typical use of nested networks is where a high level action requires communication with another function: a G example of this is shown in Figure 11k.1.

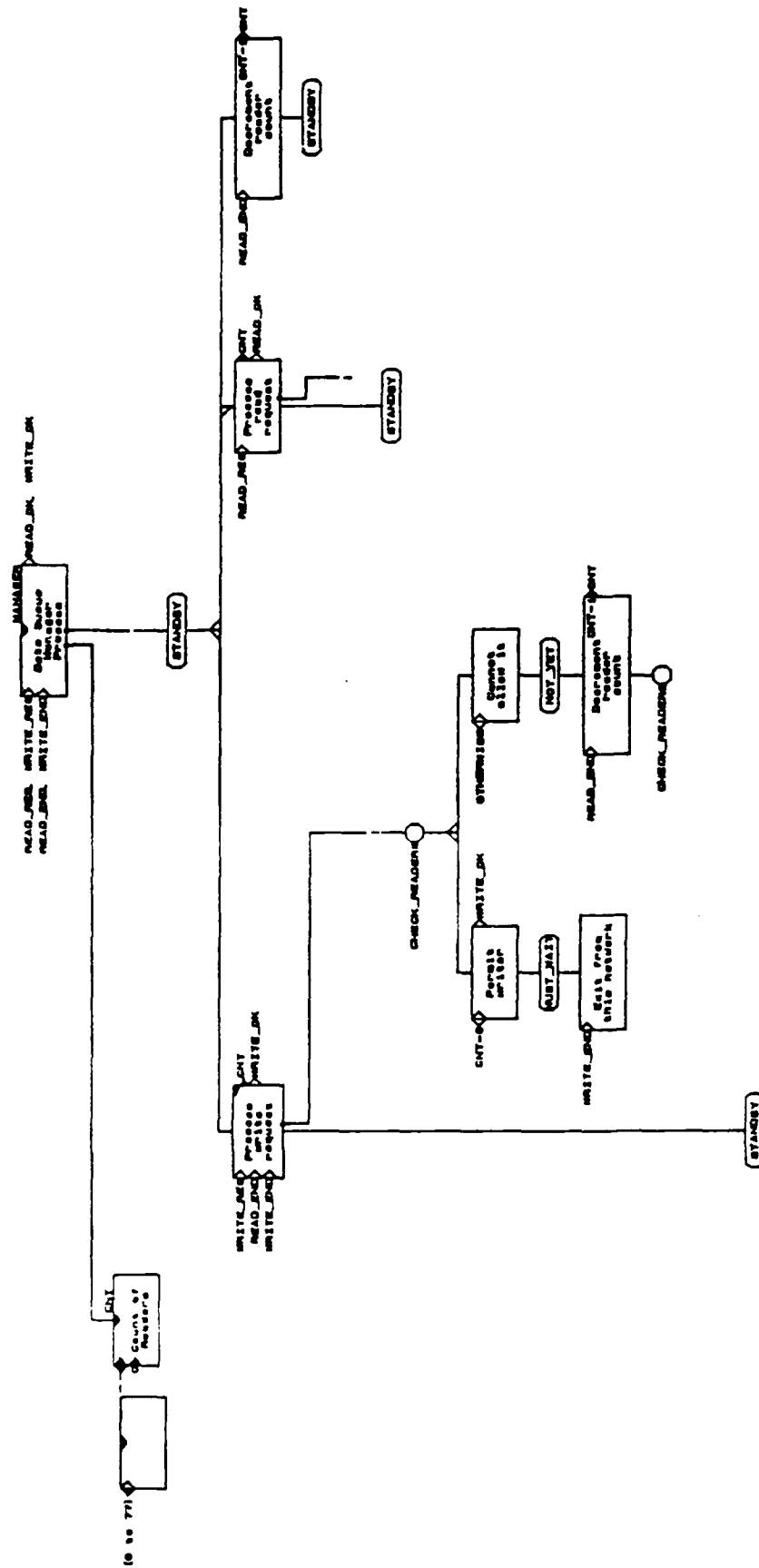


Figure 11j.1 : INTERFACE DEFINITION AT ANY LEVEL



12. Is there a paradigm embedded in your system? If so, describe it briefly.

The G.T notation is methodology independent and therefore does not postulate any embedded paradigm. This means that the front end part of the Auto-G toolset does not contain any embedded paradigm.

Auto-G supports the development of object-oriented systems consisting of asynchronous functional objects communicating only by asynchronous messages (with non-destructive send and destructive receive). For implementing systems of this sort, Advanced System Architectures have developed special hardware kernels which enable messages to be sent efficiently between functions with a processing overhead of the order of microseconds rather than milliseconds. This efficient message handling makes it practical to forbid the use of shared data. For users wishing to implement object-oriented systems, the back end of the toolset (i.e. the Auto-G checker, Auto-X and the Code Generators) currently has rules built in that restrict the use of concepts available in the notation to what can be supported in the target environment. It is intended to produce alternative versions of these tools supporting different target environments.

13. Describe the external tools with which your system interfaces (tool compatibility).

Auto-G has been designed with all the 'hooks' necessary to integrate it fully within project support environments in which additional facilities (such as configuration management and project management) are provided by other tools.

With help from Advanced System Architectures, customers have successfully linked the Auto-G database to other tools within their own proprietary development environments: one such tool is a text editor which guides programmers coding in JOVIAL; a second tool generates test data automatically using as a basis the interface descriptions within the database; a third tool automatically generates documentation to DoD standards e.g. DOD-STD-2167.

ASA has plans to link Auto-G directly to the ORACLE database for which other tools are also available. In addition, developers of Expertware, GENOS, and ISTAR have evaluated Auto-G with a view to including it within their products. ASA is monitoring the Portable Common Tools Environment

(PCTE) Initiative; It is hoped that this Initiative will propose a set of Interface standards that can be adopted by Auto-G.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

14.1 Flow Direction

Auto-G encourages a top-down approach to design since it allows any number of levels of abstraction. Auto-G is capable of expressing the concepts used in any of the popular structural design methodologies. Advanced System Architectures provides training in system design which uses a set of objective criteria to decide how to functionally decompose the system.

Alternatively, a bottom-up approach can be used in which small functions are designed in isolation and then put together to form a system.

Whether a pure top-down or bottom-up approach is used, or a mixture of the two, the Auto-G checker performs a complete check of the system and identifies aspects that might indicate a poor design.

14.2 Hierarchical Decomposition, Architectural Perspectives and Object-Oriented Design

The view facility described in the response to Question 10 allows the system to be examined from any architectural perspective. The G/T notation fully supports both hierarchical decomposition and an object-oriented design; examples of these aspects of the notation are given in the response to Question 5.

15. Is your system supported by formal syntax & semantics? Describe briefly.

Yes. For both the graphical and the textual man-machine interfaces of Auto-G, a formal abstract syntax and semantics exists in the form of an abstract language D which is invisible to the user.

Both concrete syntaxes (G and T) are translated into/from D by Auto-G; this ensures complete consistency between the graphical and textual representations and also makes two way translations possible.

The Auto-G checker, Auto-X and the Code Generators all operate on the D representation of documents.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

The purchase and maintenance cost of the Auto-G depends upon the number of simultaneous users of the system that are anticipated - this figure is normally less than the number of people who may use this system.

For each 'user', the purchase cost of Auto-G is \$15,000.00. The annual maintenance charge of an Auto-G system is priced at 15% of the full purchase cost. For large Auto-G systems with over eight simultaneous users, there is a significant quantity discount. Site licenses can be negotiated.

Advanced System Architectures provide initial training and on-going support in the use of Auto-G tools. The cost of attending a 4 day training course at ASA's offices in Camberley, UK is \$1,125.00. Courses can also be arranged on customer premises.

Prices for other components of the Auto-G toolset (i.e. Auto-X and various Code Generators) have not yet been fixed.

17. Indicate the hostability (measure of degree of portability) of your system.

Auto-G is available on any Unix or VMS system for use via terminals (preferably with a mouse) using the Regis graphics protocol.

The system can also be supplied for Sun, Apollo, VAXstation and Atari 1040 ST workstations. These workstations can be linked into other development facilities via a local area network such as Ethernet.

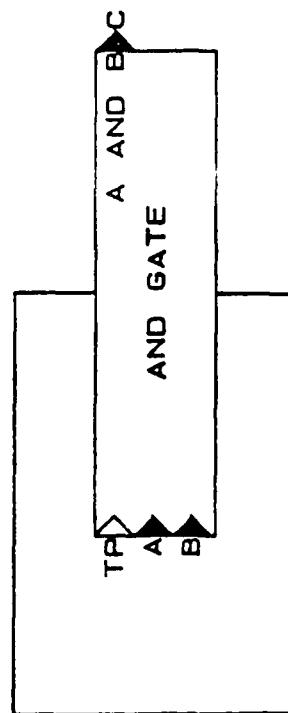
Auto-G produces hardcopy diagrams through plotters using the HPGL protocol. Laser printer output is also being developed.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

The G/T notation can describe the functionality of any system, not merely of a computing system. Auto-G provides a complete set of concepts which may be applied in all of the above applications.

As an example, Figure 18.1 represents in G a time-pulse triggered 'AND' gate. Figure 18.1 represents in G a time-pulse triggered 'AND' gate. Figure 18.1 contains both formal and informal parts: the text within the function

Figure 18.1 : A TIME-PULSE TRIGGERED 'AND' GATE



symbol (rectangle) is informal commentary text ("AND GATE"); all other text (TP, A, B, A AND B, C) is formal. The function implied by the symbol is that when a time pulse (TP) arrives, the values existing in the inputs (A, B) are read, combined by a logical AND (the expression "A AND B") and the result transferred to the output C. The function is repeated indefinitely.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

Auto-G may be used to support any structured design methodology. Advanced System Architectures recommends its own complete design methodology known as the Sofchip Design Principles. This methodology pre-supposes an object-oriented run-time environment which provides efficient message passing between functional objects. This methodology is taught by ASA on its training courses, and is fully described in ASA's training material.

Not applicable. If Auto-G is used to produce a detailed design, then program code (e.g. Ada) or executable target code may be generated automatically directly from the design. In order to generate the complete system code, the designer must design down to the lowest level of detail; some users consider that detailed design done using Auto-G is programming by another name.

The G/T notation makes it easy to structure a project as a number of parallel work activities.

Auto-G does not contain any project control facilities as it is intended to be used in conjunction with a separate project control system. Auto-G has the necessary 'hooks' to interconnect with any project control system.

The G/T notation makes it easy to manage the parallel evolution of a number of documents.

Auto-G does not contain any configuration management facilities as it is intended to be used in conjunction with a separate configuration management system. Auto-G has the necessary 'hooks' to interconnect with any configuration management system.

**20. Describe how your system supports a team development approach.
(Number of stations/users).**

The G/T notation supports decomposition of a system into a hierarchy of documents (containing statements or submodules), each of which may be developed independently of the others by a single user or by a small team working closely together. The notation makes explicit the dependency of one document upon other documents.

Auto-G is intended to be used in conjunction with a project configuration management philosophy - the G/T notation provides the means of implementing a management system making use of the access control facilities of the host operating system. If Auto-G is integrated fully into a sophisticated development environment, then there are no restrictions on how many users work simultaneously above those which the host environment imposes. Even without a sophisticated development environment, documents may be worked on in parallel provided that development work on each document is done in a separate user directory. A document under development may be checked in conjunction with other documents at any stage provided read access to other documents is permitted.

For use on large projects, one Auto-G customer is developing an infrastructure in which Auto-G workstations are networked into a large host computer which acts as a file-server, configuration manager and database machine.

21. Describe how your system supports design trade-offs.

It is intended to enhance Auto-G so that it provides assistance in calculating various design metrics. With this additional functionality, it will be impossible to compare alternative design solutions and to select the most appropriate design for the problem environment. In addition, the metrics for the chosen solution can be used to dimension the system precisely.

22. Indicate the range of problems to which the system can be applied.

The G/T notation is a complete formal notation that can be used to describe all functional aspects of any system - including functions that are electrical, electronic, mechanical, hydraulic, computer hardware and software, or human. The G/T notation is particularly suitable for describing large complex real-time systems. Auto-G makes it easy to use the G language, and provides automatic checking of requirements, specifications and designs.

Auto-G has been used successfully to develop systems in the following application areas:

- telecommunications and digital switching
- civil and military command and control
- industrial process control
- avionics and aerospace systems
- limited resource management
- application specific integrated circuit (ASIC) design

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

Mr. G. Bryant
GEC Avionics Limited
Airborne Display Division
Airport Works, Rochester
Kent ME1 2XX, United Kingdom
Telephone +44 643 44400 ext. 3237

Mr. A. Bosman
Phillips AT & T
Warrandeborgstraat 8
PO Box 18, 1270 AA Hulzen
The Netherlands
Telephone +31 35.87.48.90

Mr. J. Davis
British Aerospace plc
Bracknell Division
Downshire Way Bracknell
Berks RG12 1QL, United Kingdom
Telephone +44 344 483222

Frau B. Kuck-Wutzke
MBB ERNO, Dept. RBO22
Raumfahrttechnik GmbH
Hunefeldstrasse 1-5, D-2800 Bremen 1
West Germany
Telephone +49 421 539 4211

Mr. S. Taylor
Thorn-EMI Electronics Limited
Computer Systems Division
Wookey Hole Road, Wells
Somerset BA5 1AA, United Kingdom
Telephone +44 749 72081

Jodfrey Associates Inc.
462 Highfield Court
Severna Park, MD 21146

Jodfrey Associates, Inc.

1. Describe how your system supports early detection of inconsistencies, closure and errors.

ProMod supplies many balancing and checking functions to ensure completeness and consistency. The manual for ProMod "A Complete Description" may be consulted for some of the checks made by ProMod. Please see - Syntax Checks During Creation, Structured Analysis Analyzer, Modular Design Analyzer, and Pseudocode Analyzer.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

ProMod supplies the number of objects contained in the project library. This would include the number of data flow diagrams, data items, etc., that have been defined thus far.

3. Describe how your system supports documentation, program management and control.

ProMod does not have facilities for Gant charts etc., to support project management. However, there is a wide range of high-level and detailed level reports that can be pulled to allow monitoring of progress. For details on these reports see "A Complete Description" - Documenting and Reporting.

4. Describe how your system supports real time design.

ProMod offers the methodology of Derek Hatley/Lear-Siegler including control flow diagrams, control specifications, and state transition diagrams. See Real-Time in "Complete Description."

5. Describe how your system supports concurrency, parallelism.

The same structures found in Ada (task, package, etc.) may be used in the ProMod analysis and design phases. This is amplified in the real-time design extension.

6. Is your system constrained to a particular implementation language (Ada)?

No. ProMod does not constrain the user to any particular implementation or language.

7. Does your system produce Ada PDL?

The current implementation of ProMod consists of three phases: requirements definition, architectural design, and detail design. During detail design the engineer will identify input and output parameters and specify the logic of the algorithm for each functional component of the system. The logical specification is written in a semi-formal language patterned after that of Cain, Farber and Gordon. Parameter strings will be checked by ProMod to ensure they are of proper length and type and that they match entries contained in the ProMod data type dictionary. Use of input/output data within the context of the algorithm specification will also be checked for consistency with the parameter string declaration. The algorithm will be checked for correct syntax and to ensure that all data used have been declared appropriately. Once a valid specification has been entered, ProMod will convert it to Ada PDL using an Ada Code Frame Generator. The Ada Code Frame Generator translates parameter strings to valid Ada data declarations and the algorithm specification to syntactically correct Ada code. The programmer will need to elaborate the Ada PDL in areas indicated by ProMod and supply lower level details not found in the specification. The ProMod Ada Code Frame Generator is currently available and is being used by several Department of Defense contractors.

8. Describe how your system supports life cycle intraphase & interphase communications.

ProMod is the only commercially available tool that integrates the analysis, architectural, and design phases. ProMod transforms the conceptual analysis model to design packages. The architectural design phase allows for and checks the consistency of detail design procedures. See "A Complete Description" - The Transformer.

9. Is your system automated, executable, compilable?

The system is sold as an executable package for the VAX and IBM PC.

10. Describe the graphics support for your system.

The analysis phase is fully supported by a graphics editor. See "A Complete Description" - Objects Data Flow Diagrams.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics
- Macros
- Data flows
- Control flows

Formal specification languages may be used during the analysis phase to create operational specifications.

Any part of the analysis, design, or code features may be exported and used in other systems.

ProMod uses the design methodology of Parnas. The packages created in the architectural design may share procedures but not data. Disembodied data are not allowed to flow through the system design.

The primary tool of design used by ProMod is the package or module. It is equivalent to the Ada concept of package.

ProMod models the system with data and procedural abstractions at the analysis phase. During design, objects and procedures are encapsulated in ProMod packages.

ProMod supports a data type dictionary. It also checks for type consistency in package interactions.

ProMod offers features to support phased development and integration of design from various sources and levels of abstraction.

The use of generics is encouraged by ProMod. Actually every ProMod package is closer to a generic specification than a specific instantiation.

ProMod can easily be used to develop the definition of macros or any other sort of programming abstraction.

See "A Complete Description" - Objects Data Flow Diagrams

See "A Complete Description" - Real-Time Analysis

12. Is there a paradigm embedded in your system? If so, describe it briefly.

ProMod is based on the use of Structured Analysis (DeMarco), Modular design incorporating abstraction and information hiding (Parnas) and program design based on a generic Calne, Farber and Gordon PDL, and on several Ada PDLs including Ada itself, Ada-DL, and Key One.

13. Describe the external tools with which your system interfaces (tool compatibility).

ProMod produces ASCII files of all objects and reports. They may be included in word processing packages, documentation packages, project management packages and any other tool capable of an ASCII interface.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

ProMod supports several design methods including hierarchical and object-oriented. See "A Complete Description" Modular Design.

15. Is your system supported by formal syntax & semantics? Describe briefly.

ProMod does have a formal Syntax for each object. This is provided in a BNF style in the ProMod User's Guide.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

Total system on PC - under \$10,000

Total system on Vax - \$20,000 - \$60,000 (depending on options and type of Vax)

Training - \$1,000 dollars per day plus expenses (5 or fewer days depending on familiarity with methodologies)

Complete pricing data is available upon request.

17. Indicate the hostability (measure of degree of portability) of your system.

ProMod is available on any IBM PC, and all DEC Vax systems. The system is completely portable between Vax and/or PC systems.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

ProMod is a tool built for systems abstractions. Clients currently use ProMod to model computer boards, and subsystems as well as programming abstractions. It is a generic systems modelling tool, with a target of programming languages.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

ProMod is a complete software development methodology. This includes requirements analysis, architectural design, detail design, and programming (via code frame generators).

20. Describe how your system supports a team development approach. (Number of stations/users).

ProMod is primarily used in a team environment. This includes many users on a single VAX, many PC users, and most often, on the VAX with PCs used as work stations. The tool is not limited to any certain number of users.

21. Describe how your system supports design trade-offs.

ProMod will perform consistency and other error checking to make sure the change is truly a "trade-off." It will also provide reports for assessing the impact of the proposed change.

22. Indicate the range of problems to which the system can be applied.

ProMod is usable on any system/software design problem.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

Dan Roy
Century Computing
8101 Sandy Spring Road
Laurel, MD 20707
301/953-3330

Barry Gillis
Easton Corp.
6564 Loisdale Ct.
Ste. 900
Springfield, VA 22150-1872
703/922-5600

Lee Trent
Teledyne
19601 Nordhoff St.
Northridge, CA 91324
818/886-211 Ext. 2634

Pat VanMunn
Measurex
1 Results Way
Cupertino, CA 95014
408/255-1500

Gregg Reed
Simmonds Precision
Panton Road
Vergennes, Vermont 05491
802/877-2911 Ext 2578

Westinghouse Electric Corporation
Defense Group
Friendship Site
Box 1693
Baltimore, MD 21203

Westinghouse Electric Corporation

1. Describe how your system supports early detection of inconsistencies, closure and errors.

The SIDE Facility supports early detection of inconsistencies, closures, and errors both explicitly in its design and implicitly by its implementation. The SIDE Facility is based on a graphical methodology that enables users to visually check for inconsistencies and completeness. The underlying program also checks for type consistency and errors while creating/modifying an ADM graph. This checking is further enhanced through use of an AI language and frame representation of the data objects. This type of representation lends itself easily to a logical checking of a graph as it is being created/modified. Figure 1 (on the following page) depicts an example frame representation.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

The user may check graph completeness at any point of graph creation and modification. At this time, the program will list any unconnected entities, as well as any unset attributes that are necessary for graph completeness. This information could easily be used to generate a quantifiable measure of completeness. This capability could also be extended to several levels, to not only check completeness at the graph level, but to check if underlying (child) nodes of the graph exist in the library or must be either created or modified to match the abstract node.

3. Describe how your system supports documentation, program management and control.

The SIDE Facility supports documentation through several methods. A plotter is used to generate a picture of the graph, including its nodes and connectivity. The facility also produces tables that describe each of the nodes and the queues that connect them. The early implementation also outputs the Prolog database for the given graph.

4. Describe how your system supports real time design.

One of the driving factors behind the SIDE facility has been to provide a state of the art real time design facility that supports the ADM design methodology starting at systems requirements and transcending the development path through module or processor design. The underlying data flow methodology implicitly supports real-time development. Attributes have been

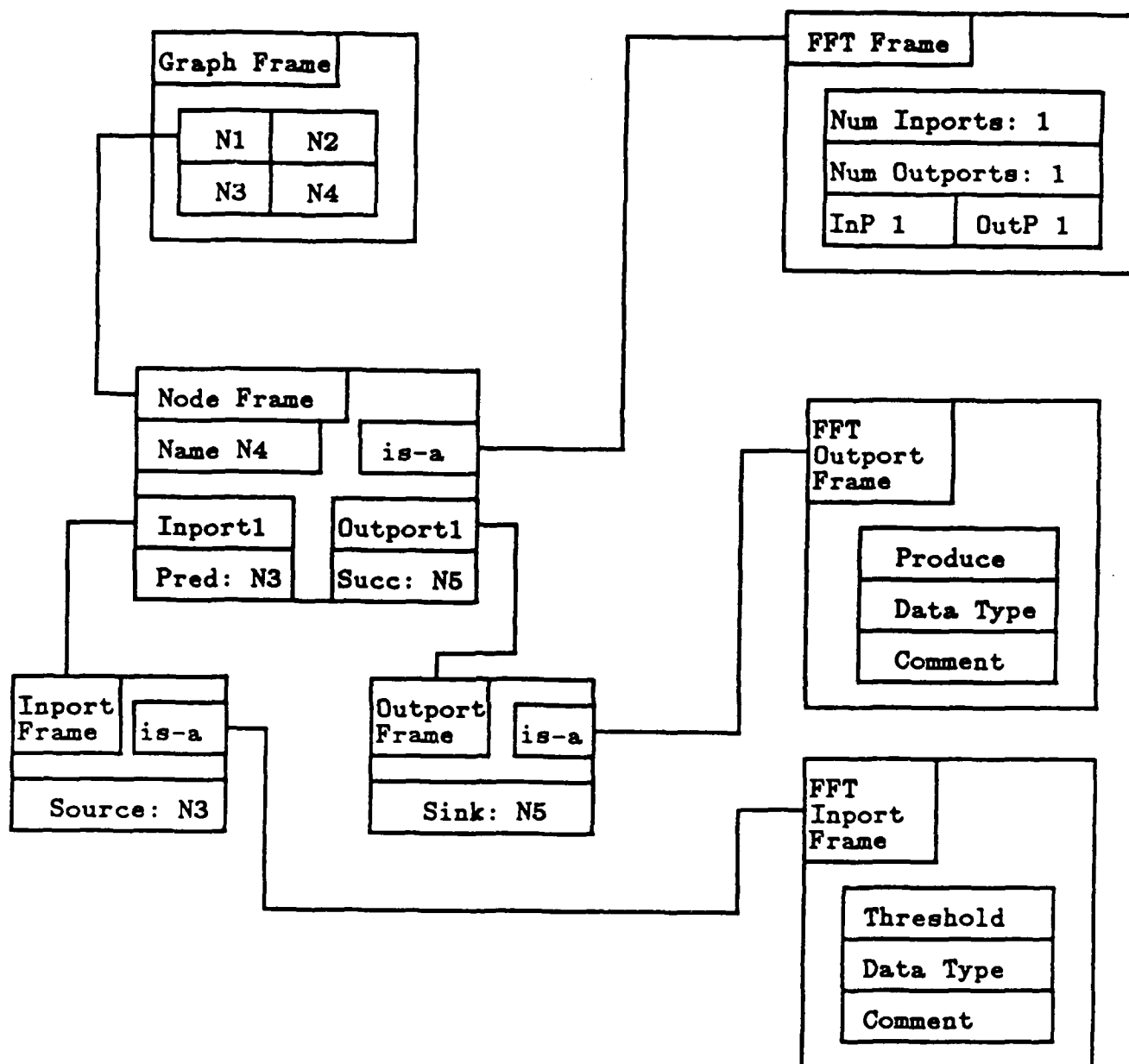


Figure 1. A frame representation of a graph containing 4 nodes points to frames that contain information on each of the nodes in the graph.

predefined but the user may add any attributes that define his system. These attributes are used to describe parameters such as input data rates, the flow of data between nodes, and the type of processing at each node. A graph simulator can be used to verify that the defined graph meets real-time requirements in a single or multiprocessing environment.

5. Describe how your system supports concurrency, parallelism.

Parallelism is an implicit feature of the Application Description Methodology with its data flow concept. Nodes represent processing elements that are defined independently and may execute independently of each other. A node is ready to execute when each of its inputs have data equal to or greater than the threshold attribute for that input.

ADM also supports concurrency in graph definitions. For example, node A may provide input to node B at a rate faster than node B can process. To solve this problem, no copies of node B execute concurrently. The data output from node A is switched after each execution to a copy of node B. In a circular fashion, and then recombined by a node that receives an input from each of the node B copies.

6. Is your system constrained to a particular implementation language (Ada)?

The Application Description Methodology is a graphical description methodology that is language independent. It uses graphical constructs coupled with textual descriptions to define a process. The database representation is then used to generate Ada specifications that can be compiled and executed. This language was chosen to comply with DoD initiatives because of its data typing and constructs but the data base could be used to generate specifications for another language.

7. Does your system produce Ada PDL?

Yes.

8. Describe how your system supports life cycle intraphase & interphase communications.

Because of the hierarchical structure of ADM, interphase life cycle communication support is provided by the multi-levels of a graph and its node. If a change is made to a node at the top-level, this change will be reflected at the Ada PDL description level and then at the detailed level underlying the node definition. Intraphase communication is supported by definition of

ADM. If a node's attributes are changed, any nodes connected to the node must also be modified if their interconnect is affected.

9. Is your system automated, executable, compilable?

A main objective of the SIDE facility has been to increase engineers' productivity and efficiency through the use of automation. Once a graph is completely defined, Ada code can be automatically generated from the graph description. This code is valid and can be compiled and executed in any MIL-STD-Ada environment.

10. Describe the graphics support for your system.

A driving thrust behind the SIDE facility has been to use graphics to provide a user friendly environment. This is done implicitly through the use of ADM, a graphical representation language. In the earlier versions of SIDE, a color graphics terminal was used as the interface to the SIDE facility. Pop-up menus and actual graph drawings were a main feature of the program. This system is now being upgraded for a LISP machine environment. The old features will be included along with multiple windows, and the use of user aids such as a mouse will be supported.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics
- Macros
- Data flows
- Control flows

Figure 2 (on the following page) shows the overall SIDE facility concept.

ADM (Applications Description Methodology) is a graphical representation language used by a systems engineer in the development of systems. The engineer uses ADM in both the requirements and the design phases of system development. The need for a graphical-based representation has developed from the limitations of purely textual languages. Graphical languages provide

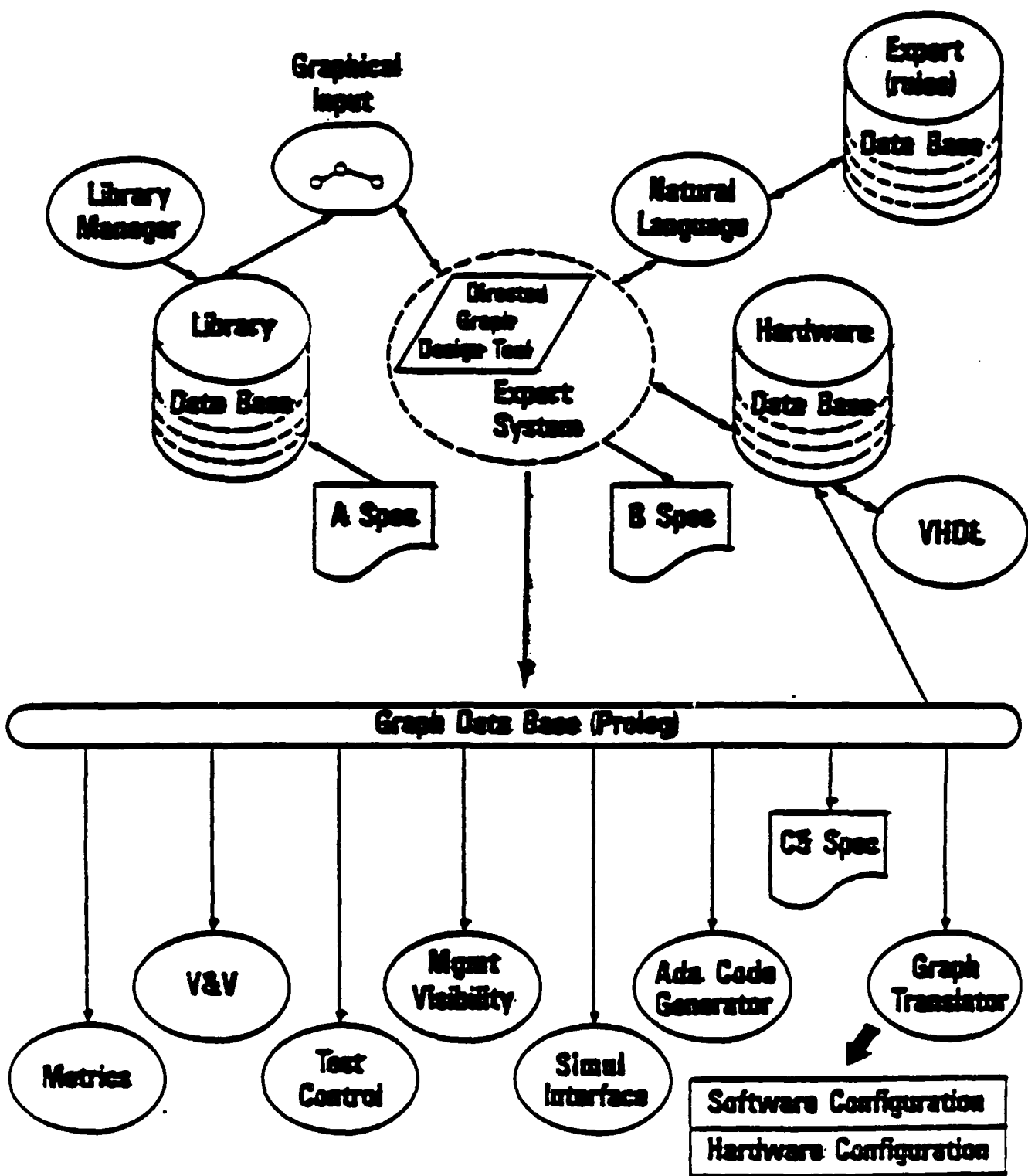


Figure 2. The overall SIDE concept consists of reusable libraries and uses a graphical input based on ADM to create the central database.

both the designer and the user a medium for representing structure coherently and with varying levels of detail.

ADM's basic structure resembles that of a data flow diagram. It consists of nodes and arcs. The nodes represent functions and/or objects, whereas the arcs depict the connectivity and flow of the graph. Each node must represent an executable module or a subgraph. A subgraph also contains nodes and arcs, again allowing its nodes to represent executable modules or subgraphs. A graph level is defined when every node is associated with a subgraph or an executable module. The system graph is complete when every level of the graph is defined.

Since graphs represent flow, they also depict system structure. Textual languages are not good representations of structure, however, they are good representations of detail. The nodes of a graph are only abstractions of the functions they represent, hence, each node will have associated text describing the node. This association allows users to view and comprehend all levels of the graph. If the text associated with a given node does not provide sufficient detail for the user, s/he can examine its subgraph.

ADM facilitates the ability to design and review in a top-down manner. The structure of the graphs allow both the designer and the reviewer (user) to approach the graph with varying levels of detail. The top-level graphs generalize the processes and data transformations *within the system*, whereas the lower-level graphs represent the increased detail of the system. Parsing the graph provides increasing detail of the system processes. Therefore, the graph incorporates and provides the features of:

- information hiding
- hierarchical structure
- separation of concerns
- top-down/bottom-up design

Unlike pure textual languages, ADM directly provides a methodology for step-wise refinement. The designer may begin system development by specifying processes as high-level abstractions, then further refine the design via the sub-graphs. The user also benefits from this development approach, because s/he can review the system at each level which it was developed. Parsing the nested structure of the graph provides quick access to desired levels of detail. Reviewing the text associated with the desired node provides descriptions of the node's characteristics and an explanation of its use. The text provides the developer the requirements of a node's subgraph. It also provides the user an overview of the underlying sub-processes, without the detail of the actual structure.

Many system designers utilize data flow diagrams during development. Using Computer-Aided Graphics, ADM can simulate this development methodology. ADM's basis is data flow representation. Introducing control flow in a concise, simple manner. Introducing the constructs for control flow provides two benefits:

- providing control flow which is documented and relatively easy to use.
- ensuring a standard usage of the constructs to increase understandability among users.

ADM not only allows the developer to design systems using standard constructs, but also tests new configurations to ensure they follow the rules for creating graphs. The graphical structure of the language aids in ensuring completeness, consistency, and connectivity of the system. Once a system is designed using the ADM methodology, the system engineer does not have the burdensome task of checking for consistency, completeness, and connectivity as s/he may with a textual specification.

Using a CAD facility provides quicker design and access than conventional hand-written approaches. The CAD facility is capable of storing the structure of the graph, nesting information, parsing algorithms, organizing text, portraying the reconfigurations, and other data base related tasks. With the relief of these responsibilities, the designer can devote more of his/their efforts toward the actual design process.

ADM allows its components to be created and modified independent of each other. This principle, 'separation of concerns', allows the system to be organized and divided into separate units. It provides greater flexibility since its modifications are well-confined. It also provides the essential groundwork for security among development. Nodes which incorporate classified information can be developed separately and refuse examination to anyone without access privileges.

The ADM methodology is implementation independent. That is, ADM does not restrict which language the node implementations are written. The lowest level nodes are abstractions of the executable modules. The executable modules are object modules written in any source language chosen by the engineer. Hence, ADM does not depend upon the knowledge of any given language to develop systems. ADM, due to its graphical basis, can be understood by programmers as well as non-programmers. This aids in closing the communication gap between technical and management personnel. As a result, walk-throughs and design reviews can become more productive.

ADM assumes the lowest level nodes to be associated with executable modules. The code which links and interfaces between the modules will be generated by the ADM language. Originally, the interface code will be generated in Ada, however, this is not to imply Ada is the only language which ADM supports. The Ada code generated by the ADM language can be manipulated by a common editor. This provides the flexibility and strength of the Ada language which ADM may not provide. By generating the interface code between the modules, the probability of error associated with human-generated code will disappear.

To facilitate the collection of executable modules, ADM will link to a reusable library of modules and routines. This library will be accessed by a "library assistant". The assistant will examine the requirements of the needed routine specified in the ADM graph, then proceed to find a suitable match within the library. The prototype to be developed for the October 1986 IR&D presentation will not include the library assistant. If support for the project continues in 1987, the assistant can be developed.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

The paradigm embedded in the SIDE facility was developed at WEC and called Directed Graph Methodology (DGM). This methodology was based on pure data flow and therefore Application Description Methodology has expanded on DGM to include control flow.

13. Describe the external tools with which your system interfaces (tool compatibility).

The SIDE facility interfaces to the following tools:

- Ada compiler.
- ECSS II System Simulator
- 1750A Simulator
- ADM Graph Simulator

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

The SIDE facility supports hierarchical decomposition by definition of the graphical based methodology. As mentioned earlier, ADM supports top-down as well as bottom-up design. The designer can create high-level nodes first, then proceed to decompose them into respective sub-tasks, namely top-down design. The designer can also select the desired low-level routines and link

them to create the high-level processes, namely bottom-up design. Allowing both directions of approach provides the flexibility most designers require. In reality, most designers incorporate both approaches when designing a system. There are no limitations on the graph representations since the user can create his own attributes to describe a given system. ADM was originally targeted for real-time multiprocessing applications, but the nodes can be used to represent any type of object and the attributes needed to describe that object can be defined by the designer.

15. Is your system supported by formal syntax & semantics? Describe briefly.

The ADM language consists of few constructs, providing beginners the ability of quick use without hours of learning the syntax. The language is broken into four basic constructs, consisting of eight distinct icons:

Data Node Types (see Fig. 3)

- Solid Circle - Internal Data Transformation
- Solid Square - External Data Transformation

Control Node Types (see Fig. 4)

- Hyphenated Circle - Internal Control Transformation
- Hyphenated Circle - external Control Transformation

Flow Types (see Fig. 5)

- Solid Arrow - Data Flow
- Hyphenated Arrow - Control Flow
- Circle w/Valve - Flow Control Valve

Data Storage (see Fig. 6)

- parallel Lines - Data Store

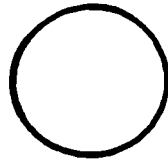
The following sections describe each of these construct types.

Data Node Types: Data transformations can produce only data. A pure data transformation, consisting of both input and output, is depicted as a circle drawn with a solid line. It must consume input and produce output. The input consumed can be either data or control signals. The output produced must be only data. These data transformations are considered internal because they rely on input and produce output.

If a node either produces just data output or consumes any input, but not both, then it is considered an external data node. An external data node is depicted as a square drawn with solid lines. As mentioned, external data

ADM CONSTRUCTS

Data Node Structures

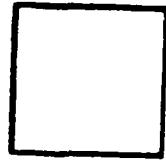


Solid Circle:

internal data transformation
i.e. number crunchers
device controllers

restrictions:

must accept either signal(s) or data
& produce only data



Solid Square:

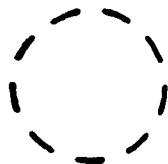
external data devices
i.e. sensors, weapons

restrictions:

must accept input or provide output,
but not both

Figure 3 - Data Nodes

Control Node Structures

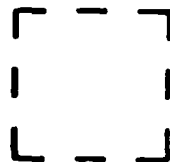


Hyphenated Circle:

Internal control transformation -
(control logic node)

restrictions:

must produce a signal & accept an
input (signal or data)
may produce data



Hyphenated Square:

external control construct
i.e. switch, timer error handler

restrictions:

must produce signals, may produce
data, no input allowed

Figure 4 - Control Nodes

Data/Control Flow



Solid Arrow:

Data Flow -

discrete values produced by any of the above constructs are queued (if necessary)



Hyphenated Arrow:

Control Flow -

values sent controlling execution will have signal types:
* event - immediate recognition
* synchronization - processes wait for other processes to "catch up"

Valves



Circle containing a Valve:

Control Valve -

control structure placed on data arcs enabling or inhibiting data flow

Figure 5 - Flow Constructs

Data Storage/Buffers



Parallel Lines:

Data Store -

Storage area for generated variables and access area for global variables

Figure 6 - Data Storage

nodes differ from internal data nodes (transformations) in they must not contain both input and output, whereas internal data nodes do.

Any data node, either internal or external, has the same associated attributes. The user need not know the value for each attribute upon initial design. S/he has the flexibility to assign a value to any attribute at any time during the development process. The default attributes with a data node are:

- Name
- Textual description
- Priority
- Delay
- Forcer
- Subscript
- Atomic
- # data imports
- # data exports

Each node of a graph represents a library module or a subgraph. Every node must have a name indicating the operations the node performs. If two nodes should perform the same operations, they may have the same name.

However, it is not recommended that two nodes have the same name if they perform *different operations*.

Data nodes may have data input. The number of data input parameters is set by the user. Associated with each Data Import will be its associated read attributes. These import values too must be set by the user, but will assume default values if not set.

Internal Data nodes, as well as external data nodes without input, must have data output. The number of data output parameters is set by the user. Associated with each Data Export will be its associated write attributes. These export values too must be set by the user, but will assume default values if not set.

Associated with each data transformation will be data input. Each data input parameter in a node has its own associated attributes. The default attributes include:

- Name
- Threshold
- Consume
- Read

Read_algorithm
Initializable
Datatype
Valve

Control Node Types: All control nodes must produce control signals and may produce data. Internal control node accepts both data and control signals as input. It is depicted as a hyphenated circle. An external control node does not accept any input. It is depicted as a hyphenated square. A control node is used to control/direct the flow of data and control the execution of specific nodes.

Signals can simulate interrupt operations and synchronizations between processes. They can be activated as well as deactivate processes. If a node is deactivated, its associated input data may be deleted or saved. A deactivation signal sent to a node transmits its signal to the nodes associated data input valves. Hence, a deactivation closes the input valves of a node. Associated with a deactivation is its save state. If it is true, the data on the input data queues is saved. If it is false, the data on the input data queues is deleted until the save value is reset to true.

Control nodes have the same attribute fields as data nodes, with an additional field of control signals produced. Control Signal attributes are described via the control arc attributes. The control arc attributes are discussed in detail in the below section - Control Arcs.

Flow Types: Data arcs are simply names associated with the queues which collect discrete data elements along a path. The attributes associated with data arcs designate the source and destination points of the arc, as well as the capacity of this queue. Hence, the default attributes associated with a data arc are:

Name
Source - Export
Destination - Import
Capacity

Some examples of MIDSS "experts" include advisors in constructing operational algorithms, software mapping experts to map the operational nodes onto Ada procedures, and hardware mapping experts that map the Ada procedures, and hardware mapping experts that map the Ada procedures onto different hardware configurations. There may also be experts to address software reusability and to perform simulations.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

The utilizations costs are TBD.

17. Indicate the hostability (measure of degree of portability) of your system.

The first version of the SIDE facility was implemented on a VAX 11/780. The graphics user-interface was written in Ada and the data base frame representation was written in Prolog.

The current implementation will be completed on a LISP machine, using common LISP and an expert system building tool written in LISP. This version will ultimately be hosted in the VAX environment.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

An ADM graph can be used to represent any level of abstraction. The node and arc attributes are then used to define the graph level. One objective of the SIDE facility is to integrate systems design by mapping the higher level node abstraction onto both the software and hardware modules in order to configure a complete system.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

At the present time, the SIDE facility embodies a development, design and programming methodology. It does not currently embody a project control or management methodology, but it can be extended to include these also.

20. Describe how your system supports a team development approach. (Number of stations/users).

The SIDE facility supports a team development approach both physically and logically. A central data base is created that is accessible by a number of

users. The data can be made read only except to privileged users to protect data integrity.

Logically, a defined graph consists of several nodes, which can be independently developed. Protections may be set so that the designer of a given graph node has complete access to the node data base and only read options for the other nodes in the graph.

21. Describe how your system supports design trade-offs.

Once a graph is completed, a graph or ECSS II simulation can be performed to measure system performance. It is then easy to change a node or its characteristics in the facility and have that change filter through the design. Simulations is then used to see the effects on the system.

22. Indicate the range of problems to which the system can be applied.

The SIDE facility has been targeted to support real-time multiprocessing applications, but the methodology is not limited to these applications. For example, the SIDE system can be used to describe multiprocessing architectures by using nodes to represent hardware modules, and the arcs to represent data and control buses. The node and arc attributes would then describe processor and bus characteristics, respectively.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

The SIDE facility has not currently been commercialized and marketed.

TASC-The Analytic Sciences Corporation
Rosslyn Office
1700 N. Moore St., Suite 1220
Arlington, VA 22209

TASC-The Analytic Sciences Corporation (AdaGRAPH)

1. Describe how your system supports early detection of inconsistencies, closure and errors.

A wide variety of design inconsistencies and errors are prohibited by AdaGRAPH's (AdaGRAPH is a trademark of The Analytic Sciences Corporation) construction rules. No direct support for closure is provided.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

Completeness checking of AdaGRAPH designs, either an entire design or any subtree of the design, is accompanied by a percent completeness metric. This metric is based upon all attributes of AdaGRAPH design objects.

3. Describe how your system supports documentation, program management and control.

Documentation is supported by the production of a variety of reports:

- o graphical and textual module maps
- o design graphs
- o completeness analyses
- o global and local data dictionaries
- o Ada PDL
- o various other textual views of modules (requirements, specs, . . .)

Program management and control may be enforced by the definition and use of predefined Ada data types and program unit idioms. Any policies may be checked by means of the data dictionaries within AdaGRAPH. Management metrics may be tracked in management views of modules.

4. Describe how your system supports real time design.

Real-time design is supported by directly graphically modeling the tasking aspects of Ada, and by representing the interaction of an Ada system with the operational environment in real-time terms (e.g., external interrupts).

5. Describe how your system supports concurrency, parallelism.

AdaGRAPH supports parallelism and concurrency by graphical rendezvous attributes within the graphical editor, and by translation of processes and their attributes into Ada tasks.

6. Is your system constrained to a particular implementation language (Ada)?

Yes, at present AdaGRAPH supports Ada; however, it could be used as a design tool for other languages which support strong typing and tasking (such as Concurrent C).

7. Does your system produce Ada PDL?

Yes. It produces an Ada PDL automatically from a graphical system design. The PDL produced is compilable and executable.

8. Describe how your system supports life cycle intraphase & interphase communications.

AdaGRAPH is intended to assist in the capture of requirements by dataflow techniques, and to support the high-level and detailed design phases which follow by allowing the user to traverse the dataflow diagrams annotating the design components with ever more detailed software information (such as call decisions, data types, etc.). In this way the transition between the three phases may be arbitrarily defined by the user. Transition to coding occurs at the direction of the user.

9. Is your system automated, executable, compilable?

Yes; AdaGRAPH is automated, and it produces compilable and executable textual Ada PDL.

10. Describe the graphics support for your system.

AdaGRAPH is based upon the Graphics Environment Manager (GEM is a trademark of Digital Research, Inc.) which is similar in appearance to the Macintosh interface. It is a mouse-driven system, using drop-down menus, icons, and keystroke inputs.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics
- Macros
- Data flows
- Control flows

AdaGRAPH is a process-oriented design system for Ada. As such it has direct support for data flows and control flows as arcs within designs. Control and data flows are not distinct arcs: a control flow is represented as a parameterless data flow, or as the call sequence attribute associated with a data flow. Data flows represent the passage of data values in the system.

Similarly, typing is represented as an attribute of data flow labels, that is the named data being transferred from module to module in the system under design. Typing is supported by the selection of a To-Be-Determined (TBD) typemark. Users may add types on the fly to the library packages.

Abstraction is supported by the abstractional features of Ada (e.g., private types), as well as by the use of higher-than-Ada-level program unit idioms. (A task idiom for interrupt handling abstracts the issues associated with interrupts at a level inexpressible in Ada.) Process and procedural abstraction are supported graphically.

Information hiding is supported by a layered representation of design: AdaGRAPH models a system as a hierarchical directed graph. It is also supported by methodological guidance.

Software reusability is supported in AdaGRAPH through reusable, extensible program unit idioms and by means of the extensible catalog of library units.

AdaGRAPH supports prototyping approaches and evolutionary development by allowing relatively non-complex systems to grow into larger, more elaborate systems, while providing code generation across the effort.

AdaGRAPH currently provides some limited support of packages. Full support for packages and generics will be provided in later releases.

AdaGRAPH uses macros for generation of user definable and extensible program unit idiom templates.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

AdaGRAPH supports the PAMELA (trademark of G.W. Cherry, Thought**Tools) method, but is not constrained by it. The basic paradigm is system development by process abstraction, similar to structured analysis techniques, but brought up to date with Ada and the software engineering principles behind that language's development.

13. Describe the external tools with which your system interfaces (tool compatibility).

AdaGRAPH interacts with a large number of commercial word processors/editors. AdaGRAPH is currently coupled with a programmable editor called Brief (trademark of Underware, Inc.), and the Alsys Ada Compiler for the IBM-PC series.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

AdaGRAPH strongly encourages hierarchical decomposition by a layered design representation. Flow is generally top-down, although we support limited bottom-up design: use of library units, and type inheritance.

AdaGRAPH provides a collection of system views: a structure-chart-like module map, as well as the data flow graphs used to design the system.

Object-oriented design may be adopted for use with AdaGRAPH, but more than likely that would be after the overall high-level design has been completed, and detailed design is starting.

15. Is your system supported by formal syntax & semantics? Describe briefly.

The graphical constructs used within AdaGRAPH form a formal, graphical language. The semantics are a subset of Ada, and have not been formalized (Ada hasn't been either).

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

AdaGRAPH Version 1.1 is currently priced at \$9,875 (first unit), \$9,100 (second-fifth), \$8,250 (sixth-tenth), \$7,100 (eleventh-twentieth). Site licenses may be negotiated. Annual maintenance is 12% of purchase price. Training has not generally been necessary, although a 3-4 day design workshop is advised. This can be arranged with various contractors, or with TASC, and the cost will be a function of the class size and expenses.

TASC sponsors a 4-day Ada Productivity Workshop which introduces the user to the Ada issues involved in system design, as well as the use of AdaGRAPH as a tool. The cost for that is approximately \$950 per person.

17. Indicate the hostability (measure of degree of portability) of your system.

AdaGRAPH is currently hosted on MS-DOS (PC-DOS), and as such will run on a wide variety of machines which support that operating system. AdaGRAPH is written in C.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

Systems engineering with AdaGRAPH is a sensible approach to system prototyping. The designer may determine that a system component should be developed in hardware, and can begin that development with an Ada specification of the component's capabilities.

AdaGRAPH is based upon the modern software engineering principles which motivate Ada: information hiding, strong typing, library units, etc.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

AdaGRAPH is a design and development system for process-oriented system development. It allows some flexibility in method. It embodies a strict

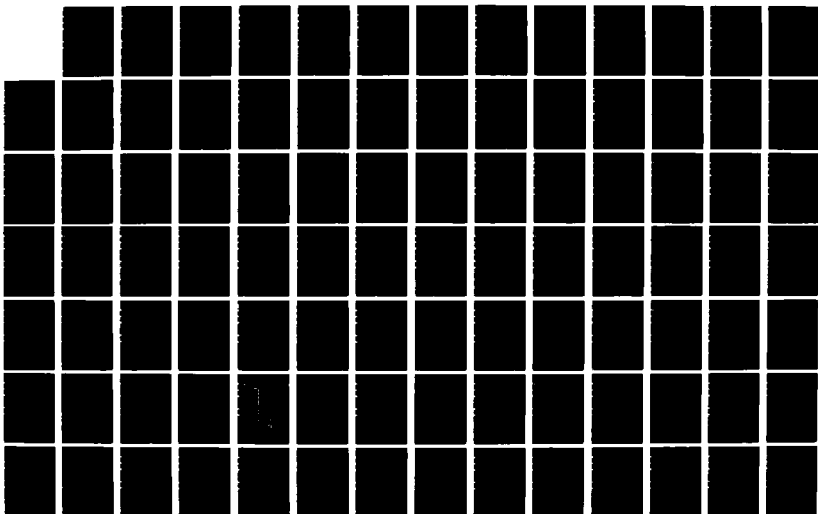
AD-A194 355

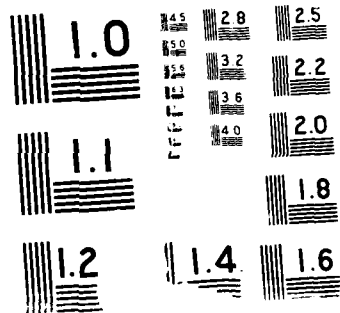
PROCEEDINGS OF THE STRATEGIC DEFENSE INITIATIVE
ORGANIZATION (SDIO) TOOL (U) INSTITUTE FOR DEFENSE
ANALYSES ALEXANDRIA VA D HEYSTEK 04 MAR 87 IDA-M-108
IDA/HQ-87-32004 NDA903-84-C-0031 F/G 12/5

2/5

UNCLASSIFIED

NL





notion of project control, but currently has limited management method constraints.

20. Describe how your system supports a team development approach. (Number of stations/users).

It is possible to use AdaGRAPH on a network of workstations (with a modern disk server and supporting transparent file access) and thereby support team development. On a single workstation, only a single user at a time may be at work, and on a non-networked workstation many users may be supported serially. Top level project designs may be copied to multiple workstations permitting team members to work on different system components.

21. Describe how your system supports design trade-offs.

Trade-offs of software versus hardware components are supported as described in 18. above. Other design trade-offs would have to be managed by the user.

22. Indicate the range of problems to which the system can be applied.

AdaGRAPH is suitable for Ada systems design and development in general; from embedded systems to business applications.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

a. Mr. Dan Wilkinson
McDonnell Douglas Aerospace
Information Services Company
5301 Bolsa Avenue
Huntington Beach, CA 92647
(714) 896-5478

b. Mr. Rich Connally

Mail Stop 432
Westinghouse Defense Electronics Center
P. O. Box 746
Baltimore, MD 21203

- c. Mr. Don King
VITRO Corporation
14000 Georgia Avenue
Silver Spring, MD 20901
(301) 231-2159
- d. Mr. Terry Nicholson
Dept. E-115, Bldg. 598, Gate 599
McDonnell Douglas Astronautics Company
Highway 94 North and Harpoon Drive
St. Louis, MO 63301
(314) 925-6650
- e. Ms. Elizabeth Brinker
Code 522.1
NASA/Goddard Space Flight Center
Building 23/E-429
Greenbelt, MD 20771
(301) 286-3129

Syscon Corporation
3990 Sherman Street
San Diego, CA 92110

Syscon Corporation

1. Describe how your system supports early detection of inconsistencies, closure and errors.

The SKETCHER tool formally applies to high level software design activities. In this capacity, SKETCHER assists with the development of the Top-Level software architecture and subsequent design refinements. SKETCHER supports the early detection of Top-Level design anomalies by 1) the enforcement of the Ada language semantics and syntax during the Object Oriented Design Diagram generation process, and 2) by the generation of compilable PDL files that accurately reflect the respective design diagrams.

The PEP tool formally applies to the Top-Level and Detailed-Level design activities. The output of the PEP tool is an enhanced PDL prototype files that is 1) compilable for the verification of the corresponding PDL representation, and 2) executable to support correctness validation of the design at these level.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

The SKETCHER and PEP tools produce boolean type progress metrics, which assess the consistency and correctness of the current state of the design. Progress is measured by 1) the compilation of SKETCHER produced PDL (with no detected errors), and 2) the compilation and execution of PEP produced enhanced PDL prototype(s). In utilizing the results of a prototypes execution, progress may be further measured (or quantified) based the number of design requirements satisfied.

3. Describe how your system supports documentation, program management and control.

The SKETCHER tool directly supports design documentation activities in that the graphic Object Oriented Design Diagrams and the corresponding PDL representations are consistent as design iteration and refinement activities occur.

Both tools indirectly support program management and control in that they operate on Ada compilation unit boundaries. Given that DoD-STD-2167 CSCI, TLCSC, CSC, and LLCSC terminology and definitions map to Ada compilation units, then the design generation, verification, validation, validation and documentation output products of the SKETCHER and PEP tools support program management and control activities.

4. Describe how your system supports real time design.

The SKETCHER and PEP tools are generally applicable to any Ada software development project. They are not limited to, nor excluded from, use on the development of specific software applications. The requirements that drove the SKETCHER tools development (i.e., graphic conventions and PDL generation requirements) were formalized on a research and development project involving the use of the Ada language in the development of communication system software. This development took into account the employment of a layered architecture, the implementation of standardized protocols, prioritized concurrent processing, common dynamic buffer management and timing services.

Both SKETCHER and PEP provide the means to express the requirements and mechanizations (e.g., size and time constraints, parallel processing) associated with real time software. PEP in particular, is designed to address the risks associated with real time software development, by permitting rapid prototyping of critical algorithms, implementation strategies, and interfaces.

5. Describe how your system supports concurrency, parallelism.

The SKETCHER and PEP tools support the syntax and semantics of tasks, as specified by the Ada language, as follows:

The SKETCHER tool provides interactive commands for the declaration of tasks and the inclusion of task body structure skeleton statements within the generated PDL. It does not support task types or objects, although this capability is a planned tool enhancement scheduled for FY 1987.

The PEP tool supports the full range of parallelism supported by the Ada language, extended to provide for execution tracing of task activation, termination, and rendezvous events. It also provides for the time stamping of rendezvous event occurrences.

6. Is your system constrained to a particular implementation language (Ada)?

Yes. Both the SKETCHER and PEP tools are strictly limited to the semantics and syntax of the Ada language as specified by MIL-STD-1815A. In the case of the PEP tool, the PDL representation that can be submitted to the tool is extended as per the grammar presented with the response to question 7 below. Furthermore, both tools are implemented in the Ada language.

7. Does your system produce Ada PDL?

The grammar to be used by PEP consists of full Ada augmented by the use of embedded English comments which are contained within square brackets ("[English]"). A subset of this grammar, shown below in Backus-Naur form, illustrates the application of the square bracket notation. The symbol " $\Rightarrow$ " has been used to highlight the location where the square brackets notation has been inserted into the Ada language. (See Figure 1 on the following pages)

8. Describe how your system supports life cycle intraphase & interphase communications.

Figures A and B illustrate the generalized activity flows associated with the use of the SKETCHER and PEP tools. The tools were designed to accommodate the highly iterative activities associated with the software development process in general. The scope of the flows (i.e., interphase or intraphase) are user and/or methodology dependent and are not limited by the tools' characteristics or functionality. The use of graphics and PDL have historically proven to be efficient and effective means of communication between development phase activities, between development group members, for internal reviews, for formal reviews with customers, etc.

9. Is your system automated, executable, compilable?

Yes. Both SKETCHER and PEP are automated and executable tools, implemented in the Ada language. Both tools produce compilable PDL files as products, and in the case of PEP, the output product is an executable and enhanced version of the submitted PDL representation.

SQUARE_BRACKETS ::=
 "[any english language structure with no
 embedded quotes or brackets]"

SUBPROGRAMS

SUBPROGRAM_SPECIFICATION ::=
 procedure IDENTIFIER [FORMAL_PART] ;
 function DESIGNATOR [FORMAL_PART] return TYPE_MARK
 FORMAL_PART ::=
 (SQUARE_BRACKETS) ;
 (PARAMETER_SPECIFICATION
 [: PARAMETER_SPECIFICATION])

STATEMENTS

SEQUENCE_OF_STATEMENTS ::=
 STATEMENT { STATEMENT }
 STATEMENT ::=
 { LABEL } SQUARE_BRACKETS ;
 { LABEL } SIMPLE_STATEMENT ;
 { LABEL } COMPOUND_STATEMENT

EXPRESSIONS, ET. AL.

EXPRESSION ::=
 RELATION { and RELATION } ;
 RELATION { and then RELATION } ;
 RELATION { or RELATION } ;
 RELATION { or else RELATION } ;
 RELATION { xor RELATION }
 RELATION ::=
 SQUARE_BRACKETS ;
 SIMPLE_EXPRESSION
 [RELATIONAL_OPERATOR SIMPLE_EXPRESSION]
 SIMPLE_EXPRESSION [not] in RANGE ;
 SIMPLE_EXPRESSION [not] in TYPE_MARK

DECLARATION

DECLARATIVE_PART ::=
 { BASIC_DECLARATIVE_ITEM } { LATER_DECLARATIVE_ITEM }
 BASIC_DECLARATIVE_ITEM ::=
 SQUARE_BRACKETS ;
 BASIC_DECLARATION ;
 REPRESENTATION_CLAUSE ;
 USE_CLAUSE

GENERIC

GENERIC_DECLARATION ::=
 GENERIC_SPECIFICATION ;
 GENERIC_SPECIFICATION ::=
 GENERIC_FORMAL_PART SUBPROGRAM_SPECIFICATION
 GENERIC_FORMAL_PART PACKAGE_SPECIFICATION
 GENERIC_FORMAL_PART ::=
 generic { GENERIC_PARAMETER_DECLARATION }
 GENERIC_PARAMETER_DECLARATION ::=
 (SQUARE_BRACKETS)
 IDENTIFIER_LIST : { in { out } } TYPE_MARK
 [:= EXPRESSION] ;
 type IDENTIFIER is GENERIC_TYPE_DEFINITION ;
 PRIVATE_TYPE_DECLARATION
 with SUBPROGRAM_SPECIFICATION [is NAME] ;
 with SUBPROGRAM_SPECIFICATION [is <>] ;

Figure 1

The Ada PDL grammar summary given above supplements that of the Ada language itself, and is used as the criteria by which the correctness of the syntax of PDL being processed is judged. The following are examples of the correct use of square brackets in Ada PDL.

```
with TEXT_IO ;
generic
  [ generic parameters are yet to be defined ]
procedure ADA_PDL_EXAMPLE
  ( [input and output parameters to be defined] ) is
-- This procedure is example of the use of square
-- brackets to defer detail during PDL writing.

  [ type LOCAL_TYPE is not yet completely defined ]
  type NEW_INTEGER is range [lower to upper values] ;

begin
  -- start processing
  [ initialize the system ]
  TEXT_IO.PUT_LINE ( " hi there novice user " ) ;
  -- complete processing

  [ shut down the system ]
exception
  when others =>
    [ handle errors appropriately ]
end ADA_PDL_EXAMPLE ;
```

Figure 1 (cont'd)

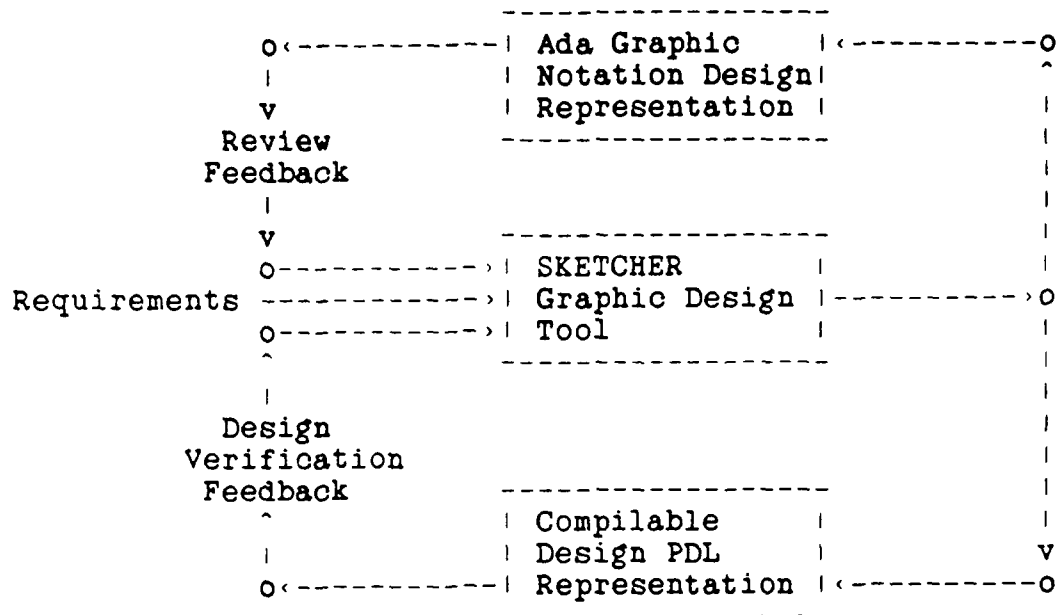


Figure A ADA GRAPHIC NOTATION FLOWS

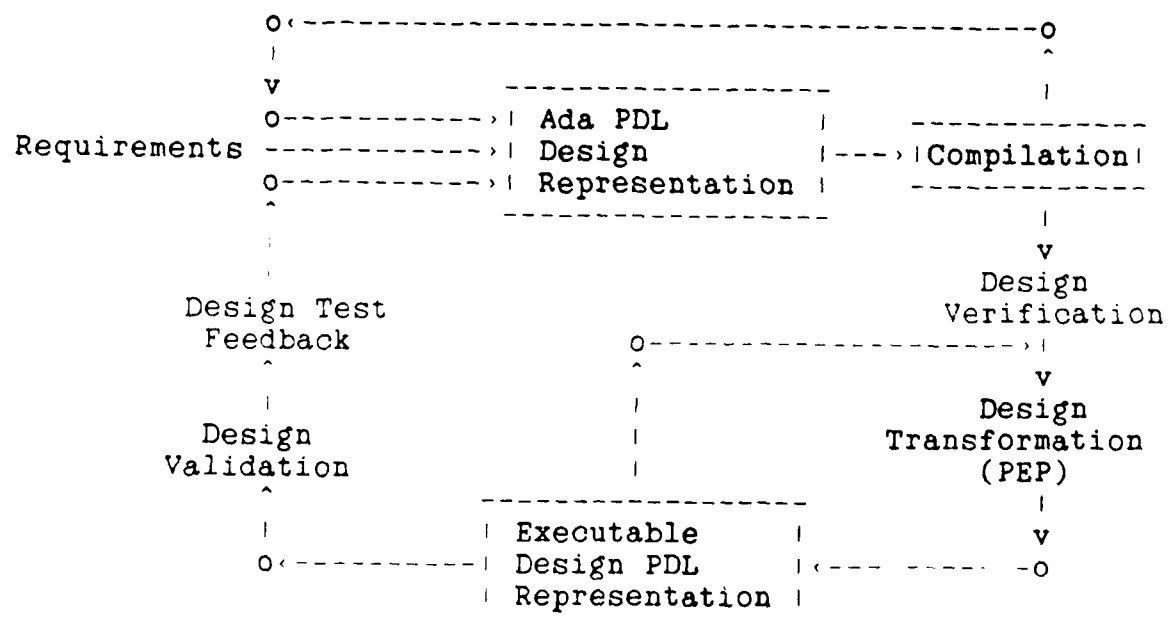


Figure B RAPID PROTOTYPING FLOWS

10. Describe the graphics support for your system.

The SKETCHER tool is an interactive graphics design tool that supports the development of software architectures and subsequent design refinements. SKETCHER currently operates with a Tektronix 4107 series bit-mapped color graphics terminal. The Ada Graphic Design Notation utilizes icons that are derived from the works of Booch's Object Oriented Design representations and Buhr's graphic conventions.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics
- Macros
- Data flows
- Control flows

Early prototyping and rapid prototyping are supported by the SKETCHER and PEP tools. SKETCHER provides an interactive tool for the development of prototype PDL skeleton shells which front ends the PEP tool. This forms an automated means for the development of experimental prototypes (possibly during the requirements definition phase) or more formal rapid prototypes that may be generated at the design or coding phase of a development.

Software reusability is supported by the SKETCHER tools ability to interactively generate software architecture components that address reusability issues (such as the localization of specific hardware and/or operating system interfaces to unique package areas).

Information hiding is supported by both tools by their ability to both hide and defer details. Local (e.g., to a package) information can not be accessed without being explicitly exported. Both tools support the deferral of levels of detail, which are unimportant at the current design level.

Packaging concept is supported by the SKETCHER and PEP tools by the adherence to the Ada language package feature as specified by the Ada language.

Abstraction is supported consistent with the Ada language.

Typing is supported consistent with the Ada language by PEP. This support is extended to include the ability to examine and set objects of arbitrarily complex data types. SKETCHER's support of typing is limited to the identification of key data types, which is often critical even at architectural levels of the design process.

Evolutionary development is supported directly by SKETCHER by the use of generalized "TBD" types and objects.

Generics are supported by the SKETCHER and PEP tools by the adherence to the Ada language generic feature as specified by the Ada language.

Modules are not supported by SKETCHER or PEP.

Data flows and associated analysis activities are supported by the prototype execution tracing options of the PEP tool.

Control flows are supported by SKETCHER through the ability to indicate subprogram and task entry calls on the diagram. PEP supports the processing of all Ada forms of flow control.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

There are no paradigms built directly into either PEP or SKETCHER. However, the User's Manuals of both provide usage guidelines which, when followed, result in significantly improved productivity and designs.

13. Describe the external tools with which your system interfaces (tool compatibility).

Both SKETCHER and PEP communicate with other system tools through the use of standard text files, which contain the Ada PDL being produced or being processed. This is implemented through the utilization of Ada's `TEXT_IO` capabilities. This mechanization permits SKETCHER and PEP to interface with a wide variety of existing tools including tools such as text editors, compilers, configuration managers.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

SKETCHER was designed and developed to specifically support object-oriented design for Ada software. The graphical design commands supported by SKETCHER include both topdown (e.g., creation of a subprogram within an existing object) and bottomup (e.g., creation of a package or subprogram encapsulating existing objects). The designers flexibility/creativity is further supported by the ability to move and resize objects anywhere on the diagram which would not result in illegal Ada syntax or semantics. SKETCHER's command repertoire includes all the commands necessary to support any decomposition technique.

Because PEP processes Ada PDL without any constraint based on past processing (i.e., compilations), it supports any design variation possible. Its flexibility and adaptability permit the rapid prototyping of alternate design concepts, thereby encouraging the designer's creativity in developing the final design solution. PEP's support for various types of hierarchical decomposition is essentially equivalent to that of any conforming Ada Program Library (i.e., there are no restrictions except on compilation order).

15. Is your system supported by formal syntax & semantics? Describe briefly.

Yes. PEP and SKETCHER support the generation and processing of a formally defined Ada PDL which is defined in Bacus-Naur form (see the response to question #7 for a summary of the grammar) and has been verified using various analysis tools. The Object-Oriented Design Diagram conventions (which were adapted from the works of Buhr and Booch) are of necessity formal, to permit the on-the-fly semantics checking which is performed by SKETCHER. The formal definition is given in the SKETCHER User's Manual.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

SKETCHER and PEP are both currently proprietary tools of SYSCON, and as such a cost for the software is currently not available. The following utilization requirements are nominal based on SYSCON's internal experience.

SKETCHER

Required Hardware: Graphics Terminal (e.g., TEK 4207)
Graphics Printer (e.g., TEK 4696)
Required Software: None
Required Training: Approximately 4 hour class
(proficiency takes approx 1 week of use)

PEP

Required Hardware: None
Required Software: Compatible Ada Compiler and Command Language
Interpreter
Required Training: Basic Usage - 2 hour class
Advanced Usage - 3 day class
Required Expendables: None
Notes: An ANSI style terminal will improve display
efficiency during prototype execution.

17. Indicate the hostability (measure of degree of portability) of your system.

Both SKETCHER and PEP are written in Ada to obtain the maximum amount of hostability possible. SKETCHER contains no system (host) dependencies and should be nearly 100% rehostable as is (this has been our experience to date). PEP utilizes the host Ada compiler and is partially written in the host's command language. The inter-program communication mechanism and command language facilities used by PEP are all very simple, to minimize the effort required to rehost the tool. The balance of the PEP software is written in Ada, and should be 100% rehostable.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

SYSCON's tools use the inherent capabilities of the Ada language to provide for interdisciplinary representations. At the top-level of system design or engineering, SYSCON typically uses "virtual packages" to encapsulate functional requirements, without necessitating a commitment to a particular implementation method (e.g., hardware or software). The utilization of Ada's interface definition mechanisms provides a method for capturing the details of the current level of abstraction while deferring decisions about lower levels.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

A high level overview of the software methodology utilized by SYSCON is presented in Figure C (on the following page). This methodology is strongly oriented toward the cited development methodology category. The SKETCHER and PEP tools, together with a validated Ada compiler, are the primary development tools components of an APSE currently utilized by SYSCON for the development of Ada software.

Other tools that are also part of the APSE are presented in the table on Figure C. These facilities are included for completeness in the presentation of the methodology. Specific details of the APSE tool facilities, with the exception of the SKETCHER and PEP tools, are beyond the scope of questionnaire.

**20. Describe how your system supports a team development approach.
(Number of stations/users).**

Both SKETCHER and PEP are multi-user systems which can be used by as many designers as the host hardware is able to computationally support. Integration of several teams/individual efforts is accomplished at the program library level (e.g., through compilation of the Ada PDL). The absence of a built-in configuration management scheme permits SKETCHER and PEP to be ported to a wide variety of APSE's with minimal impact on existing CM systems.

21. Describe how your system supports design trade-offs.

One of the prime motivations of both SKETCHER gives a designer the capability to develop and evaluate many different design architectures prior to committing to the one which best addresses all requirements. PEP's ability to generate rapid prototypes give the designer an opportunity to perform detailed evaluation of algorithms, data structures, and approaches prior to full-scale development.

DOD-STD 2167 Level	Development Phase	Parallel/Support Activity	Applicable Tools
Pre-Software Development	Concept Formulation	Ada Development Environment Indoctrination	(1,2,3,4) (5,6) (7,8,9)
		v	
Software Requirements Analysis	Requirements Formulation	Standards/Metrics	(7)
		v	
	Software Architecture [Graphs, Compilable PDL, Executable Prototypes]	Rapid Prototyping	(5,6,11)
		v	
Preliminary Design	Top Level Design [Graphs, Compilable PDL, Executable Prototypes]	Formal Design	(5,6) (8,9, 10,11)
		v	
Detailed Design	Detailed Level Design [Graphs, Compilable PDL, Executable Prototypes]	Informal Test & Integrate	(5,6) (8,9, 10,11)
		v	
Coding & Unit Test	Code & Debug [Coding & Transformation Guidelines]	Develop Code	(6) (8,9, 10,11)
		v v v	
CSC Integration Test	Test & Integration	Formal Testing	(6) (8,9)
		v	
CSCI Integration Test	Evaluation	Software Metrics	(8,10, 11)
	Maintenance	v	v

Figure C ADA BASED DESIGN METHODOLOGY OVERVIEW

MAPSE	DEVELOPMENT TOOLS	SUPPORT TOOLS
(1) Ada Compiler	(5) SKETCHER	(7) Drafter
(2) Source Line Editor	(6) PEP	(8) Ada Statement Analyzer
(3) Library Management		(9) Pretty Printer
(4) Debugger		(10) Abstractor
		(11) Ada Architecture Analyzer

22. Indicate the range of problems to which the system can be applied.

PEP and SKETCHER are generic software development support tools applicable to development of systems of all kinds. In certain circumstances, for example the development of a trusted system using a specification language, standard procedures may have to be altered somewhat. Any system which can be developed using Ada PDL as part of the design process, can successfully utilize SKETCHER and PEP.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

Both PEP and SKETCHER are proprietary tools of SYSCON Corporation. In that role, they are used to support SYSCON's ongoing Ada activities, and are thoroughly integrated into the SYSCON Ada Software Development Methodology.

SofTech
460 Totten Pond Road
Waltham, MA 02254

SofTech

NOTE: The system that is described in the following answers is the AGDS. Some parts of the system have been prototyped using IRAD funds as a "proof-of-concept", but there is not yet funding for a complete system, and no plans to produce a commercial product.

NOTE: SADT is the name of the original method devised by SofTech, and IDEFO is the public domain name. The terms are used interchangeably in this discussion.

1. Describe how your system supports early detection of inconsistencies, closure and errors.

There are two forms of consistency checking. First, the IDEFO drawing package prevents the user from drawing illegal IDEFO diagrams (such as arrows or boxes in the wrong place). Second, we have developed the set of rules for checking global consistency (duplicate arrow or box names, diagrams with external inputs that don't match the parent, etc.).

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

No response was provided for this question.

3. Describe how your system supports documentation, program management and control.

IDEFO is designed to include a documentation system. As such, the completed AGDS would include the IDEFO documentation. Further, it is planned that the system would allow Ada code, including comments, to be hidden behind the IDEFO information.

4. Describe how your system supports real time design.

No response was provided for this question.

5. Describe how your system supports concurrency, parallelism.

IDEFO diagrams normally contain no information about the timing of events, so that functions can happen consecutive or concurrently. We have extended the IDEFO diagrams to include timing information, so that the user can specify concurrency.

6. Is your system constrained to a particular implementation language (Ada)?

Much of the system (IDEFO construction, consistency checking, simulation, etc.) is language independent. However, the method used for the final translation to PDL is currently based on Ada.

7. Does your system produce Ada PDL?

Yes. It produces a compilable super-structure of an Ada program.

8. Describe how your system supports life cycle intraphase & interphase communications.

The entire design process is done by using behavioral IDEFO. These diagrams can be verified, simulated, reviewed by others, and checked for reusability.

Once the design enters the programming phase, high level Ada code is automatically generated, which can be filled in to produce running programs. These programs can again be simulated.

As the system enters the maintenance phase, running programs can be returned to the IDEFO semantic database for purposes of documentation and reusability.

9. Is your system automated, executable, compilable?

The system is planned as an automated system, producing compilable Ada code.

10. Describe the graphics support for your system.

Design and review are handled by a graphics front-end. Running on an IBM PC. Simulation also utilizes graphics.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing

- Evolutionary development
- Generics
- Evolutionary development
- Generics
- Macros
- Data flows
- Control flows

Software reusability is planned to be handled by access routines to a IDEFO database. The user has the ability to browse, request specific routines, find reusability information, and look for common functions.

Information hiding is done by using the hierarchical structure of IDEFO to hide lower levels of the design. Further, the system will be able to hide Ada code within the SADT diagrams.

Packaging is used in the generation of the Ada code. Each box within an IDEFO diagram becomes an Ada package.

Typing is user controllable.

Generics are implicit in the conversion method.

Data and control flows are both supported by IDEFO.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

The most efficient way to build systems is to design in behavioral IDEFO, ignoring programming considerations. This design requires graphics, workstations, simulation, verification, review, and reusability.

The second part of system design is iterating between code and design.

13. Describe the external tools with which your system interfaces (tool compatibility).

There are converters for moving from the IDEFO semantic database to other tools.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

IDEFO is designed as a hierarchical method. As such, there are people using our tool from the top-down, or bottom-up.

15. Is your system supported by formal syntax & semantics? Describe briefly.

See papers on SADT and IDEFO.

16. Outline typical utilization costs for your system (cost of acquiring, using, training & maintaining it).

No response was provided for this question.

17. Indicate the hostability (measure of degree of portability) of your system.

Currently, it requires an MS/DOS machine. The design for the final system requires any workstation and main-frame combination.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

IDEFO has been used in software, hardware, factory automation, personnel procedures, and hundreds of others.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

It includes design and development.

20. Describe how your system supports a team development approach.
(Number of stations/users).

Users work at workstations. Diagrams are placed in a database on a mainframe, to be reviewed by others at their workstations.

It is planned that the completed diagrams and code of different organizations can be checked for consistency, and simulated or combined on the mainframe.

21. Describe how your system supports design trade-offs.

No response was provided for this question.

22. Indicate the range of problems to which the system can be applied.

No response was provided for this question.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

No response was provided for this question.

Associative Design Technology, Inc.
142 Brigham Hill Road
P.O. Box 518
North Grafton, MA 01536-0518

Associative Design Technology, Inc.

1. Describe how your system supports early detection of inconsistencies, closure and errors.

The central module of the PadTech Design Core is an inference engine called the "process machine." The process machine applies system- and user-defined rules of logic (inferences and constraints) to construct a knowledge base that contains the process design. The process machine is thereby able to detect inconsistencies immediately. It will not allow the user to enter logically inconsistent facts in the knowledge base.

By definition, the knowledge base, built and maintained by the process machine is assumed to be a closed world. This means that the knowledge base is the universe. A fact can only be true if the supporting data is included in the knowledge base. The process machine detects immediately any logical assertion or query that would violate this closure principle and informs the user or expands the knowledge base as appropriate.

The process machine has built-in error handling that can inform the user of any errors resulting from attempted logical operations. The error is reported through either the textual or graphics interface to the process machine.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

The PadTech Design Core has not automated the production of quantifiable metrics.

3. Describe how your system supports documentation, program management and control.

The PadTech Design Core does not provide pre-packaged automated project management tools for documentation, program management or access control. These features, however, can be treated as universal attributes of objects in the system design. Hence, like any other object attribute, they can be provided for and even made mandatory when using PadTech to design a system. Therefore, these features are completely customizable for each project.

4. Describe how your system supports real time design.

The PadTech Design Core is intended to meet the needs of real time design. Time in the PadTech Design Core is treated as a sequence of events. The operation of PadTech processors can be synchronized with a real time sequence of events to any granularity required by the system designer.

5. Describe how your system supports concurrency, parallelism.

The PadTech Design Core supports the creation of designs with massive parallelism, and allows the designer to incorporate as much concurrency as allowed by the problem. It supports the notion of "processors." Processors are, in essence, executable objects triggered by changes in the state of the knowledge base. The result of executing a processor is called an event. An event actually marks a state transition and, in general, signifies the creation/termination of an object or a set of objects and the relations connecting them. A "process diagram" is used to capture the "trigger rules" and "processor-produces-event" relationships that together describe a complete process. The trigger rules, which determine under what conditions a processor will "fire," can be quite complex and introduce concurrency. Anytime the trigger rule conditions are met (i.e.: anytime that the state of the universe of discourse is as described in the trigger rules) then a processor will fire. If the trigger rule calls for a processor to be fired for every instance of a particular object (or set of objects and their relationships) then the processors are made to execute concurrently. At that point, each processor is autonomous. Communication takes place strictly through the knowledge base. There is theoretically no limit on the number of concurrently fired processors.

6. Is your system constrained to a particular implementation language (Ada)?

The PadTech Design Core is not constrained to create designs for a particular language. In fact, a PadTech design may be used to integrate modules implemented in diverse languages.

7. Does your system produce Ada PDL?

The PadTech Design Core will have full Ada PDL production capability in Revision 2.00. A prototype Ada PDL production capability will be demonstrated with Revision 1.00. Designs produced by PadTech are particularly well-suited to support Ada software engineering principles.

8. Describe how your system supports life cycle Intraphase & Interphase communications.

One of the goals of the PadTech Design Core is to elevate and automate the role of a process architect and to totally integrate the design work of the architect with the implementation of the engineer. The result is that the traditional boundaries between design and implementation phases of the software life-cycle are not distinct. Indeed, the dichotomy between design and implementation is largely removed. PadTech is built upon the notion of "archtyping." The early results of the design effort are, in fact, more than a prototype; it is actually the basis for building the application.

At the same time, the designer is able to quickly modify the model whenever new information requires a change to his/her plans. The result is an unusual rapid prototyping approach and not a strict waterfall software development life cycle model.

9. Is your system automated, executable, compilable?

Various aspects of the PadTech Design Core are fully automated. For example, the graphics design tool automatically generates PADL (Process Architecture Design Language) which is then sent to the process machine to query/modify the knowledge base. This same approach is used to implement 3GL processors that communicate with and are integrated by the process machine. ADT provides libraries for generating PADL in order to automate this approach. "Processors" in the ADT vernacular are executable objects. ADT does not offer PADL compilation capabilities at this time.

In terms of translation from the ADT design language (PADL) to Ada, the production of Ada (or Ada PDL) is automated. ADT does not provide an Ada language processor.

10. Describe the graphics support for your system.

The PadTech Design Core offers two fully functional user interfaces, one textual- and one graphics-based. The graphics-based module is called PadSurface. PadSurface is currently implemented on a Silicon Graphics color, high-resolution workstation. The user interacts with the system through a mouse and the keyboard. The graphics interface allows the user to create both types and instances. It supports three system-supplied diagram types (activity diagrams for planning, concept diagrams for data design, and processor diagrams for flow control) as well as user-defined diagram types.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics
- Macros
- Data flows
- Control flows

Because many of the concepts are related, and some do not have generally accepted definitions, we have discussed the concepts as a group in the context of the PadTech methodology. The PadTech Design Core takes a unique approach to the software design process. This approach makes use of many concepts commonly found in object-oriented and logic programming environments. In particular, the notions of types, instances of, software reusability, and abstraction are handled in a very sophisticated and complete manner. A few definitions of the PadTech concepts are included in this questionnaire.

The PadTech Design core consists of a knowledge base, the process machine and various interface systems including a PADL editor-interpreter and PadSurface (a graphics interface). The knowledge base is divided into the information base and the rule base. The rule base contains constraint rules, inference rules and consistency rules. The process machine is responsible for creating and maintaining the knowledge base. The process machine receives messages from processors (reusable executable software modules). The messages may be queries, realm modifications or triggering requests.

A realm provides granularity in the knowledge base. It is a portion of the world from a particular point of view. A realm consists of a state of affairs; i.e., objects that bear relations to one another. Some of these objects are executable and are called processors or operators.

Everything occupying a realm is an object. A type is an object that presides over its instances. A type is a prototype or template that specifies a pattern to which its instances must conform (inheritance rules are applicable). One object is a subtype of another if and only if they are distinct

and necessarily every instance of one is an instance of the other. Objects may have attributes. An attribute is a binary relation and the value of an attribute is an object of a type that satisfies the intension of the binary relation. The relation itself has various attributes including cardinality. Hence relations can be mandatory or optional, and an upper bound for the number of instantiations of the relation can be enforced. An attribute is either a property or a component. An attribute is a component if a change in its values for a given object actually changes the identity of that object.

A partition of a given type is a collection of subtypes, constrained to share no instances.

A processor is an executable object. An event type is a type that signals the transition of the knowledge base from one state to another. Events are produced by processors and, in turn, can trigger other processors.

Activities are high-level looks at parts of a process. An activity produces products. In turn, products are consumed by activities.

A diagram is a collection of instances of types given in the diagram's view, further restricted by the diagram's space, and limited to a particular realm. An activity diagram contains only activities, products, and the relations produces and consumes. A Processor diagram contains only processors and events, and the relations produces and triggers. Processors can be divided into action processors that must always produce an event and decision processors that produce an event only under certain predefined conditions. Concept diagrams contain various user-defined types, not included in activity or processor diagrams. Concept diagrams are linked to activity diagrams through the products.

Processor diagrams are linked to activity diagrams through the activities. Activity diagrams are linked to user-defined subjects. Concept diagrams capture dataflow information. Process diagrams capture control flow information.

The PADL interpreter supports the notion of PADL macros that accept parameters and act as functions.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

The paradigm underlying the PadTech Design Core is largely compatible with the work of the ISO committee for Conceptual Schema (Ref: "Concepts and Terminology for the Conceptual Schema and the Information Base", International Organization for Standardization (ISO), ISO/TC97/SC5/WG3, March 1982, copies available from Secretariat: ANSI (USA), publication number ISO/TC97/SC5 - N 695).

This paradigm combines elements of an object-oriented approach with logic programming environments. The result is a proprietary paradigm which supports not only "conceptual analysis" but also provides an integration platform to control the execution of "conceptual processors."

The actual basic model underlying the product is customizable. Hence the user can elect to hide the PadTech paradigm by placing some other methodology as a shell over the packaged basic model. Hence the product can be made to support a variety of standard methodologies.

13. Describe the external tools with which your system interfaces (tool compatibility).

The PadTech Design Core includes a runtime library designed to allow external tools to communicate with the process machine. Any tool capable of manipulating string parameters (character data type) can make use of the PADL Statement Builder library.

This set of routines will generate PADL statements and send them to the process machine. It also facilitates PadTech knowledge base queries.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

The PadTech Design Core supports the analysis of systems in a manner similar to that used in most object-oriented and logic programming environments. In general, the emphasis is on conceptual analysis: i.e., a logical decomposition of the problem into a set of objects and relations between objects. PadTech takes the design methodology much further than most object-oriented or logic programming tools and even includes a method for very high-level planning.

15. Is your system supported by formal syntax & semantics? Describe briefly.

The PadTech Design Core defines a process architecture design language known as PADL. The syntax for this language is formally defined by the BNF-like input for the Unix tools LEX and YACC. (YACC accepts specifications which define a LALR(1) grammar with disambiguating rules. The semantics for PADL have been defined in predicate calculus. The actions of the process machine (inference engine) can also be defined in predicate calculus.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

The beta revision of the product requires a Silicon Graphics 3010 terminal and a Data General DS/7500 workstation. The total cost of the Silicon Graphics hw/sw is approximately \$46,110. The total cost of the Data General hw/sw is approximately \$31,520. The initial cost of ADT's software product, the PadTech Design Core, is \$35,000. Monthly support cost for the Design Core is \$350.

17. Indicate the hostability (measure of degree of portability) of your system.

ADT intends to port the PadTech Design Core product to a number of systems in 1987. The product is written in Common Lisp and C to facilitate porting activities. At this time, the product is available only in the Silicon Graphics - Data General configuration described above.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

The PadTech Design Core is intended to be interdisciplinary. It promotes the notion of a process architect. PadTech processes can be implemented in computer software and hardware, physical systems, or people. The symbols and terminology chosen to represent the process design are customizable by the process architect. Therefore, the tool can be used by a variety of experts (engineering or otherwise).

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

The PadTech Design Core embodies a planning, design, development and, to some extent, a programming methodology (although the emphasis is at a system level and is not necessarily applicable on the traditional algorithmic level of 3GL structured programming).

The PadTech Design Core does not enforce or assume a particular project control or management methodology.

20. Describe how your system supports a team development approach. (Number of stations/users).

The PadTech Design Core Revision 1.00 has been implemented on a single user workstation (one station per user). Full multi-user (team) access to a single design knowledge base will be available in Revision 2.00 expected to go to Beta test in June of 1987.

21. Describe how your system supports design trade-offs.

The PadTech Design Core provides a system architect with a very powerful set of tools to facilitate the production of dynamic, flexible process designs. The architect is, therefore, able to modify designs or try new approaches very easily.

22. Indicate the range of problems to which the system can be applied.

The PadTech Design Core is applicable to a wide range of problems. It is best used for very large, complex systems with a high degree of concurrency required. Any process, computer-based or otherwise, can be analyzed or designed using the PadTech Design Core.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

As of January 1, 1987, only an Alpha version of the Design Core has been released. The Beta version of the product is expected to be released by February 1, 1987, and Revision 1.00 is expected to ship by April 1. ADT has two Alpha customers listed below:

1. Data General Corporation
ATTN: Peter Doonan
400 Computer Drive
Westborough, MA 01580
Phone (617) 870-6348
2. Coopers & Lybrand Associates Ltd
ATTN: Hans Deiter Scholz
Plumtree Court
London EC4A 4HT
Phone 01-583-5000

Note: It is the intention of Data General to both use the product internally to help meet the needs of their corporate MIS group and to market the product in both the commercial and federal marketplaces.

Coopers & Lybrand U.K. has successfully used the product in their role as consultants to the British Royal Navy. They intend to use the product in various other consulting projects both within Great Britain and the rest of Europe.

Both DG and C&L can provide further information concerning their use (or their clients' use) of the product.

AD CAD, Inc.
University Place, Suite 200
124 Mt. Auburn Street
Cambridge, MA 02138

1. Describe how your system supports early detection of inconsistencies, closure and errors.

By its very nature, the descriptions and specifications in STATEMENT1's visual language are precise, yet clear and comprehensive, and thus allow for relatively easy manual detection of design errors and inconsistencies from the earliest stages.

The descriptions are completely 'understood' by STATEMENT1, thus allowing for automatic testing and simulations of the system under description (henceforth, the SUD) right from the start. Consequently, even if the description at a given point in time is very high-level and lacking most low-level details, it can be simulated and tested for consistency, completeness and many additional nontrivial properties at the level of detail that happens to be available.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

STATEMENT1 does not provide any explicit progress metric. We do not really believe that the distance from an ultimate final specification can be measured by naive numerical means. One can label entities in STATEMENT1 as 'not fully specified yet' and count how many such remain, and similarly one can count the levels of detail already present in the description and try to estimate how much deeper the final description will go. Obviously, however, such metrics cannot be overly significant.

3. Describe how your system supports documentation, program management and control.

STATEMENT1 provides extensive documentation facilities, including several kinds of graphical outputs, fixed format documents (such as data dictionaries, interface documents, N-square charts, etc.) and, in the near future, also user-specified documents that will be able to adhere to the many standards in the field. The main point to note is that STATEMENT1 contains and understands a vast amount of intricate information about the structure, functionality, data-flow, and, notably, dynamic behavior of the SUD. It can thus retrieve subsets of that information either explicitly using a powerful query language, or implicitly using document generation.

STATEMENT1 will support conventional project management facilities, including version control and limited user access when working in a multiple-user environment.

4. Describe how your system supports real time design.

STATEMENT1 was designed with complex, computer-embedded real-time systems in mind. It is difficult to describe the entire approach in a few words, but it might suffice to say that STATEMENT1 utilizes a novel formalism, statecharts, for the behavioral description of such systems. Statecharts are described in:

D. Harel, "Statecharts: A visual Formalism for Complex Systems," *Science of Computer Programming* 8 (1987), to appear.

Time (and real-time in particular) is captured in statecharts by special timeout events that are used to limit the time of being in a state, waiting for an event, executing an activity, etc. Since essentially all entities in STATEMENT1 are multi-level and hierarchical, including (among other things) states, events and activities, the expressive power of these timing specifications is very rich. It goes without saying that the simulation and testing capabilities take the timing constraints fully into account.

5. Describe how your system supports concurrency, parallelism.

Again, dealing with the concurrency and parallelism inherent in complex systems is among the main concerns behind the STATEMENT1 system. In fact, one of the main features of statecharts is the extension of finite state machines and their diagrams to cater for concurrent state-components, on all levels, via the orthogonality construct. This removes a major obstacle from the use of the very natural state-based formalisms for really complex systems - the state-space explosion phenomenon.

6. Is your system constrained to a particular implementation language (Ada)?

No. STATEMENT1 can be compiled, in principle, into any high-level programming language, and for that matter also into microcode or silicon. Its database holds all information in a simple internal format suitable for manipulation in any such language.

7. Does your system produce Ada PDL?

Not yet, although the production of Ada PDL should be possible in a few months. The difficulty is in the rich expressive power of the languages of STATEMENT1 relative to Ada PDL. A subset of the STATEMENT1 languages that matches the capabilities of Ada PDL more closely could be treated for Ada PDL production quite painlessly in a very short time.

8. Describe how your system supports life cycle intraphase & interphase communications.

It does so by being based on languages that are, on one hand, clear and easy enough to serve as the lingua franca of the preliminary specification phases of the life cycle, even those parts carried out by the client or potential user, and, on the other hand, precise and rigorous enough to be fully simulated and tested as a prototype of the final implemented system. Descriptions of the SUD on varying levels can thus be communicated between groups and across phase boundaries, since they are all carried out in the same three graphical languages, and are supported by the very same system.

9. Is your system automated, executable, compilable?

Yes, yes and no. STATEMENT1 is fully automated as explained above. It is executable in the sense that its descriptions can be 'run' in precise simulations of the desired SUD's behavior, including indications of the functions and activities being carried out, the data and control signals flowing, the components and subsystems active, and the changes that take place in modes, states and conditions. It is, as of now, not directly compilable into code, though work on compilation into Ada is underway.

10. Describe the graphics support for your system.

Here, too, the topic begs for an extensive response, which is beyond the scope of these notes. Actually, to say that STATEMENT1 'has graphic support' is a gross understatement - It is really a graphical system par excellence. The underlying philosophy of STATEMENT1 is not that formal descriptions should be supplemented by graphics, but that they should be entirely graphical. STATEMENT1 allows for the specification of the SUD from three complementary and interrelated points of view, structural, functional and behavioral, and each is described using a novel diagrammatic language (respectively, module-charts, activity-charts and statecharts). The languages have several common features, including hierarchy depicted by

encapsulation, modularity, multi-level connections, etc. The graphic editors support the usual kind of insert, delete, zoom, pan, scroll, move and copy instructions, in a menu-driven way, with the system translating all graphical information into internal representations in its database.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics
- Macros
- Data flows
- Control flows

In a way, STATEMENT1 can be claimed to support most of these concepts, though not always explicitly. STATEMENT1 definitely provides both early and rapid prototyping, by the formal, precise and fully detailed simulations, as explained earlier. Reusability and macros are supported insofar as the descriptions are highly modular, and thus can be 'plugged into' many different places, and reused in other contexts. This applies to states ('behavioral macros') activities ('functional macros') and even components and modules ('physical macros'). Information hiding, packaging (indirectly) and abstraction are supported by virtue of the 'deep' hierarchical nature of all STATEMENT1 descriptions. For example, activities impose strict scoping rules, not unlike those found, (e.g., in Pascal or Ada), so that internal events, conditions, signals and data are hidden from higher levels and exterior entities. Data and control flow are basic to, respectively, activity charts and statecharts, and show up very clearly in the graphics, as well as in the simulations and the reports and documents.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

No, none of the usual (overkilled) ones. We believe in clear, precise and simulatable descriptions of all aspects of the SUD, with a firm commitment to what one might call 'visual formalisms' i.e., formal diagrammatic languages.

13. Describe the external tools with which your system interfaces (tool compatibility).

STATEMENT1 does not interface with other tools.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

As discussed above, hierarchical descriptions form one of the cornerstones of STATEMENT1. They can be utilized in top-down, bottom-up, inside-out, or mixed fashions. The user of STATEMENT1 is supposed to be a highly trained professional, and since STATEMENT1 does not impose any rigid recipes or templates for making progress, the designer's creativity is as important as anything else. Object oriented design, though not a central concern in the conception of STATEMENT1, is possible using separate activities for separate objects, and specifying their interrelationships via the controlling statecharts.

15. Is your system supported by formal syntax & semantics? Describe briefly.

Definitely. Each of the three graphical languages has a precise graphical (and hence also textual) syntax, and a formal mathematical operational semantics. Activity charts, for example, specify 'possible' data-flow, an unusual concept which allows their statechart to control the actual dynamic data-flow. Statecharts have quite a novel semantics, that is described in detail in:

D. Harel, A. Pnueli, J.P. Schmidt and R. Sherman, "On the Formal Semantics of Statecharts," submitted for publication.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

The basic software package costs around \$15K per graphical workstation, with various add-ons (e.g., testing and simulation modules, project management routines, etc.) costing between \$2K and \$10K each. The price includes training and maintenance.

17. Indicate the hostability (measure of degree of portability) of your system.

STATEMENT1 is written mainly in Pascal, and currently runs on a Vax or Micro-Vax with a VMS operating system and a color Tektronix graphical terminal. Versions running on Vax-Station, Sun, Apollo and IBM workstations are forthcoming. A standard plotter and an additional alphanumeric terminal are recommended.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

STATEMENT1 utilizes clear graphical languages for precise specification and analysis. These languages transcend the differences between the various relevant engineering disciplines, a fact that has been thoroughly proved in three years of work with the languages by a large and heterogeneous team of engineers in the Lavi avionics project at the Israel Aircraft Industries. Thus, STATEMENT1 has been demonstrated to bridge the gaps and span the differences in approach, mentality and communication media of software, hardware and systems engineers.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

STATEMENT1 provides a novel and complete specification and design methodology. It does not touch upon the programming problem per se, except insofar as STATEMENT1 descriptions can lead (via direct downward compilation) to final working code.

20. Describe how your system supports a team development approach. (Number of stations/users).

STATEMENT1 is suited ideally for a number of workstations (say, between 2 and 25) linked via a network, with or without a mainframe in the background. Its project management facilities will be tailorable towards such team efforts.

21. Describe how your system supports design trade-offs.

Since the descriptions resulting from working with STATEMENT1 are on a level higher than conventional software, hardware, microcode, etc., STATEMENT1 transcends this division too. Its descriptions can, in principle, be compiled into software in a high-level language, into microcode or other low-level formalisms, and even directly into silicon. For ideas on the latter see:

D. Drusinsky and D. Harel, "Statecharts as an Abstract Model for Digital Control Units," CS86-12, The Weizmann Inst. of Science, Rehovot, Israel. Submitted for publication.

22. Indicate the range of problems to which the system can be applied.

STATEMENT1 can be used for the specification, analysis and design of any complex reactive system. Reactive systems appear in a wide spectrum of application areas. Their complexity stems from many diverse discrete events and conditions that control their behavior. Examples include avionics and weapons systems, VLSI, communication networks and protocols, process and control systems, and so on.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

Dr. Yonah Lavi
Manager R&D Computer Systems
Israel Aircraft Industries,
Lod, Israel
Tel: 072-3-9713716

Michael Zeevi
Div. Head for Software & Electronic Warfare
ELTA Israel Electronics Industries Ltd.
Electronics Division
Ashdod, Israel
Tel: 072-55-30738

Jacob Bailis
Methodologies
El-Op Electro-Optics Industries Ltd.
Science Based Park
Kiryat Weizmann, Rehovot, Israel
Tel: 072-8-486706.

Research Triangle Institute
P.O. Box 12194
Research Triangle Park, NC 27709

Research Triangle Institute

1. Describe how your system supports early detection of inconsistencies, closure and errors.

ADAS supports consistency checking at several levels. The first basic consistency checks take place during the system design and definition capture phase, using the ADAS Interactive graphics editor. On both single-level and multi-level systems, ADAS checks for flow and type consistency between functions and between levels. More detailed consistency checking is done using the ADAS consistency checking tool, which also checks functional consistency against the reusable templates from which the functions are modeled.

Due to the versatility of ADAS in defining systems at any hierarchical level and to any desired level of detail, it is possible to do top-down system design and simulation at each level and at any phase of the design. In doing so, the designer can determine the degree of system completeness based on the system requirements and specifications.

ADAS catches system conceptual design errors in the early stages of development through the use of consistency checking and Petri Net simulation. Other errors are detected through hierarchical analysis and simulation as the system is defined in increasing detail. Depending upon the degree of detail to which the system is defined, logic, processing, and coding errors can be detected via the ADAS modeling and simulation process.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

At the present time, ADAS provides progress tracking and reporting through performance simulation. The designer has the flexibility to determine the degree of detail and completeness at which he desires to model and analyze. Plans are nearing completion for a system requirements/specifications, documentation, and progress tracking system that will be a part of the ADAS system.

3. Describe how your system supports documentation, program management and control.

RTI uses the present ADAS system to develop documentation and manage programs. However, at the present time, no specific documentation or program management tools are part of ADAS. As described above, a system is being implemented that will allow changes made to documentation or the system design to be reflected into the other.

4. Describe how your system supports real time design.

ADAS supports a design methodology which promotes the co-design of software and hardware. To represent real-time analysis and simulation of the defined system, ADAS allows software functions to be assigned to hardware modules, thus representing real-time hardware constraints and real-time software contention for shared hardware.

5. Describe how your system supports concurrency, parallelism.

The ADAS methodology promotes the co-design of software and hardware. In defining the two separate systems, it is possible to define concurrent and/or parallel functions. The ADAS tool set operates on whatever system has been defined and captured so as many concurrent and parallel processes and functions as needed can be defined, analyzed, and simulated.

6. Is your system constrained to a particular implementation language (Ada)?

ADAS currently supports as programming languages Ada and C, and supports as hardware description languages Helix and ISP. RTI is under contract to the Navy VHSIC Program to add support of the VHDL hardware description language.

The source language of ADAS is C with the Ada interface written in Ada.

7. Does your system produce Ada PDL?

ADAS allows for the designer to code functionality in Ada for individual software modules. An ADAS program then generates Ada code to control the operation and interconnection of the individual modules.

8. Describe how your system supports life cycle intraphase & interphase communications.

ADAS is designed to be used from the very beginning of the design process. At any point in the design phase, designs can be saved and modifications to the system can be studied. Once the system is in other phases of the life cycle, proposed modifications can be reflected in the ADAS model before actual changes are implemented in the system.

9. Is your system automated, executable, compilable?

Yes, all of the above. ADAS tools are automated to the point where a user can easily capture designs and analyze systems. ADAS is also executable, so that a user can do complex simulation and performance analysis with very little intervention. When a system design is defined to a level of detail where each function can be described by a block of code, these functions can be represented by the code which is compiled and becomes part of the system definition.

10. Describe the graphics support for your system.

The majority of the ADAS design interface is graphical. The graphics editor is used to capture and edit the system design. Systems, functions, and interconnects can be defined graphically. A graphics terminal and a mouse are required for even basic system design. ADAS simulation tools have an optional graphical output, thus providing a visual analysis of the data and control flow during simulation of the system.

ADAS runs on a wide range of graphics devices from workstations such as the Vaxstation II, Micro Vax II(GPX), and Sun 3/160 to low cost color graphics terminals on multi-user UNIX and VMS computers, such as the Vax 8600.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics

- Macros
- Data flows
- Control flows

ADAS supports prototyping by allowing a designer to model a system by entering performance information as opposed to a complete structured description. Feasibility studies can be done, modeling several alternative architectures and/or algorithms, before a candidate system is selected for functional modeling. Thus, with the ADAS system both early prototyping and rapid prototyping can be done.

ADAS allows software blocks describing functions to be reused where the same or similar function occurs for functional simulation.

Using ADAS hierarchical modeling, it is quite easy to hide information in lower levels of a design.

ADAS supports the Ada packages.

Levels of abstraction can be modeled easily using the ADAS hierarchical structure.

As part of the system design, functions and interconnects can be typed. Type matching of model functions and interconnects is part of the consistency checking that is done.

ADAS allows design from the very highest system level down to the most detailed level of the system. Changes at any level in the system hierarchy are reflected throughout the entire system design. Designs at any point can be captured and saved for later comparison.

All ADAS-defined systems are created using a basic set of reusable templates, thus providing a continuity based on the standardized templates. New templates can be created and added as the designer requires. The use of a specific set of templates provides standardization across the system design.

User-written macros are supported in many of the ADAS tools.

Data and/or control flows are represented by interconnects between functions or hardware modules. Characteristics such as queues can also be associated with the flows.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

ADAS is based on a structured, hierarchical system design methodology and supports this methodology through the use of an integrated set of software tools. A major feature of the methodology is the co-design of software and hardware and the integration of both into a resultant system.

13. Describe the external tools with which your system interfaces (tool compatibility).

Helix and ISPS: ADAS provides an interface to both of these hardware description languages, allowing hardware module descriptions to be written in one of these languages and used in functional modelling.

VHDL: Via a Navy VHSC Program contract, ADAS will soon be able to generate VHSC Hardware Description Language through hardware functional simulation using a hardware description language.

Silicon Compilers' GENESIL: Two-way information exchange of hardware description information allows generation of integrated circuit mask data and functional simulation of the resulting chip. Information area, power consumption, etc., are reported back to the system designer.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

The ADAS design capture and editing tools provide an easy method of defining a hierarchical system model. Once a top-level model has been defined, each function within that level can be defined in more detail by creating a new graphical model representing that specific function. This process can be done continuously at each level until the desired level of detail has been attained. When simulating or analyzing the system, the definition of the hierarchical models can be reflected all the way back up to the top-most model, thus attaining a more realistic system view by using the performance information passed up from the lower models.

There are no restrictions on the type or style of designs that can be defined by the designer in either software or hardware. Designs can easily be exchanged for others to allow comparison of different architectures or algorithms.

15. Is your system supported by formal syntax & semantics? Describe briefly.

ADAS uses a graphical representation rather than a language-based representation.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

Cost on a Vax 11/750 - 11/780 class machine is \$50K/user with a \$10K/year maintenance fee. On Vax 8600 class machines, the cost is \$100K/user and maintenance is \$20K/year. A four-day, basic and advanced training course would cost \$1700/person.

17. Indicate the hostability (measure of degree of portability) of your system.

ADAS will run under DEC's VMS operating system or UNIX. Workstations, including Vaxstations II, MicroVax II(GPX), and Sun 3/160 are supported. Work is under way to port ADAS to the Apollo workstations.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

The methodology on which ADAS is based supports system design by decomposing the system into separate software and hardware designs. The two are then combined to form the integrated system design. Thus, the methodology supports multi-disciplinary design, allowing computer architects and software engineers to work in parallel on the design. Each can understand the ADAS representations.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

The methodology is complete in the area of system design and development, and the area of software program design. ADAS is structured so the designer defines the system as interconnecting functions, thus enforcing a very structured design discipline. Project control and management tools are in the initial design stages.

20. Describe how your system supports a team development approach. (Number of stations/users).

ADAS allows any number of users and is set up so that portions of the system can be developed independently and later integrated together to form the total system. This allows users with different areas of expertise to work independently of the total system before bringing it all together.

21. Describe how your system supports design trade-offs.

One of the major advantages of ADAS in design and analysis of systems is the ease with which the system model can be created, changed, and re-analyzed to evaluate system trade-offs. The information reported from an analysis or simulation can be easily checked to determine the effect of changes even when changes are made during a simulation.

22. Indicate the range of problems to which the system can be applied.

ADAS is best suited for multi-algorithm, multi-processor system design. Generating of microcode for a custom chip set, mapping instructions to the most optimum areas of a CPU, and determining the best hardware configuration for a specific software function are just a few of the applications for which ADAS has been used.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

Todd Carpenter
Honeywell
3660 Technology Drive
Minneapolis, MN 55418
612/782-7412

Dan Nash
Raytheon Company, Missile System Div.
Hartwell Road
Bedford, MA 01730
617/273-9004

TRW
c/o
U.S. Army Strategic Defense Command
ATTN: DASD-H-SBY
P.O. Box 1500
Huntsville, AL 35807-3801

TRW

1. Describe how your system supports early detection of inconsistencies, closure and errors.

The system allows all requirements, design, and test information to be recorded in an element-relationship-attribute (ERA) data base. Because the information is in a data base, it may be queried in various ways to allow analysis of completeness and consistency after the completion of a methodology phase. The DCDS query and analysis systems provides automated analyses which provide the user a listing of anomalies in his specification compared to what should have been defined in the data base, at that point in the development process.

The primary objective of DCDS is to incrementally generate and verify requirements, design, and test information in such a way as to increase software development productivity and to attain greater software reliability. These two benefits are expected to be achieved through the following ancillary benefits:

1. Increased Understanding of Distributed Software Development -- by identifying a sequence of decisions to be made to accomplish the development, identifying the criteria for making the decisions, providing a method for representing the intermediate products, and identifying consistency criteria, the nature of the design process is more fully exposed and can be better understood by software developers. Increased understanding of the software development decisions results in a separation of concerns so that needed development decisions can be made in the correct order, at appropriate times, and with less iteration.

2. Early Requirements Emphasis -- system and software engineering in the development of a Data Processing System (DPS) and the requirements for the software that is to run on it concentrates on developing clear, complete, consistent requirements early in the development phase. This approach has been shown to have an important role in reducing software breakage, since otherwise, the "real" requirements are often uncovered late in the development, thus requiring iteration back through the development phases.

3. Automated Consistency/Completeness Analysis -- a significant objective of the DCDS methodology and language design was the identification of consistency/completeness criteria which could be verified using automated tools. This became a driving factor for determining what should be included in each DCDS data base. These criteria are derived from the semantic rules

established in each phase during which data base entries are accomplished and assure that human errors don't creep into the data base undetected.

4. Discipline -- a primary benefit of any methodology is the formulation of a systematic method for performing an activity. This serves to focus the developer on the specific decisions to be made, the appropriate order in which to make them, and provides object milestones for the measurement of progress. The expected result is both lower cost and increased reliability, because everything is done when it is supposed to be done, and in context with other DCDS methodologies with which it may interface.

5. Strong Traceability -- this means that the consistency achieved at the software requirements level is preserved by the design, that any modifications to the requirements level is preserved by the design, that any modifications to the requirements can be directly traced to the portions of the design which implement them or tests that validate them. Further, any portion of the design can be traced back to the satisfaction of some requirements/design decision. In addition, traceability between key elements of each data base are provided so that change of any such elements automatically identified upstream and downstream elements that may require changes. If the properties of the requirements are preserved in the design by construction, errors from poor traceability should be absent from the code, and the reliability of the end product is thus increased. Furthermore, the cost of performing modifications of the design in response to modifications to the requirements during the implementation and maintenance phases should be reduced by the established traceability.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

No automatic metrics are provided. However, knowledge of progress status is known because of knowledge of which of the 34 methodology phases the user is working in. Each phase or activity has a recognizable completion criteria so that explicit movement from phase to phase can be recognized.

3. Describe how your system supports documentation, program management and control.

a. Documentation: Central Project Master Data Base and the Query capability facilitates the creation of system documentation (textual and graphically) from the specification and design information in the data base.

b. Program Management and Control: The program manager is supported by the overall Distributed Computing Design System (DCDS)

Concept which is designed to increase software development efficiency and effectiveness; to thoroughly address requirements to gain the cost and schedule benefits of getting this done correctly, understandably, and early; and to provide life cycle traceability from requirements to test. The ultimate benefit to the program manager is reduction in risk.

4. Describe how your system supports real time design.

DCDS was designed from the beginning to provide the means to handle real-time design, concurrency, and parallelism. Real-time design is the province of the Distributed Design Methodology (DDM) where means are provided to express the design of individual software tasks the condition under which they are available for dispatch, their relative priorities, and the application operating system rules for determining dispatch. The design resident in the data base is directly usable for design simulations to establish that real time processing can be accomplished. DCDS incorporates a mechanism for identifying and marking performance validation paths within the system model.

5. Describe how your system supports concurrency, parallelism.

Concurrency and parallelism is handled initially during system analysis by the System Requirements Engineering Methodology (SYSREM), which is an extension of the techniques described in Paragraph 10.2 of MIL STD-499 (USAF) and in the Department of Army Field Manual 770-78, particularly the application of Functional Flow Block Diagrams (FFBDs). FFBDs provide a powerful means of depicting system functional flow, but leave unanswered other important system issues needed to attain adequate system definition. The more important ones are as follows:

- How to decompose and allocate system performance, not just system functions.

- How to accommodate the need to simultaneously deal with concurrency or parallelism.

- How to clearly separate the standard (expected) system operation from the operational constraints caused by subsystem failures, limitation of available resources (e.g., interceptors), etc., in order to attain a clear perception of system operations under various conditions without cluttering up the standard functional flows.

- How to determine when functional decomposition is sufficiently complete to support proper allocation to the components of the system.

-How to derive needed coordination functions to control competing needs of various functions of the system.

-How to derive needed interface functions to support communication between system components.

-How to define the conditions for transitioning from one system function to another.

SYSREM was motivated in its design to extend MIL STD-499 and FM 770-78 concepts to address these issues and to provide a formal way to capture system definition in a data base such that automated consistency/completeness checks can be applied.

Through SYSREM, parallel processing possibilities are identified graphically during the functional decomposition process. These possibilities can be implemented through the techniques provided in the Distributed Design Methodology (DDM).

6. Is your system constrained to a particular implementation language (Ada)?

No. The DCDS Specification Language supports the use of any implementation language. Both Pascal and Ada have been used to date.

7. Does your system produce Ada PDL?

It supports the use of PDL during the design phases to express design. An early version of an Ada PDL Processor is available in the DCDS toolset.

8. Describe how your system supports life cycle intraphase & interphase communications.

The DCDS methodologies support each of the software life cycle phases. Within each methodology a specified set of language constructs are used to explicitly capture requirements, design, or test information (depending on which methodology is involved). Appropriate portions of the data base products in the data base of one methodology are used to initiate the data base of downstream methodologies.

Interphase Communication is accomplished as follows:

-Methodology provides steps and instructions which guide users through each phase.

-Centralized Project DB and a query systems makes specifications available to everyone working a particular phase.

-Consistency checks are required at various junctures within phases: these helps ensure consistency among the project team members and reduce errors.

Interphase Communication is accomplished as follows:

-Information and specifications from each phase are carried forward (via the project data base).

-Traceability (from phase to phase) is supported by the methodology, the specification language, and the support environment (software tools).

9. Is your system automated, executable, compilable?

DCDS supports automatic construction of executable modules from the system architecture described in the data base.

The underlying software providing the tools to support DCDS are provided in an executable version for use on a VAX. Although currently written in Pascal, an Ada version will be released in October of 1987 which will support transportability to other systems (i.e., IBM PC/AT).

10. Describe the graphics support for your system.

DCDS provides graphical interfaces fully integrated with textual representation of functional control and data objects for both system level requirements networks (F_NETS) and software requirements networks (R_NETS). The graphical structures provides a method illustrating concurrency, sequence of operations, and decision points within the requirements networks. Enhanced capabilities to allow for definition of any logical set of structured flows via a generic structure processor are under development with completion in September 1987. DCDS graphics are designed for portability using the DI-3000 package on the VAX. This allows (with graphics software emulators) use of an IBM PC as a graphics terminal. Note that all data is textually maintained in the Project Master Data Base such that full functionality of the system can be obtained without graphics--for instance with a VT100 terminal.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics
- Macros
- Data flows
- Control flows

The DCDS concept of early development of critical algorithms and their evaluation via simulations built rapidly from the data representation of the system architecture supports both early and rapid prototyping.

The methodology assumes an archive of reusable algorithms is available and directs the module designer to consider and select algorithms to be used in the system.

The methodology is geared toward decomposing a system into specification packages which identify the interfaces, functions to be performed, and the performance requirements associated with the functions. This approach allows the design of individual functions to proceed with minimal knowledge of other parts of the system.

The methodology and its supporting specification languages support the development of data structures from a high-level specification in requirements to a low-level specification in design. Also DCDS directs the user (software engineer) to functionally decompose a system down to the stimulus-response level. This requires the software engineer to think at various levels of abstraction as the system is being developed. The subnet facilitates thinking at higher levels and decomposing into lower levels.

DCDS requires the use of data object typing to define data structures with the system.

DCDS supports the development of systems in threads/paths which correspond to primary interfaces or messages/transactions to be processed by the system.

This partitioning of the system, according to its interfaces, supports the DCDS model(s) and specifications may be changed or expanded appropriately.

A special version of DCDS supports the definition of Ada generic units.

DCDS contains capabilities to generate hierarchy/data flow structures from the requirements and design specifications stored in the Project Master Data Base. Also, a static data flow analyzer tool is provided.

The requirements networks (R_NET, F_NETs) provide conditional nodes (i.e., "or" node) used to specify major decision points with respect to the sequence of the specified processing steps.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

An overall paradigm for DCDS is resident in the model shown in the diagram and tables following the questionnaire response. Some of the methodologies also incorporate their own paradigms for achieving requirements and design.

13. Describe the external tools with which your system interfaces (tool compatibility).

DCDS has an Open Architecture for assimilating externally-developed tools and transferring information to/from external tools. The common interface is the Project Master Data Base (ERA Model) with the automated tools used to enter, extract, and format data stored in the data base. The current DCDS integrated tool-set consists of the following major functions:

- Translation: Analyzes the DCDS specification language input statements and subsequent entries into the data base corresponding to the syntactic/semantic definitions. This function provides initial consistency checking of the inputs.

- Extension: Allows modification to the nucleus of elements, relationships, and attributes of DCDS languages to permit expansion of language constructs to meet special user/project needs.

- Completion and Consistency Analysis: Provides static flow analysis and the capabilities of a generalized extractor system for checking completeness and consistency against semantics rules for the data base contents.

-Database Query and Retrieval: Provides output from the data base via sets defined by user query commands to allow manual and predefined analysis and automated documentation.

-Graphics: Allows interactive development or modification of structure graphics and produces hardcopy plots of the structures for documentation.

-Simulation: Provides the means to create, execute, and analyze simulation of the functional operation of a system under development in terms of the requirements/design established in the data base.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

DCDS provides a logical topdown decomposition of the problem. It uses hierarchical decomposition only at the systems analysis level to establish System or Segment Specifications. Unique rules are provided for system decomposition to assure that a functions decomposition defines its subordinate function within a control flow context that allows parallel representation of system actions.

DCDS uses an augmented functional decomposition approach to develop requirements and designs. It is not just a simple, top-down method of decomposition. The sequence of functions, potential for concurrency, and control flow are combined with functional decomposition to produce requirements and design models. Additionally, DCDS encourages some bottomup development during the early stages of the system development process when critical assumptions and concepts must be simulated, tested, and proven. This approach provides a clear, concise understanding of required software actions greatly reducing the ambiguities that abound as the traditional functional decomposition approaches offered by other systems.

Design creativity is enhanced by achieving a clear understanding of system and software requirements early, by providing a road map (not a constraining cookbook) of a suggested sequence of developer actions in the methodologies, and by providing underlying tools to assist the user in every phase of the system definition and software development.

Aspects of object oriented design are captured in some abstractions and approaches used in DCDS. Objects of interest are introduced in the software requirements phase where the required data transformations are precisely defined on the processing logic diagrams described above.

15. Is your system supported by formal syntax & semantics? Describe briefly.

The languages used to define the requirements, design, and testing all have a formal syntax. In addition, certain semantic rules are expected to be followed by the users. For example, if the user defines an input message, he is expected to assure that he also defines its data contents. These semantic rules are the basis for the predefined consistency checks available for use in the various methodology phases.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

This is a Government owned system. Although there is no cost to acquire it, Government approval must first be attained. In addition, it currently is not being released to foreign users. Also, training material is currently under development.

17. Indicate the hostability (measure of degree of portability) of your system.

The underlying DCDS software is currently hosted on a VAX using the VMS operating system. Additionally, the Ada version of DCDS is currently being re-hosted onto an IBM PC/AT.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

Questions 18 and 19 are similar in nature and scope. Therefore, only one answer was developed. The answers for both questions is presented under Question 19.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

DCDS is an extremely comprehensive set of five methodologies. It incorporates 21 phases and approximately 110 steps. The five methodologies within DCDS are:

-System Requirements Engineering Methodology (SYSREM) for defining and specifying system requirements, with an emphasis on the data processing subsystem.

-Software Requirements Engineering Methodology (SREM) for defining system software requirements, with an emphasis on stimulus-response behavior.

-Distributed Design Methodology (DDM) for developing a top-level architectural design for the system software, including distributed design, process design, and task design.

-Module Development Methodology (MDM) for investigating and selecting algorithms, defining detailed design, and producing units of tested code.

-Test Support Methodology (TSM) for defining test plans and procedures against requirements, producing an integrated tested system, and recording test results.

The diagram following the questionnaire response illustrates how the DCDS Methodologies synergistically work together to support the system development life cycle from system requirements through integration testing. In addition to supporting the traditional system life cycle phases, DCDS supports project management activities. The DCDS project master data base is a repository for storing system specification, decisions, assumptions, etc., which may be interrogated or summarized in reports for project management. The DCDS system specifications may be reviewed by management for accuracy, completeness and consistency using the DCDS query capability.

DCDS also supports the management of software development. Specifically, the Automated Unit Development Folder consolidates the requirements, design, codes, test plan, schedule, etc., into one file which may be updated and queried by project management.

**20. Describe how your system supports a team development approach.
(Number of stations/users).**

DCDS has contemplated the probability that most systems are developed by teams of various disciplines and the methodologies are designed with that in mind. Typical users are:

SYSREM: System Engineers
 SREM: Software Requirements Engineers
 DDM: Distributed Process Designers (i.e., design system
 software architecture)
 MDM: Detailed Designers, Implementors
 TDM: Integration Testers

Each has been given his own language set to define the results of his efforts in data bases. These may be distributed for use by various teamed contractors or for separate departments within the same company. With appropriate configuration management, multiple users may work with a single data base.

21. Describe how your system supports design trade-offs.

Data bases provide the information to support simulations at different levels of detail. The logic structures (specify functions, control/data flow, interfaces and performance objectives) that are defined in SYSREM (under development) and in SREM (exists) provides the simulation framework. Thus, if the results are not as expected, there is no issue as to whether the requirements or the simulation fidelity is the problem. When the requirements, represented by the logic diagrams, are modified to improve the system's performance, the simulator is automatically changed in accordance with the changes to the logic diagram structures. Other generic simulation output exists for design trade-offs that allow the user to tailor a simulation to the level of fidelity appropriate for his design, and then to allow the simulation to run using the contents of his design data base as a direct input to the simulation.

22. Indicate the range of problems to which the system can be applied.

DCDS is designed for large, distributed, real time, embedded systems, such as that contemplated for SDI. However, it can also support smaller system, monolithic systems, non-real time systems, or non-embedded computer systems. The user simply skips the portions of the methodology that do not apply to his application.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

Mack Alford
GE
P.O. Box 8555
Building 7, Room 7236
Philadelphia, PA 19101
(215) 354-2035

Barry Boehm
TRW
Building 02 (Room 1310)
One Space Park
Redondo Beach, CA 90278
(213) 535-2184

David Palmer
GRC
P.O. Box 6770
Santa Barbara, CA 93160-6770
(805) 964-7724

Larry Marker
TRW
213 Wynn Drive
Huntsville, AL 35805
(205) 837-2400

Cliff Barkley
TRW
213 Wynn Drive
Huntsville, AL 35805
(205) 837-2400

Alice Brown
Nichols Research Corporation
4040 Memorial Parkway
Huntsville, AL
(205) 833-1440

Wayne Smith
GRC
307 Wynn Drive
Huntsville, AL 35805
(205) 837-7900

Carolyn Pasini
TRW
Building 119 (Room 4824)
One Space Park
Redondo Beach, CA 90278
(213) 217-6581



Table 1-1. The Phases of the System Requirements Engineering Methodology (SYSREM)

- Phase 1: Define System - Supports transformation of the need into a formal problem statement of the semi-closed system, which includes the system and its environment. The top-level inputs and outputs and the performance indices of the system are defined.
- Phase 2: Identify Subsystem Configurations - Supports definition of subsystem configurations which describe potential classes of solutions to the system problem. This allows definition of the classes of inputs and outputs of the system regarded as a black box containing the defined subsystems.
- Phase 3: Decompose System Logic - Supports transformation of the semi-closed system requirements into the open system's requirements by decomposition to the level where the system's perception of the items in the environment occurs. This allows identification of the sequences of inputs and outputs between the system proper and its environment.
- Phase 4: Decompose and Allocate to Subsystems - Supports continued decomposition of the functional requirements until they can be uniquely allocated to the subsystems defined in Phase 2, and allows identification of the subsystem interfaces.
- Phase 5: Determine Subsystem Feasibility and Cost - Based on the allocated requirements, supports estimation of resource utilization, cost, and schedule for each subsystem; identification of potential failure modes; and definition of subsystem level critical issues.
- Phase 6: Solve Subsystem Critical Issues - Supports evaluation of analysis results of subsystems to identify system-level approaches to solve critical issues to achieve required system performance.
- Phase 7: Develop Exception Layer Decompositions - Supports development, of the additional layers of requirements needed to deal with exceptions caused by system failures and resource limitations. These new functions are then decomposed and allocated, as described in the preceding phases.
- Phase 8: Plan System Integration and Test - Supports establishment of the plans, schedule and costs for development, integration, and testing of the developed subsystems to yield the desired system.
- Phase 9: Optimize Over Designs - Supports analysis of alternative system designs (subsystem configurations) for selection of the final system.
- Phase 10: Produce System/Subsystem Specifications - Supports production of the System Segment Specifications in accordance with customer formats in preparation for the System Requirements Review or the System Design Review.

86-03-7825

Table 1-2. The Phases of the Software Requirements Engineering Methodology (SREM)

- Phase 1: Define Elements and Build Data Base - Supports initial definition in an RSL data base of the DPS requirements by identifying the interfaces, message flow and contents, global information, and the processing flow to respond to input messages.
- Phase 2: Evaluate Kernel - Supports consistency/completeness checks of the data base produced in Phase 1 to provide the software engineer a list of anomalies for his correction.
- Phase 3: Complete Functional Definition - Supports detailed definition of data and additional consistency/completeness checks to assure correct membership of data within messages and repetitive sets. Also supports evaluation that all data is both used and produced within the defined processing logic and analyzes the logic of the defined data flow. Provides information of anomalies for correction.
- Phase 4: Complete Management and Control Information - Supports examination of the completeness of traceability within the data base and the status of open issues for solution and provides identification of needed corrections.
- Phase 5: Accomplish Dynamic Functional Validation - Supports building and executing a functional simulation to accomplish a dynamic evaluation of the DPS requirements directly from the data base. This provides a means to examine the DPS system operation within an environment defined in a user-built simulation driver.
- Phase 6: Develop Performance Requirements - Supports identification of paths on the processing flow diagrams to which performance requirements can be allocated and the points on these paths where data can be recorded for use in evaluating attainment of the performance requirements. Also supports definition of how the system uses the recorded data to determine a pass/fail criteria for each performance requirement.
- Phase 7: Accomplish Analytic Feasibility Demonstrations - Supports evaluation of different algorithmic approaches within the context of the processing flows in the data base and to evaluate allocated performance requirements for pass or fail within the environment modeled by the simulation driver. Assists in identification of infeasible and non-testable requirements.
- Phase 8: Produce Software Specification - Supports production of the Software Requirements Specification in accordance with customer formats in preparation for the Software Specification Review.

86-03-7826

Table 1-3. The Phases of the Distributed Design Methodology (DDM)

- Phase 1: Accomplish Geographical Node Design - The first of four overlapping levels of design which defines the geographically separate network of nodes. Results in requirements and constraints for local node design.
- Phase 2: Accomplish Local Node Design - The second of four overlapping levels of design which defines the design of local nodes or groups of computers or computer systems connected by a local communication system and interfacing with the communication network connecting the geographical nodes. Results in requirements and constraints for computer-system design.
- Phase 3: Accomplish Computer-System Design - The third of four overlapping levels of design which defines the internal design of local computer systems based on common architectural considerations. Supports identification of the details of the interconnection networks and protocols between the local computer systems. Also supports decisions as to allocation of requirements between various software levels and the hardware. Results in requirements and constraints for local process design.
- Phase 4: Accomplish Local Process Design - The last of four overlapping levels of software design which defines the details of the processes to be allocated to the various computer systems. Includes definition of application tasks and their budgets scheduling/dispatching criteria, priority structures, intertask communication, error handling, overload control, and process control. Also supports definition of global data management, access protocols, and interface refinement. Results in the requirements and constraints for task design.
- Phase 5: Accomplish Task Design - Supports application of process and task design rules to identify the task control routine and its layers of supporting application routines. This includes opening Automated Unit Development Folders (AUDFs) which are used to organize, in a single place, all aspects of the development of each unit of code from requirements to unit test for the tasks to be designed. Results in the requirements and constraints for the detailed design implemented in the Module Development Methodology (MDM) of DCDS.
- Phase 6: Produce Design Specification - Supports production of the Software Top Level Design Document in accordance with customer formats.

86-03-7827

Table 1-4. The Phases of the Module Development Methodology (MDM)

- Phase 1: Identify and Analyze Potential Algorithms - Supports identification of algorithm needs and their interfaces, evaluation of existing or new algorithmic approaches, and determination of timing and sizing estimates of candidates. Candidates are offered for consideration by the process designer.
- Phase 2: Define Reusable Modules - Based on the evaluations of Phase 1 and Process designer feedback, supports selection of a candidate set of algorithms for development and supports reusable module design of selected critical algorithms.
- Phase 3: Accomplish Design of Routines - Supports preliminary achievable design in PDL for routines to accomplish algorithms that are not using existing reusable modules. These preliminary designs, their requirements, and their interfaces are used to initialize Automated Unit Development Folder (AUDF). After PDR, supports development of detailed designs.
- Phase 4: Develop Unit Test Plan - Supports development of the unit test plan in the AUDF. Also supports definition of test procedures, test cases, pass/fail criteria, test drivers, and data loggers for unit tests, where appropriate.
- Phase 5: Code and Unit Test - Supports development of the in line code to replace the PDL comments between the PDL control flow statements of Routines entered into the AUDF. Supports prevention of control flow changes within Routines without Task Designer approval. Also supports recording of unit test results in the AUDF.

86-03-7828

Table 1-5. The Phases of the Test Support Methodology (TSM)

- Phase 1: Prepare Initial Test Approach - Supports initiation of the TSL data base from the SSL, RSL, and DDL data base to capture the information necessary to produce the requirements/verification matrix, and the test/requirements matrix.
- Phase 2: Develop Test Plans and Environment - Supports definition of the major segmentation of the test program into builds, and schedules their time sequence. Test names and test aids are identified and scheduled, and a preliminary test plan is produced.
- Phase 3: Develop Test Procedures - Supports detailing the test procedures and test cases with inputs, expected outputs, and pass/fail criteria. All necessary drivers, harnesses, simulators, and stubs are developed and tested. Uses Automated Test Development Folder (ATDF) and the TSL Data Base to organize and plan testing for each build.
- Phase 4: Perform Integration Tests - Supports the integration testing and production of periodic status reports and the final integration test report. Uses the ATDF and TSL Data Base to record test results. The individual builds transition into this phase separately, according to their schedule.
- Phase 5: Perform Acceptance Tests - Supports the test program for formal qualification testing. Acceptance test plans, test procedures, and test reports are produced the same way as for integration testing.

86-03-7829

Intermetrics, Inc.
733 Concord Avenue
Cambridge, MA 02138

Intermetrics, Inc.

1. Describe how your system supports early detection of inconsistencies, closure and errors.

The use of PSL/PSA for requirements analysis and system specification assists in the early detection of large scale design flaws. This is the purpose for which PSL/PSA was designed, and has proven valuable in years of industrial use. PSL/PSA does this by identifying overlapping, redundant or inconsistent information in the design description. This is accomplished by examining a graph of the structure of the design.

Once the system has proceeded to lower level design and the system has been translated to Byron, errors and inconsistencies continue to be easy to find, because it is easy to trace code back to the requirements that a specific piece of code is intended to satisfy. This also makes it possible to determine whether all requirements have been addressed, which assists in determining whether closure has been satisfied.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

After the software system being created reaches the point where the Byron tools can be applied, each unit in the system is marked to indicate what phase of the life cycle it has reached. As the code evolves, it gradually passes through each of the phases: specification, design, coding and test. At any point it is possible to query the system as to what phase any given unit has reached.

In addition, tools to perform such metric analysis as the Halstead metric have been written based on the information available in the Byron data base. Thus, the potential exists for many such tools, although they are not part of the current set of tools supplied with Byron.

3. Describe how your system supports documentation, program management and control.

The Byron system includes a document generator and formatter whose purpose is to simplify the process of creating documents which describe the system. The user describes the document which is to be produced in a high level language called Bdoc, informing the processor what information to extract from the code and how to format it. The user currently receives five document templates with the Byron system: a calltree generator, a user

manual generator, a data dictionary generator, a dependency table generator, and a 483 style c5 generator. In addition, work is in progress on templates to generate many of the 2167 style documents. The high level language Bdoc makes it easy to create new documents, and it is easy to keep documents current because they are generated directly from the code and the comments. Thus control of document versions tends to become mostly a matter of controlling versions of system code.

4. Describe how your system supports real time design.

Real time design is supported by the PSL/PSA tools through the ability to describe timing and state transitions within the system. It is possible to analyze the behavior of the system based on the information described. When the design passes to the Byron system, the user may apply all the usual Ada constructs associated with real time applications, and may tailor the use of the Byron PDL to support the use of these constructs.

5. Describe how your system supports concurrency, parallelism.

The PSL/PSA tools permit the user to easily describe parallel processes, rendezvous, and the like, and to analyze systems described in this manner. When the design passes to the Byron system, the user may apply all the usual Ada constructs associated with real time applications, and may tailor the use of the Byron PDL to support the use of these constructs.

6. Is your system constrained to a particular implementation language (Ada)?

The PSL/PSA system is wholly language independent. The Byron system relies on an Ada based PDL and is therefore slanted toward Ada. However, the system has been used to create both Fortran and CMS2 code in the past, and other languages would present no different problems. One customer does not even own an Ada compiler. The Byron system is most easily and naturally used, however, when the implementation language is Ada.

7. Does your system produce Ada PDL?

The system is able to produce Ada PDL at several stages. First, at the juncture between PSL/PSA and Byron as described above. Secondly, Ada bodies containing both comments and skeletal code (a PDL description) can be generated from Ada specifications already analyzed into the Byron database.

8. Describe how your system supports life cycle intraphase & interphase communications.

The ease with which one can communicate between phases depends on which phase one is in. It is easy to communicate between requirements analysis and system specification, and also between the phases following design. This is because the form of the design during requirements analysis and system specification is the PSL/PSA form, while the form for the rest of the life cycle is that of the Byron PDL. While the information is in one form or the other, the phase delineations are fairly loose. Basically, whatever form is applicable is being expanded on in whatever manner is appropriate for the phase one is in. The information created during the previous phases is present in the form that is being expanded upon.

The communication between the system specification phase and the design phase is more rigorously controlled. When the PSL/PSA code is deemed complete, a tool is used to create Ada/Byron code skeletons based on the PSL/PSA code, incorporating the information available in the PSL/PSA code in the Ada/Byron code.

9. Is your system automated, executable, compilable?

The system is comprised of a series of executable tools which assist in the analysis of statements of system design. These statements take two forms, input code to the PSL/PSA system and input code to the Byron system. The Byron code is in the form of Ada code with structured comments and so is compilable. There is a tool which automates the translation from PSL/PSA code to Byron code. In addition, facilities are provided to permit the user to tailor existing tools to create new tools reporting on the information in the descriptions of the system.

10. Describe the graphics support for your system.

The system runs primarily on mainframes; however, a PC-based front-end is available with PSL/PSA which is complete structured analysis workstation combining analysis, graphics, and documentation facilities in a flexible working environment. The Byron tool itself has no graphical interface. It is possible, however, for Byron users to write both graphical and interactive reports using the Byron Program Library Access Package (a set of database query packages written in Ada and provided with the product).

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics
- Macros
- Data flows
- Control flows

It is possible to create programs with the Byron Program Library Access Package which will examine the Ada specifications which make up the high level design, and automatically generate skeleton bodies for them which simulate the intended working of each Ada unit. Thus it is possible to create a preliminary executable at an early stage in the design work.

Since the documentation for each unit is included in and with the code, the code tends to be better documented, and the documentation is certain to be current and in sync with the code. Removing these pitfalls make code more easily reusable.

Because of the structure of the data base in which the information about the code is stored, code is naturally divided into interface and implementation code. Interface code is generally just those Ada specifications that the outside world needs to access, while implementation code tends to be not only the bodies for those specifications, but also whatever utility packages are used to handle internal details which users of the system as a whole should never know about.

The Ada packaging mechanism is fully supported. In addition, the data base contains the concept of "catalogs" which are divided into the interface and implementation parts described above. These catalogs provide a higher level packaging mechanism. For instance, a user might have a trig package, a numerical integration and differentiation package and a linear programming package, all of which might be grouped together to form a math catalog.

When the system is initially described using PSL/PSA, a large degree of abstraction is possible. As the system evolves, the abstraction mechanisms gradually become those mechanisms provided by Ada.

Byron provides full access to Ada typing mechanisms.

The system permits code to evolve easily as new requirements or new implications of old requirements are discovered. This may result in newer portions of the system undergoing design while older portions are in the coding phase of the life cycle, but this poses no problems for the system.

Byron provides the full capability of Ada generics.

It would be possible to write a macro processor using the Byron Program Library Access Package, but no tool for this purpose currently exists.

PSL/PSA permits data flow analysis as well as other design methodologies. In addition, the Byron data base maintains the information needed to construct data flow diagrams, although no tool is currently provided to perform this analysis.

A calltree generator is provided with the Byron tools. Information exists for more detailed flow of control analysis, but current tools do not take advantage of it.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

The PSL/PSA paradigm is that a system may be described at a high level as a series of objects or entities and the relations between them. These relations may take many forms and describe the interactions between the objects/entities.

The Byron paradigm is that after a system has been specified, it is possible to describe it using Ada code and structured comments. In general, the design will begin primarily as structured comments, and code will be added as the design and coding get more complete.

13. Describe the external tools with which your system interfaces (tool compatibility).

It is easy to interface the Meta-Byron tools with other tools, because both PSL/PSA and Byron provide tools to permit the user to extract information from the data base and create a report. Carefully handled, this report then

becomes the input to the tools in question. Also, the tools within Byron are loosely coupled. For instance, the Byron document generator outputs a file which is passed to a formatter for final formatting. Some users prefer to use a formatter other than that supplied with the system, so they decouple the formatter and install their own. The only necessary modification is to have the document generator output slightly different formatting commands, and this is easily accomplished.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

The PSL/PSA portion of the system supports a variety of design methodologies, including Yourdon, Warnier/Orr, DeMarco and Object oriented Design. The core of PSL/PSA is the Problem Statement Language, which permits a user to describe a system in terms of objects and the properties of, and relations between these objects. Different architectures will result from different interpretations placed on the objects. For instance, the objects may represent data, or they may represent program units. When the design passes to the Byron portion of the system, the methodology used may again vary. Depending on how the Byron system is configured, different methodologies may be supported.

In general, the natural mode of the system is hierarchical decomposition. As the system evolves, the task becomes better understood and the need for new units becomes apparent. This does not, however, preclude the use of bottom-up design techniques: If a given set of packages is available, that will tend to influence the design, as people write specifications which the existing packages satisfy and incorporate them in the system.

15. Is your system supported by formal syntax & semantics? Describe briefly.

The PSL/PSA system supports a language called the Problem Statement Language. This language permits the description of objects in terms of their properties and their relations with other objects. For instance, the statements

```
DEFINE PROCESS new-employee-processing;  
  DERIVES hired-employee-report;
```

defines a process object "new-employee-processing" which creates as an output "hired-employee-report".

The Byron system supports a superset of Ada, that is, Ada with the addition of structured comments. These comments take the form

```
--|<ada_identifier:><txt>
```

where the `ada_identifier` is a keyword describing what kind of text the comment is expected to contain. Examples of keywords are "overview," "algorithm" and "modifies." The user is permitted to define keywords by stating the name of the keyword, what kind of Ada entities it may be associated with, and the life cycle phase when it is expected to be specified.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

	Byron Costs		
	Microvax	Vax 7xx	Vax 8xxx
per primary CPU	\$15,000	\$25,000	\$35,000
required training (2 persons)	<u>\$ 1,100</u>	<u>\$ 1,100</u>	<u>\$ 1,100</u>
TOTAL	\$16,100	\$26,100	\$36,100

Maintenance for first year is included. After that, it is 20% of the purchase price.

PSL/PSA Costs

approximately \$120,000 total (rough order of magnitude)

Includes: 5 copies of Structured Architect
 1 copy of Architect Integrator (including report specification interface)
 PSL/PSA
 Meta-Plus (PC graphics package)
 8 person training in Ann Arbor, Consulting

17. Indicate the hostability (measure of degree of portability) of your system.

Meta-Byron will be available on any DEC/VAX system running VMS. IBM re-hosts to various operating systems are planned. Byron as a stand-alone tool is available on IBM 370 architectures running either CMS or MVS, DEC/VAX-VMS, and the Sperry 1100 series. PSL/PSA as a stand-alone tool

available on all of the above systems as well as DEC ULTRIX and Apollo UNIX.

Byron itself is written almost entirely in Ada, as it is fairly easy to rehost it to any system which has an Ada compiler. However, a large amount of code is involved, so a rehost is not necessarily a fast or cheap job.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

The system directs its focus toward the creation of software, so very little support for tasks such as hardware design is provided. It is possible to use the system to express the requirements which such tasks place on the software.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

The system attempts to be as methodology free as possible. The PSL/PSA portion of the system can support the methodologies of Yourdon, Warnier/Orr and DeMarco as well as Object Oriented Design. The Byron portion of the system is similarly flexible. Nevertheless, if the user doesn't want to use any established methodology, there is a default methodology. This is primarily a design and programming methodology, dealing with the information which should be present in the code at various stages of the life cycle.

20. Describe how your system supports a team development approach. (Number of stations/users).

In general, there are no limits to the number of users on the system. The practical limit is that of the computing power of the machine being used. Other than that, PSL/PSA has been used on many large projects, and Byron has been used in at least one project which peaked at approximately fifty users.

21. Describe how your system supports design trade-offs.

It is possible to include design rational in the design, and to describe possible future enhancements in ways that make them easy to find when time or money become available to perform the enhancements.

22. Indicate the range of problems to which the system can be applied.

The system assists the user in creating software throughout the entire life cycle, from requirements analysis to test and maintenance.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

Paul Szulewski
Charles Stack Draper Laboratories
M.S. 71
555 Technology Square
Cambridge, MA 02139
(617) 258-1832

Larry Gilchrist
Hughes Aircraft Co.
M.S. 618/L215
1901 W. Malvern
P.O. Box 3310
Fullerton, CA 92634
(714) 732-5760

Paul Wood
UNISYS Corporation
M.S. Y11A6
UNISYS Park
P.O. Box 64525
St. Paul, MN 55164
(612) 456-7300

Mike Wheeler
Grumman Melbourne Systems Division
Huntington Quad #3
4th Floor, M.S. J03-118
Melville, NY 11746
(516) 752-3169

Bob Calland
Naval Ocean Systems Center
Code 624 (B)
San Diego, CA 92152-5000
(619) 225-6231

Integrated Systems, Inc.
101 University Avenue
Palo Alto, CA 94301-1695

Integrated Systems, Inc.

1. Describe how your system supports early detection of inconsistencies, closure and errors.

AutoCode/Ada supports the early detection of inconsistencies, closure, and errors in the following ways.

a. Error checking on entered data is performed automatically as the elements of a design are assembled from a library of building blocks. This ensures that all data definitions are consistent with the intended function.

b. The graphical interface automatically provides an unambiguous visual representation of the functionality and interconnection of all elements of a system. This gives visual feedback on inconsistencies.

c. The model linker that prepares system models for simulation and code generation performs extensive checking for completeness of model specifications and inconsistent interconnections.

d. A comprehensive simulation capability available with *AutoCode/Ada* provides the ultimate test for accuracy and completeness of a design. Designs can be simulated directly or the actual Ada code that is produced by *AutoCode/Ada* can be simulated.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

AutoCode/Ada does not directly produce a progress metric. It will, however, report on missing or incompletely specified elements when requested to assemble a system. Indirectly, its automatic documentation features allow you to examine the current state of completion any system or system component.

3. Describe how your system supports documentation, program management and control.

AutoCode/Ada is based on block diagrams. These diagrams are created graphically and form the basis for documentation of a design. User specified textual documentation on system elements can be entered (and is automatically maintained in the database) as the system is created. Program management and control can be exercised by auxiliary software (dictated by the

user) that provides for configuration control, access restrictions and monitoring of the database files created by *AutoCode/Ada*.

4. Describe how your system supports real time design.

In *AutoCode/Ada*, a system design is specified as a collection of Process-Blocks. Blocks within a Process-Block are connected with each other and with the outside environment via data flows or control flows. These flows are referred to as data or control signals.

Process-Blocks can contain primitive functional blocks and perhaps other Process-Blocks (the system is fully hierarchical). A large library of predefined primitive blocks is provided for constructing computational algorithms and control logic, including state transition diagrams. At the Process-Block level, the user can specify attributes that determine computational triggering and timing of that process. This allows the user to specify both periodic and event driven real-time systems that encompass multirate/multitask applications.

5. Describe how your system supports concurrency, parallelism.

AutoCode/Ada provides a real-time model that views systems as collections of concurrent, prioritized tasks. These tasks can be scheduled for periodic execution or can be triggered by asynchronous events. Execution of these tasks is conducted on the basis of preemptive priority based scheduling. Data and control flows passing between tasks are handled such that all tasks can be safely computing in parallel. Thus, the Ada source code generated by *AutoCode/Ada* represents applications that can be targeted to uniprocessing as well as multiprocessing and distributed environments.

6. Is your system constrained to a particular implementation language (Ada)?

No. A FORTRAN version of *AutoCode* is also available. A C version is under development.

7. Does your system produce Ada PDL?

The block-diagram language used in *AutoCode/Ada* is a graphical process description language. Ada code is produced directly from these block diagrams. Virtual code associated with an Ada compatible PDL can be inserted in the generated code by the user via code generation templates that are provided as part of the *AutoCode/Ada* system.

8. Describe how your system supports life cycle intraphase & interphase communications.

AutoCode/Ada supports system development by using a structured graphical language that is applicable to all phases of the development process. The use of a common graphical notation dramatically improves communications throughout the software life cycle and eliminates transformation errors that can occur between phases. Specifications, designs and documentation are maintained in catalogs that can be accessed by both designers and implementors. During the implementation phase, one of the principal benefits of *AutoCode/Ada* is that the implementation produced by the code generator is an exact expression of the design.

9. Is your system automated, executable, compilable?

The generation of code from block diagrams is automatic. The Ada code generated by the code generator is both compilable and executable.

10. Describe the graphics support for your system.

AutoCode/Ada provides a powerful, graphics-oriented interactive interface with hardcopy support for a number of industry standard plotters and laser printers. Workstation versions of *AutoCode/Ada* make extensive use of the latest in workstation interface technology. The interface includes multiple windows, mouse driven graphic layout and manipulation, pop-up forms, and pull-down menus.

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics
- Macros
- Data flows
- Control flows

AD-A194 353

PROCEEDINGS OF THE STRATEGIC DEFENSE INITIATIVE
ORGANIZATION (SDIO) TOOL (U) INSTITUTE FOR DEFENSE
ANALYSES ALEXANDRIA VA D HEVSTEK 84 MAR 87 IDA-M-388
IDA/HQ-87-32884 MDA903-84-C-0031

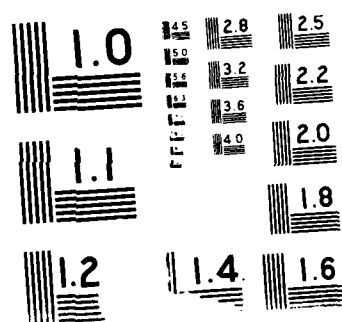
3/3

UNCLASSIFIED

F/G 12/5

NL





AutoCode/Ada facilitates both rapid and early prototyping of a real-time system. A prototype of a system design can be quickly assembled in the *AutoCode* graphical environment and then evaluated using *AutoCode*'s extensive simulation capability. An early prototype can be progressively refined into a complete design. At any point in this process, the engineer can invoke the code generator to generate the code for any subsystem within the design, thus allowing evaluation of the actual implementation code.

Instead of storing the actual Ada code for a system implementation, *AutoCode/Ada* uses a template of the code with which it regenerates the code as needed based upon a block diagram. The block diagram specification is a reusable, transportable representation of a system that can be stored in simple ASCII files.

The form of hierarchical modeling used by *AutoCode/Ada* allows the user to construct self-contained, modular components at any level. Details can be hidden within the modular components enabling the user to deal with much more abstract objects, thus reducing the level of complexity of the system being created.

Packaging, as it applies to Ada, is supported at code generation time. The code which is generated by *AutoCode/Ada* is provided as one or more Ada packages that can be supplemented by user library packages. Features of the packaging can be controlled by the user via the code generation templates.

AutoCode/Ada allows the user to view his system at any desired level of abstraction. If the user begins with general ideas that form a top level view of the system, this view can be progressively decomposed into lower levels. Finally, a level is reached that requires only primitive functional descriptions. With *AutoCode/Ada*, this decomposition process is accomplished by manipulating icons to describe the system in much the same way that an engineer sketches designs by hand.

Fundamental data types are predefined by *AutoCode/Ada* and are implicit in the use of each functional block. These types are automatically mapped into language type definitions during code generation.

A system design can evolve through progressive refinement of an early prototype. This type of evolutionary process is strongly supported by the system simulation capability in *AutoCode* which provides for continuous reevaluation of the system or any of its components. At each stage in the evolutionary process, the current design is captured and maintained in a design catalog which can be used to enforce configuration management.

The concept of templates is used extensively in *AutoCode/Ada* because each predefined building block in the graphical language is such a template. Customization of each template is provided by data supplied by the user. Also, Ada language generics can be utilized directly by the user within user specified "source code" blocks.

Text oriented macros are not used in *AutoCode/Ada*.

AutoCode/Ada is based upon a graphical block diagram language that specifies flows of both data and control signals.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

The paradigm embedded in *AutoCode/Ada* is the block diagram. It forms the complete basis for design and implementation of real-time systems.

13. Describe the external tools with which your system interfaces (tool compatibility).

AutoCode/Ada interfaces with MATRIX and SYSTEM_BUILD, which are both ISI products that support system design and simulation. Files are created by *AutoCode/Ada* in a form that can be easily monitored and controlled by user specified configuration management and control tools.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

The block diagrams created with *AutoCode/Ada* are fully hierarchical in that any Process-Block can contain other Process-Blocks. Process-Blocks can also be replicated freely. There are no restrictions on building systems from the top down, the bottom up, or the middle out. Graphical layout is free-form. The objects in *AutoCode/Ada* are well defined blocks and user created modular components that are functions of user-defined parameters and of the data flowing into them.

15. Is your system supported by formal syntax & semantics? Describe briefly.

AutoCode/Ada is based on a concisely defined interactive graphical interface that allows you to assemble systems graphically from predefined and user defined components. The semantics of all such components is formally

defined. As such, no formal syntax is required. Designs can also be created from command files, which must follow formal syntax.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

For *AutoCode/Ada* on a VAXstation, typical costs would be: 15-30K\$ for the workstation hardware, 33K\$ for the *AutoCode/Ada* software, approximately 1K\$ of direct training costs plus 1 man-month of time required to become proficient. Software maintenance costs for *AutoCode/Ada* are 20%/year. Quantity discounts are available.

17. Indicate the hostability (measure of degree of portability) of your system.

The SYSTEM_BUILD package on which *AutoCode/Ada* is largely based is currently supported on VAX systems, IBM mainframes, IBM PCs. Workstation versions include VAXstations and Apollo workstations (SUN Workstation versions are being developed.). *AutoCode/Ada* is currently available on VAXstations and will be made available on other systems upon demand.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

The block of diagrams of *Autocode/Ada* are oriented towards systems engineering. Similar block diagramming techniques are commonly used (by hand) in software engineering. *AutoCode/Ada* is a software engineering tool that supports system specification, system design, automated documentation, simulation and code generation.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

AutoCode/Ada can be used to support any structured design and development methodology. Programming is supported by automated code generation and can be used to develop executable application code (with exception of hardware device drivers). Project management and control can be performed

by auxiliary software that provides for configuration control and monitoring of the database files created by *AutoCode/Ada*.

**20. Describe how your system supports a team development approach.
(Number of stations/users).**

AutoCode/Ada supports team development through the decomposition of a system into a hierarchy of subsystems, each of which may be developed independently and then integrated. Catalogs of block diagram representations of system components can be shared between development teams as necessary. The number of simultaneous users is determined only by the host system. For example, VAX hardware can support multiple users on a single machine or distributed workstation access through a network. User supplied configuration control tools can be used to control access to systems under development.

21. Describe how your system supports design trade-offs.

AutoCode/Ada supports design trade-offs through rapid simulation and system prototyping.

22. Indicate the range of problems to which the system can be applied.

AutoCode/Ada is a real-time system design and specification tool with applications in guidance and control problems for aircraft, spacecraft, missiles, process control systems and servomechanisms. Extensive simulation capabilities are currently available with the *AutoCode/Ada* toolset (and more are under development) making it suitable for use in major simulations including battle management, trajectory evaluation, etc. Automated code generation provides a significant capability for real-time simulations including those requiring hardware in the loop.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

AutoCode/Ada is currently in beta-test at three sites:

Patrick Rogers
University of Houston Clear Lake
High technology lab
Houston, TX 77058-1068
(713) 488-5921

Alok Das
USAF/RPL
DYSS-Stop 24
Edwards AFB. CA 93523
(805) 275-5412

Niranhan Rao
Boeing Military Aircraft Company
MS K7575
P.O. Box 7730
Wichita, KS 67277
(316) 526-7918

Software Products and Services, Inc.
14 East 38th Street, 14th Floor
New York, NY 10016

Software Products and Services, Inc.

1. Describe how your system supports early detection of inconsistencies, closure and errors.

Both error prevention and early error detection are routinely supported by EPOS. A powerful set of analysis programs provide redundancy, consistency and completeness checks of parts of the system requirements, as well as checks for completeness, type conflicts, inconsistencies in the export/import relations between modules/packages, missing value assignments for data, etc. Some tool analysis is done automatically; the remainder is user-invoked. EPOS checks the requirements specification and conceptual design for:

- correct syntax.
- compatibility of formal elements input with information in the project data.
- compatibility of decision rules with decision process definitions.
- completeness and consistency of decision tables.
- redundancy.
- completeness of substitution and references to requirements.

EPOS checks the system design specification for:

- completeness of hierarchy design.
- name conflicts.
- non-referenced design components
- important hierarchical features such as data type and range consistency.
- non-contradictory synchronization
- completion of project requirements.

Additional analysis and support of automatic code generation facilities such as checks for sufficiency of design for transformability to Ada code are also supported.

2. What type of progress metric does the system produce? Is it quantifiable measure of completeness?

There is support for several representation (requirements specifications, design, program code). Several levels of abstraction -- indicating different degrees of completeness -- can be visualized. The degree of completeness can be measured by the completeness of the representation itself, and by the fulfillment of requirements from an earlier representation. EPOS uses a single unified database which integrates requirements, specification (design) and

management activities permitting a comprehensive overview of progress and development throughout the project.

3. Describe how your system supports documentation, program management and control.

Portions of (or complete) documentation can be produced in several ways from the project database. The system includes project planning (work breakdown structure, network planning) as well as project control (actual/nominal comparisons, visualization of progress, etc.). Text and graphs are produced automatically to support documentation, design and program management. EPOS will automatically produce a large variety of graphical outputs providing representations of both design and project management information.

4. Describe how your system supports real time design.

All necessary real time constructs are provided in the specification language at a very high, problem-oriented level, and are therefore easy to understand. For example, synchronization, mutual exclusion, cyclical operations, etc., are supported. Additionally, EPOS provides the option to automatically generate Ada code or Ada code fragments with the further ability to provide annotated Ada in support of the Ada development activity. Furthermore, high-level real time constructs can be transformed by EPOS automatically into lower level Ada code constructs.

5. Describe how your system supports concurrency, parallelism.

All real time aspects including tasking, concurrency, and parallelism are supported. If the target programming language for the code generator is capable of dealing with these requirements (as in the case of Ada), appropriate real time constructs are generated in the program (Ada) code.

6. Is your system constrained to a particular implementation language (Ada)?

No. The system is language-independent. For selected implementation languages (such as Ada, Pascal, Fortran, etc.) EPOS offers additional specific support. EPOS design support functions for all languages include assistance for design, development and assembly of other HOLS such as Jovial or assembly language.

7. Does your system produce Ada PDL?

No, not directly. EPOS, however, provides a design language to design systems and programs, which fulfills the requirements of PDL objectives. Additionally, there is the option to produce annotated Ada code directly. Code fragments may also be output to assist design and rapid prototyping.

8. Describe how your system supports life cycle Intraphase & Interphase communications.

Different representation for the different phases of the life cycle. Tracing, feedback, consistency checks and, to some extent, automatic transformation are provided between these representations. EPOS's unified database provides current status information to all participants throughout the project life cycle.

9. Is your system automated, executable, compilable?

The system is highly automated throughout its various phases of operation. It is also capable of providing (for certain HOLS) automated source code generation. It provides integrated support for software/hardware development, several transformations are automated. The system is executable.

10. Describe the graphics support for your system.

Utilizing its unified database, EPOS is capable of automatically generating graphs and text. Intensive graphical output (hierarchy diagrams, data structure diagrams, data flow and control flow diagrams, Petri-nets, hardware block diagrams, module connection diagrams, bar charts, PERT charts, Gantt charts, etc.). Graphic input of specifications is in development (prototype versions for hierarchy diagrams and flowcharts exist).

11. Describe how your system supports concepts of:

- Early prototyping vs. rapid prototyping
- Software reusability
- Information hiding
- Packaging concept
- Abstraction
- Typing
- Evolutionary development
- Generics
- Macros

- Data flows
- Control flows

Parts of the design can be prototypes using code generation capabilities, i.e., code fragments for Ada development.

The required object description and the concept of library modules, together with various powerful mechanisms for specifying attributes and searching, are the basis for software reusability, which can be addressed at various levels of abstraction.

The module concept allows the application of principles of information hiding. The Ada analysis package, in addition to its other functions, enforces compliance with visibility rules.

Modules/packages can be specified and analyzed with analysis programs which check import/export consistencies and visibility rules.

Various number of levels of abstraction can be defined according to user needs.

Predefined types (FLOAT, BOOLEAN, etc.) as well as user-defined types are possible. Analysis is provided.

Different builds are possible. Tracing of changes and extensions possible from the requirements specification to the design and code.

Generics are not directly supported in the current version, but similar concept is achievable.

Explicit specification of macros is possible. The semantics are known to the Ada generator.

Specification of data flow is supported as one of the design methodologies. Graphical documentation and analysis are provided.

Specification of control flow is supported, either as a dominant function in the function-oriented design methodology, or in addition to other specifications in other methodologies. Extensive graphical documentation and analysis are provided.

12. Is there a paradigm embedded in your system? If so, describe it briefly.

The paradigm (similar to Balzer [Balzer, Cheatham, Green: "Software Technology in the 1990's: Using a New Paradigm", IEEE **Computer**, November, 1983]) is to automate later phases (code generation) in the development life cycle and to emphasize earlier phases.

13. Describe the external tools with which your system interfaces (tool compatibility).

Very general interface to extract information from the data base and provide it for other tools. The machine editor is interfaced and used directly during development. Special interfacing to graphic editors is in development (prototype versions). General interfaces to plot tools.

14. Describe how your system supports hierarchical decomposition and flow direction (topdown, bottoms-up, both, etc.), architectural perspectives (designer creativity) and object-oriented design.

Directly supports hierarchical decomposition with appropriate syntax constructs and corresponding abstraction levels. Both top-down and bottom-up approaches are possible and supported. Object-oriented design is one of the principles of the system (language elements are design objects).

15. Is your system supported by formal syntax & semantics? Describe briefly.

Yes. System includes specification languages with formal syntax and semantics. Syntax checker (Parser) is provided. A set of analysis tools for evaluating the semantics and for transforming representations according to predefined semantics is also included.

16. Outline typical utilization costs for your system (cost of acquiring, using, training, & maintaining it).

Single VAX Installation	\$42,500
1 week of training	\$11,250
Annual maintenance/upgrade fee	<u>\$ 6,600</u>
Total	\$60,350

Price in effect as of February 1, 1987.

Specialized support is also offered.

17. Indicate the hostability (measure of degree of portability) of your system.

Very high degree of portability (one of the objectives of the system). EPOS is already available on the DEC VAX family (730-8600); IBM PC XT, AT and AT compatibles (Toshiba 3100); IBM mainframe (VM/CMS and MVS/TSO); Intel 8086/80286 (IRMX); Siemens 7000 (BS2000). Porting of the system of other machines is available on request. Motorola 68000 (UNIX), UNIX 4.2 and Bell 5 implementations will be available in March of 1987.

18. Describe how your system supports interdisciplinary abstractions/representations (i.e., systems engineering, software engineering, hardware engineering).

EPOS supports systems engineering, i.e., support of software development as well as hardware configuration. System and hardware development is supported in an environment that itself supports multiple methodological approaches and simultaneous mapping of software and hardware concepts.

19. How complete is the methodology - do its principles embody

- A development methodology only
- A design methodology only
- A programming methodology only
- A project control methodology
- A management methodology
- All of the above

All of the above - EPOS itself represents a full design and development philosophy with integrated methodology support. Additionally, EPOS is method-independent and supports various standard methodology approaches such as function-oriented, device-oriented, event-oriented, module-oriented, data structure-oriented, data flow oriented and method neutral, with the ability to automatically shift methodologies during use.

**20. Describe how your system supports a team development approach.
(Number of stations/users).**

Project planning of team staffing and work assignments, as well as responsibility analysis is provided. The work of the team members is performed in a so-called decentralized workstation concept with a central project database (on mainframe or PC) and several workstations for the developers.

Team members can work on different types of machines, with data exchange and integration possible.

21. Describe how your system supports design trade-offs.

Not directly supported, although analysis is provided to analyze different design approaches.

22. Indicate the range of problems to which the system can be applied.

EPOS has been successfully used in a large number of different projects, including real time projects and commercial applications. Project size has ranged from small, one/two-person projects to complex efforts involving several aerospace companies.

Target languages may range from assembler to Ada and other high-level languages.

The system is suited for all phases of development and a range of team sizes, with integrated support for development and management.

23. List the names, addresses, and phone numbers of five (customers) major users of your system.

The EPOS system is in use on major projects in approximately 250 installations world-wide. Users include:

West German Ministry of Defense, Grumman Aerospace, MBB (Messerschmitt-Boelkow-Blohm), Phillips, Robert Bosch, base 10, and Contraves, to name a few. Specific contact information is available on request.

EPOS REFERENCES

Mr. Eckhart Schwab
Ministry of Defense - Bonn
Rue VII
Emeleistr.
D-5000 Bonn WEST GERMANY
019/288 124 53

Dr. Kurt Keppner
Director, Technical Planning Div.
Contraves GmbH
Winterspurer Str. 17-19
D-7768 Stachach WEST GERMANY
077/77 8 13 57

Dr. Klenk
MBB (Messerschmitt)
Dept. LKE 111
D-9012 Ottobrun WEST GERMANY

Dr. Schmittke
Phillips Kassel
Dept. KEA
Postfach 310320
D-3500 Kassel WEST GERMANY

Dr. Mohr
IBM EE Laboratory
Schloßmühlchenstr. 220
D-70630 Boeblingen WEST GERMANY

8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

Distribution List for IDA Memorandum Report M-308

NAME AND ADDRESS	NUMBER OF COPIES
<u>Sponsor</u>	
Lt Col Jon Rindt SDIO/PI 1E149 Pentagon, Washington D.C. 20301-7100	3 copies
<u>Other</u>	
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2 copies
IIT Research Institute 4550 Forbes Blvd., Suite 300 Lanham, MD 20706	1 copy
<u>CSED Review Panel</u>	
Dr. Dan Alpert, Director Center for Advanced Study University of Illinois 912 W. Illinois Street Urbana, Illinois 61801	1 copy
Dr. Barry W. Boehm TRW Defense Systems Group MS 2-2304 One Space Park Redondo Beach, CA 90278	1 copy
Dr. Ruth Davis The Pymatuning Group, Inc. 2000 N. 15th Street, Suite 707 Arlington, VA 22201	1 copy
Dr. Larry E. Druffel Software Engineering Institute Shadyside Place 480 South Aiken Av. Pittsburgh, PA 15231	1 copy

Dr. C.E. Hutchinson, Dean
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755

1 copy

Mr. A.J. Jordano
Manager, Systems & Software
Engineering Headquarters
Federal Systems Division
6600 Rockledge Dr.
Bethesda, MD 20817

1 copy

Mr. Robert K. Lehto
Mainstay
302 Mill St.
Occoquan, VA 22125

1 copy

Mr. Oliver Selfridge
45 Percy Road
Lexington, MA 02173

1 copy

IDA

General W.Y. Smith, HQ	1 copy
Mr. Seymour Deitchman, HQ	1 copy
Mr. Philip Major, HQ	1 copy
Ms.Charlene Pandoli,HQ	1 copy
Dr. Jack Kramer, CSED	1 copy
Dr. Robert I. Winner, CSED	1 copy
Dr. John Salasin, CSED	1 copy
Dr. Cathy Jo Linn, CSED	3 copies
Ms.Julia Sensiba, CSED	2 copies
IDA Control & Distribution Vault	3 copies

END
DATED
FILM
8-88
Dtric