2

Special Report
CMU/SEI-92-SR-9

The Domain-Specific Software Architecture Program

LTC Erik Mettala
and Marc H. Graham, eds.

June 1992

4620f

Special Report
CMU/SEI-92-SR-9
June 1992

The Domain-Specific Software Architecture Program

LTC Erik Mettala
DARPA SISTO

Marc H. Graham
Technology Division, Special Projects

X

A-1

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

[illegible]

John S. Herman, Capt, USAF
SEI Joint Program Office

The Software Engineering Institute is sponsored by the U.S. Department of Defense.

This report was funded by the U.S. Department of Defense.

Table of Contents

List of Figures

The Domain-Specific Software Architecture Program

LTC Erik Mettala
DARPA SISTO
3701 N. Fairfax Drive
Arlington, VA 22203-1714

Marc H. Graham
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract: The DARPA Domain Specific Software Architecture Program (DSSA) is a five-year effort that has been active since July 1991. This document contains an overview of the work being done in the program as of July 1992.

Software architectures serve as frameworks for software reuse. Domain-specific software architectures also serve as a common language in which domain engineers can discuss, understand and teach the principles of their craft.

There are six independent projects within the DSSA program. Four of these projects are working in specific, militarily-significant domains. Those domains are Avionics Navigation, Guidance and Flight Director for Helicopters; Command and Control; Distributed Intelligent Control and Management for Vehicle Management; Intelligent Guidance, Navigation and Control for Missiles. In addition, there are two projects working on underlying support technology: Hybrid (discrete and continuous, non-linear) Control and Prototyping Technology.

This report contains brief descriptions from each project and an overview.

1 Introduction

System engineers design systems out of various materials: metals, hydraulics, electronics and software. When working with material other than software, they use tools and components specific to the task at hand. Those artifacts are the packaged expertise of a host of other engineering disciplines. The software in the system, on the other hand, is hand crafted by the software engineers, uniquely for the system being built, but out of the same low-level, general purpose building blocks for every system. The work being done in the DSSA program in domain-specific software development is meant to transform the relation of system and software engineers into one in which software engineers create building blocks and construction tools out of and with which system engineers create software subsystems.

In contrast to the DSSA program's vision of software development, Figure1-1illustrates the relation of system and software design today. The figure is drawn from the perspective of a controls engineer solving a problem using an iterative process of simulation and analysis. The

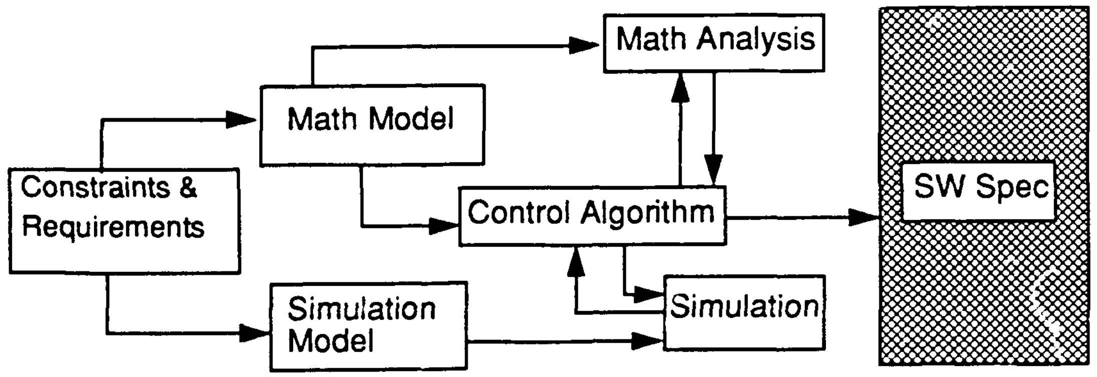


Figure 1-1 Hitting the Brick Wall!

solution takes the form of an "algorithm" that is well-defined in control theory and is known to be correct (within the accuracy of the analytic and simulation models). Before the algorithm can be implemented, it hits the "brick wall" called the Software Specification. Behind the wall, software engineers receive an unwieldy translation of the algorithm that has lost definition and quite possibly accuracy. The translation from the language of control theory to the language of software cannot add anything; it can only introduce errors. domain-specific software development tears down brick walls like these by applying the application engineer's problem-oriented language more directly to the task of software construction.

Figure 1-1 should not imply that the DSSA program is concerned only with control applications or only front-end issues such as algorithm design. The individual projects within DSSA cover a wide range of software development activities, from concept formulation and prototyping, software process design and measurement through software development, documentation and maintenance. No single project tackles all these problems, but the program as a whole is distinguished by its interest in the entire problem of software development.

2 Structure of the DSSA Program

The DSSA program is a five-year program that has been active since July 1991. There are six independent projects in the program. Each project is led by an industrial research lab, each has at least one academic partner, and each works with a military lab in the context of a specific research product. Four of the projects are applying themselves to specific military domains. They are:

- **Avionics Navigation, Guidance and Flight Director**
  Principal Contractor: IBM Federal Sector Division
  Principal Investigators: Lou Coglianese, Mark Goodwin, Will Tracz.
  Academic and Industrial Partners: Don Batory, University of Texas; Kirstie Bellman, Aerospace Corporation; David Gries, Cornell; David McAllester, MIT; Rick Selby, UC at Irvine; Dick Taylor, UC at Irvine
  Military Lab Partner: Wright Aeronautical Laboratories.

- **Command and Control**
  Principal Contractor: GTE Federal Systems
  Principal Investigators: Christine Braun; William Hatch; Theodore Ruegsegger
  Academic Partners: Bob Balzer, Martin Feather, Neil Goldman, Dave Wile, USC/Information Sciences Institute; Bob Might, George Mason University
  Military Lab Partner: US Army Communications and Electronics Command (CECOM).
- **Distributed Intelligent Control and Management (DICAM) for Vehicle Management**
  Principal Contractor: Teknowledge Federal Systems
  Principal Investigators: Frederick Hayes-Roth, Lee Erman, Allan Terry
  Academic Partners: Barbara Hayes-Roth, Gene Franklin, Stanford University
  Military Lab Partner: US Army Armament Research, Development and Engineering Center (ARDEC).
- **Intelligent Guidance, Navigation and Control**
  Principal Contractor: Honeywell Systems and Research Center
  Principal Investigators: Mike Jackson, Steve Vestal
  Academic Partner: Ashok Agrawal, U. of Maryland
  Military Lab Partner: Office of Naval Research (ONR).

Two of the projects investigate enabling technologies underlying domain-specific software construction:

- **Hybrid Control**
  Principal Contractor: ORA Corporation
  Principal Investigators: Richard Platek, James H. Taylor
  Academic Partners: Anil Nerode, John Guckenheimer, Cornell
  Military Lab Partner: ARDEC.
- **Prototyping Technology**
  Principal Contractor: TRW
  Principal Investigator: Frank Belz
  Academic Partner: David Luckham, Stanford
  Military Lab Partner: ONR.

2.1 Overview of Technical Issues Raised by DSSA

2.1.1 Software Architectures

Within the DSSA program there are three distinct approaches to architectures. The IBM and GTE projects are based on domain modelling approaches exemplified by the work of Prieto-Díaz and Cohen, and are collected in a recent IEEE tutorial [Prieto-Díaz 87], [Kang 90], [Prieto-Díaz 91]. This approach identifies the critical aspects (objects, operations and relationships) in a domain or class of problems as the experts in the domain perceive them. These aspects are represented in some way as a domain model.[1] The model is independent of any implementation. A software architecture is drawn from the model. Where the model describes

1. Domain modelling is a rich and evolving field to which the DSSA researchers are making varied contributions.The reader should not assume that there is or will be a single DSSA domain modeling technology.

a family of problems, the architecture describes a family of solutions. The architecture constrains possible solutions by setting, at various levels of abstraction and detail, a collection of components and component interfaces.

Software architectures serve as frameworks for software reuse. As explained below, domain-specific software development encompasses generative as well as reuse or compositional techniques. Domain-specific software architectures also serve as a common language in which domain engineers can discuss, understand and teach the principles of their craft. Participation of the George Mason Center for C^3I in the GTE project puts it in a favorable position for influencing the consensus building process. Partnership with military laboratories is a way in which each of the projects interacts with its user community.

The Teknowledge project is based in part on a particular software architecture, or "architectural style", associated with the work of NIST in robotic control [Albus 89]. That architecture proposes a multi-level hierarchy of controllers. The lowest level deals with control of individual servomechanisms; the highest level controls the interaction of groups of groups of controllers. Each controller has access to a "conceptually global" information base and world model. To this structure, Teknowledge adds a two-level architecture for each controller. A "domain controller" (DC) has responsibility for determining plans of action without regard for time constraints. A "meta-controller" directs the execution of the planned actions with the goal of maximizing the use of scarce resources, particularly time constraints.

The Honeywell and ORA projects' concept of a software architecture is quite different. Honeywell proposes multiple formal engineering models for the domain of guidance, navigation and control. The models include differential and difference equations for the description of control laws, scheduling theory and optimization for schedulability analysis and Markov processes for the determination of reliability. Software architectures are derived from these formal models. ORA is developing the mathematical field of hybrid (both discrete and continuous) control theory as the basis of a software architecture.

2.1.2 Domain-specific Software Development

Various mechanisms of domain-specific software development are under investigation within the projects. Compositional mechanisms facilitate reuse of existing artifacts, including software. Generative mechanisms are used when needed components are not available.[1] Constraint-based reasoning systems and module interconnection languages are critical underlying technologies for software composition. Prototyping technologies underlie generation. The TRW project serves as a technology conduit from the prototyping community into the DSSA program.

1. Programming in third generation programming languages (e.g., Ada) is a form of software generation. Domain-specific software development will not eliminate the need for such programming any more than third generation languages eliminated the need for assembler programming. The aim is to drastically reduce the amount of such programming.

Parameterized mechanisms are both compositional and generative. In using a parameterized software development mechanism, an engineer specifies the functionality, behavior and constraints on a system component by filling out a form. The recorded values drive the mechanism in the creation of the specified component out of pre-existing software and software templates. (See Batory's work in database management system generation [Batory 88].)

Parameter driven application generators are useful for domains whose variability is well understood in advance. For domains whose variability is broader or less well understood, other mechanisms are needed. These mechanisms may be domain independent or domain-specific.

The Honeywell and ORA projects are developing mechanisms specific to the domain of control systems. Currently ORA seeks to improve the state of the art in control law generation, stressing the development of non-linear control algorithms based on dynamical system simulation. Honeywell is designing a language for the specification and analysis of control algorithms. The software generated to implement those algorithms is analyzed for schedulability, bound to hardware platforms, and composed with portions of a collection of runtime support modules to produce a given application.

As an early test, the GTE project applied some of its ideas to message handling in command and control systems. The project created a domain-specific message handling language for describing the format, validation, and processing of C3 messages. A program generator translates this DSSA language into a domain independent specification language, AP5 [Cohen 87], which is then translated into the target programming language. Initial results show a 100-fold decrease in the amount of code needed to format, validate and process messages used in the Army Tactical Command and Control System (ATCCS).

2.1.3 Process Issues

Domain-specific software development requires new processes to regulate the engineering activities comprising system design. IBM and GTE are developing descriptions of processes based on domain modelling paradigms. Both projects include measurement based process improvement programs. They and others are investigating the use of process enactment languages. Teknowledge is developing a mixed-initiative process language, supporting interaction among the application developer, core development environment, and the tools loosely coupled to that environment.

Honeywell has proposed a process that organizes the efforts of control engineers, system engineers, reliability engineers, safety engineers, etc., as well as software engineers [Krause 91]. Control software development is an interdisciplinary activity that requires an integrated product development process, sometimes called "concurrent engineering".

2.1.4 Development Environments

Although tools and environments are an important product, the DSSA program is not an environment program. Thus, the program looks to import much of the needed technology from others. In particular, it looks to STARS for reusability library frameworks; to Arcadia for various features such as Chiron, Amadeus, APPL/A and possibly others; and to the PROTOTECH community for prototyping and module interconnection. The program has formed a working group on environment integration, infrastructure and tooling issues to more efficiently share knowledge and expertise.

3 Role of the SEI

The Software Engineering Institute plays an important supporting role in the DSSA program. The SEI is committed to being an integral component of DARPA and to accelerating the transition of technology from the research community into the DoD and the defense industry. The SEI has developed expertise in the nature of technology transition and is able to assist the DSSA projects and their military partners in planning for the transition of DSSA products into the laboratories and eventually into the field. Even though the DSSA program is not yet through the first year of its life, this planning has already begun.

Other opportunities for SEI support to DSSA are being explored. Some of these, at the time this is being written, are as follows:

- **Process Description.** As mentioned earlier, DSSA contractors have an interest in software process description. The SEI currently has a project that, in cooperation with the STARS program, is involved in describing software processes. STARS has a strong interest in the kinds of processes DSSA requires. Thus, the SEI can play a role in the cross-fertilization of two DARPA programs.
- **Software Development Environments.** The SEI's Environment Focus Area has expertise and interest in tool integration and environment frameworks. The DSSA program's work in environments represents an opportunity for mutual benefit. Some early meetings designed to realize this benefit will have been held by the time this is published.

References

[Albus 89] Albus, James; McCain, H.G.; Lumia, R. *NSA/NBS Standard Reference Model For Telerobot Control System Architecture (NASREM).* Technical Report 1235, National Bureau of Standards, 1989.

[Batory 88] Batory, Don S. *Building Blocks of Database Management Systems.* Technical Report TR-87-23, University of Texas at Austin, February 1988.

[Cohen 87] Cohen, Donald. "Automatic Compilation of Logical Specifications Into Efficient Programs". *Proceedings of the 5th National Conference on Artificial Intelligence.* ACM Press, August 1987: 20-25.

[Kang 90] Kang, K.C; Cohen, S.G; Jess, J.A; Novak, W.E; Peterson, A.S. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* (CMU/SEI-90-TR-21, ADA235785). Software Engineering Institute, November 1990.

[Krause 91] Krause, Jim; Vestal, Steve. "Thoughts on a DSSA-Based AGN&C Development Process". Personal correspondence, September 1991.

[Prieto-Díaz 87] Prieto-Díaz, Ruben. "Domain Analysis for Reusability". *Proceedings of COMPSAC 87.* ACM Press, 1987.

[Prieto-Díaz 91] Prieto-Díaz, Ruben; Arrango, Guillermo.*Domain Analysis and Software Systems Modelling.* IEEE Computer Society Press, 1991.

An Avionics Domain-Specific Software Architecture

Lou Coglianese, Mark Goodwin, Roy Smith, and Will Tracz
IBM Federal Systems Company
MD 0210
Owego, NY 13827
LOU@OWGVM3.VNET.IBM.COM
TRACZ@OWGVM0.VNET.IBM.COM

Don Batory
University of Texas at Austin
Computer Science Department
Taylor 2.124
Austin, TX 78712-1188
dsb@cs.utexas.edu

Kirstie Bellman
CSTS
The Aerospace Corp.
PO Box 92957
Los Angeles, CA 90009-2957
bellman@aerospace.aero.org

David Gries
Computer Science Department
Cornell University
Ithaca, NY 14853
gries@cs.cornell.edu

David McAllester
Department of Electrical Engineering and Computer Science
MIT
Cambridge, MA
DAM@WHEATIES.AI.MIT.EDUMIT

Rick Selby and Richard Taylor
Department of Computer Science
University of California at Irvice
Irvine, CA 92717
selby@ics.uci.edu
taylor@ics.uci.edu

Abstract: The DSSA-ADAGE (Domain-Specific Software Architecture-Avionics Domain Application Generation Environment) Project is part of DARPA's Domain-Specific Software Architecture Program. IBM, in cooperation with researchers at the Aerospace Corporation, MIT, Cornell, the University of Texas at Austin, and the University of California at Irvine, is striving to create a workstation-based environment to support the development, maintenance, and upgrade of avionics systems through the reuse of large portions of well-designed and well-documented software. This paper qualifies the scope of the ADAGE Project within the avionics domain and describes the project's research goals and approach. The paper concludes with a summary of progress made to date in defining an avionics architecture and domain model, an avionics domain-specific vocabulary, and domain-engineering and architecture-based development processes.[1]

1. This research was partially sponsored by DARPA and the Avionics Logistics Branch, Wright Laboratory, Wright-Patterson AFB, OH, under contract F33615-91-C-1788.

1 Introduction

DSSA-ADAGE is a joint industry/university research effort to apply leading edge basic research to the avionics domain. DSSA-ADAGE is an extension to the large-scale avionics reuse effort [Coglianese 87] (i.e., RASP (Reusable Avionics Software Project [Bunts 90]) and software composition [Tracz 91a] research at the Owego, NY Laboratory of IBM-Federal Sector Division (FSD), where software developers have created avionics software for over a dozen fixed-wing aircraft and rotorcraft).

The avionics application domain encompasses the use of electronics to provide operator support for performing an assigned task with an aircraft or space vehicle.[1] Typically, for each new avionics system, systems developers create or re-create the software that provides the operator interface and integrates the electronics rather than reuse existing software. The goal of the DSSA-ADAGE is to provide system develope s with the necessary environment to locate, adapt, compose, generate, integrate, and evaluate avionics applications within the subdomains of Navigation, Guidance, and Flight Director by analyzing a problem domain and creating/refining a set of standardized solutions within it.

1.1 The Avionics Problem Domain

An avionics system integrates the complex components of crew, airframe, powerplants, sensors, and specialized subsystems into an intelligent airborne system for achieving specific mission objectives within time and space constraints. Within the overall DoD mission, these objective can include delivery of ordnance on designated targets within a specified time "window", recording of electronic, geographic and other remotely detectable information over a selected area within a specified time frame, or transport of personnel and material between locations on a specified schedule. While executing these missions the airborne system may need to meet other constraints, such as minimizing use of human resources, avoidance of detection by surface/air/space based sensor systems, rendezvous and cooperation with other friendly mission elements, or flight into adverse weather, at night, without external navigation aids.

The core requirements of most advanced avionics systems include the capability to navigate, provide guidance, and provide flight direction (see Figure 1-1). Navigation is the ability to estimate the aircraft state relative to one or more reference frames. Guidance contains algorithms which analyze temporal and spatial mission objectives to produce desired lateral, vertical, and/or speed profiles based on aircraft performance characteristics and selected optimization criteria. Flight direction accepts the error signals and recommends control inputs to the crew (or aircraft automatic flight control system for coupled flight) which will null out the error signals thus locating the aircraft on the desired path.

1. Portions of this paper are based on "DSSA Case study: Navigation, Guidance, and Flight Director Design and Development", by Lou Coglianese, Roy Smith, and Will Tracz, which appeared in the proceedings of the 1992 IEEE Symposium on Computer-Aided Control System Design (CACSD'92), March 1992.

Ultimately the success of any integrated system hinges on the ability of the operators to access system capabilities and accurately interpret system data in a timely fashion [Smith 91]. Integrated avionics systems provide controls and displays giving the flight crew access to system data and situational data tailored to mission tasks.

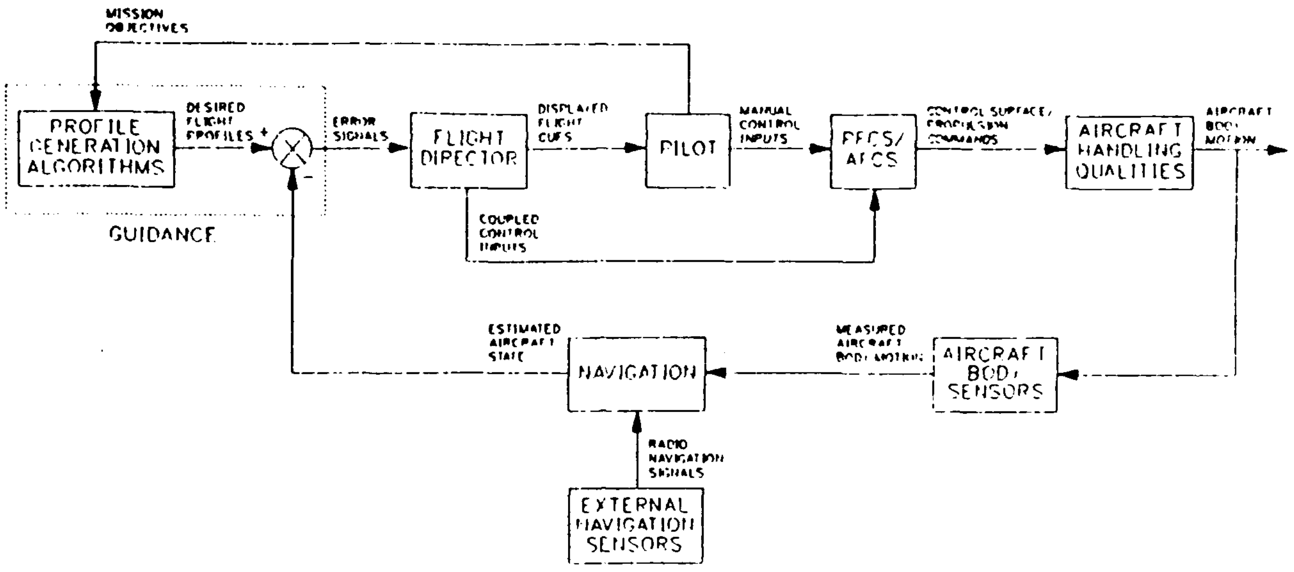


Figure 1-1 Typical Pilot In the Loop Navigation, Guidance, and Flight Director

1.2 DSSA-ADAGE Research Objectives

The DSSA-ADAGE project's research focuses on:

1. avionics requirements analysis and application synthesis through component-based, semi-automated composition,
2. architecture-based avionics system instantiation and verification through constraint-based reasoning, and
3. hypermedia-based environments integrated with domain-specific processes interpreted by process-centered, measurement-driven development tools for end-to-end lifecycle support.

1.3 DSSA-ADAGE Approach

The DSSA-ADAGE approach is based on the premise that although the probiems in Navigation, Guidance, and Flight Director are difficult, many of the solutions are well understood. For any new system, there will be several features that will require new and innovative techniques, but many more can be built by combining and adapting existing solutions. Therefore, analysts can identify constraints inherent in the avionics domain and define portions that can be brought under control. Concepts in the domain can be organized both from the perspective of the physical problems that they solve and from the way the components work together in a computer program to solve them. This organization of components, interfaces and behaviors is referred to as a Domain-Specific Software Architecture (DSSA). A DSSA not only provides a framework for reusable software components, but it also organizes design rationale and structures adaptability.

The DSSA-ADAGE team is building an architecture and a hypermedia-based environment that assists analysts and software developers in automating avionics development. The ADAGE environment, depicted in Figure 1-2, relies on:

- hypermedia representation of design record knowledge to facilitate software understanding,
- wrapping components with pedigree [Bellman 90] information to be used in analysis, modelling, development, and integration,
- constraint-based reasoning tools to reduce the user's adaptation and selection workload, software composition technology [Tracz 91b] to construct analyses, models, simulations and real-time software for embedded systems by combining components and user selections with implementation models, and
- a formal representation ([Osterweil 87], [Sutton 90]) of iterative spiral development process models and process measurement tools [Selby 91] to guide user actions.

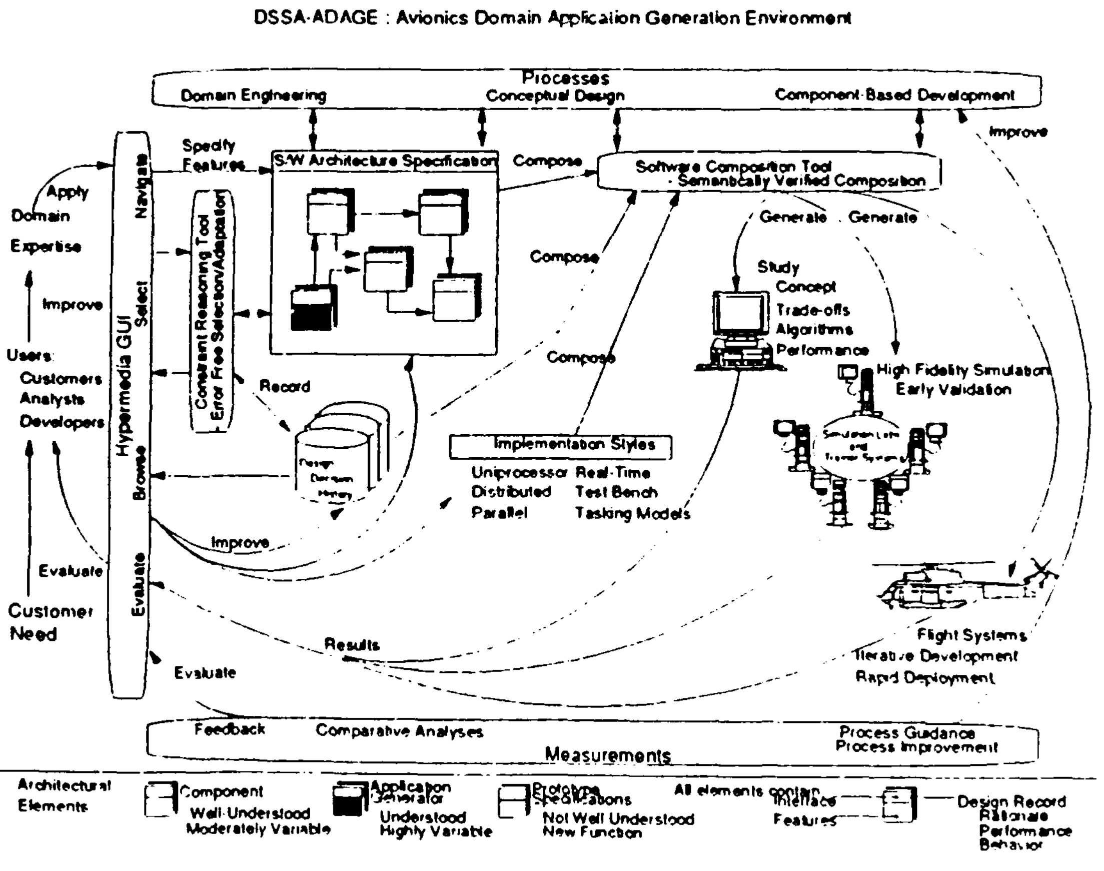


Figure 1-2 DSSA-ADAGE Approach

1.4 Progress to Date

The DSSA-ADAGE project came under contract in September of 1991. Most of the effort spent during the first quarter focused on gathering momentum. Team members working with domain experts defined and applied a domain engineering and component-based application generation process that afforded academic team members the opportunity to understand avionics domain terminology and concepts.

Lou Coglianese provided a preliminary avionics architecture definition of Navigation, Guidance and Flight Director at the first team meeting on September 26-27. Navigation analysis focused on models of data sources, the aircraft state vector, earth and atmospheric models, and relational coordinate models. The initial avionics domain knowledge engineering process results were sent out for review by team members and external sources. As a result, Sholom Cohen, at SEI, recognized the preliminary architecture to be similar to that used on the CAMP (Common Ada Missile Package) System (with the exclusion of a pilot in the loop).

Don Batory has created a preliminary version of domain model (i.e., a layered software architecture model for the avionics domain. The goal is to apply the GenVoca [Batory 91] design/-domain-modeling concepts to avionics software. Work on the ADAGE environment work started prior to coming under contract through the efforts of Ken Anderson, a University of California at Irvine graduate student. Ken successfully ported a portion of the Arcadia [Taylor 88] and ANNA tool suites to an IBM RS/6000.

Dick Taylor has evaluated current multi-media system capabilities and has defined the requirements for hypermedia extensions for Chiron [Keller 91].

The preliminary draft of the ADAGE Domain Engineering Process Description has been completed and is serving as a strawman for further definition and discussion. An outline of the ADAGE Component-Based Application Development Process Description (previously referred to as the ADAGE Megaprogramming Process Description) is also being reviewed. Finally, a draft version of the Avionics Domain Dictionary is being circulated for review among team members.

References

[Batory 91] Batory, D.S. and O'Malley, S.W., *The Design and Implementation of Hierarchical Software Systems Using Reusable Components, Austin, TX,: University of Texas*, TR-91-22, 1991.

[Bellman 90] Bellman, Kirstie L. and A. Gillam, "Achieving Openness and Flexibility in Vehicles", *Proceedings of the SCS Eastern Multiconference*, vol. 22, no. 3, pp. 255-260, April 1990.

[Bunts 90] Bunts, A. and Gundrum, V., *Lessons Learned from the Reusable Avionics Software Project (RASP)*, 1990. Internal Use Only.

[Coglianese 87] Coglianese, L., *Using Ada to Develop Reusable Ada Avionics Packages Presentation foils*, 1987. IBM Internal Use Only.

[Keller 91] Keller, R., Cameron, M., Taylor, R.N., and Troup, D.B., "User Interface Development and Software Environments: The Chiron-1 System", *Proceedings of ICSE 13*, pp. 208-218, May 1991.

[Smith 91] Smith R. S. and Murchie G. S., "The Army Special Operations Aircraft Integrated Avionics Subsystem - An Operational Perspective", *Proceedings of the Digital Avionics Systems Conference*, October 1991.

[Osterweil 87] Osterweil, L. J., "Software Processes are Software Too", *Proceedings of the Ninth International Conference on Software Engineering*, pp. 2-13, March 1987.

[Selby 91] Selby, R., "Metric Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development", *Proceedings of ICSE 13*, May 1991.

[Sutton 90] Sutton, S.M., Heimbegner, D., and Osterweil, L.J., "Language Constructs for Managing Change in Process-Centered Environments", *Proceedings of Fourth Symposium on Software Development Environments*, pp. 206-217, December 3-5 1990.

[Taylor 88] Taylor, R., et al., "Foundations for the Arcadia Environment Architecture", *Proceedings of Third Symposium on Software Development Environments*, pp. 1-13, November 1988.

[Tracz 91a] Tracz, W.J., "A Conceptual Model for Megaprogramming", *ACM Software Engineering Notices*, vol. 16, no. 3, pp. 36-45, July 1991.

[Tracz 91b] Tracz, W.J., *Formal Specification of Program Schemata*, Ph.D. thesis, Stanford University, 1991. In progress.

Domain-Specific Software Architectures: Command and Control

Christine Braun, William Hatch, and Theodore Ruegsegger
GTE Federal Systems
15000 Conference Center Dr.
Chantilly, VA 22021
braun@europa.asd.contel.com
hatch@europa.asd.contel.com
theodore@europa.asd.contel.com

Bob Balzer, Martin Feather, Neil Goldman, and Dave Wile
USC/Information Sciences Institute
4676 Amiralty Way
Marina Del Rey, CA 90292
balzer@isi.edu
feather@isi.edu
goldman@isi.edu
wile@isi.edu

Abstract: GTE is the Command and Control contractor for the Domain-Specific Software Architectures program. The objective of this program is to develop and demonstrate an architecture-driven, component-based capability for the automated generation of command and control (C2) applications. Such a capability will significantly reduce the cost of C2 application development and will lead to improved system quality and reliability through the use of proven architectures and components.

A major focus of GTE's approach is the automated generation of application components in particular subdomains. Our initial work in this area has concentrated in the message handling subdomain; we have defined and prototyped an approach that can automate one of the most software-intensive parts of C2 systems development.

This paper provides an overview of the GTE team's DSSA approach and then presents our work on automated support for message processing.[1]

1 The DSSA Concept

DSSA is based on the concept of an accepted generic software architecture for the target domain. As defined by DSSA, a software architecture describes the topology of software components, specifies the component interfaces, and identifies computational models associated with those components. The architecture must apply to a wide range of systems in the chosen

1. ©1992 IEEE. Reprinted, with permission, from *Proceedings of the 1992 IEEE Symposium on Computer Aided Control System Design*; Napa, California, March 17-19, 1992; pp. 129-136. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the IEEE copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Institute of Electrical and Electronics Engineers. To copy otherwise, or to republish, requires a fee and specific permission.

domain; thus it must be general and flexible. It must be established with the consensus of practitioners in the domain.

Once an architecture is established, components that conform to the architecture, i.e., that implement elements of its functionality in conformance with its interfaces, will be acquired. They may be acquired by identifying and modifying (if required) existing components or by specifically creating them. One of the ways they may be created is through automated component generation. DARPA has sponsored work in this area at USC Information Sciences Institute such as the AP5 application generator project, and is interested in incorporating this or related technology.

The existence of a domain-specific architecture and conformant component base will dictate a significantly different approach to software application development. The developer will not wait until detailed design or implementation to search for reuse opportunities; instead, he/she will be driven by the architecture throughout. The architecture and component base will help define requirements and allow construction of rapid prototypes. Design will use the architecture as a starting point. Design and development tools will be automated to "walk through" the architecture and assist the developer in the selection of appropriate components. The ultimate goal is to significantly automate the generation of applications. A major DSSA task is to define such a software lifecycle model and to prototype a supporting toolset.

These activities will be accompanied by extensive interaction with the development community for the target domain, and by technology transition activities. One aspect of this is that each domain team is working closely with a DoD agency that carries out major developments in the designated area. The GTE team is working with the US Army Communications and Electronics Command.

2 Why Command and Control?

There are many reasons why the command and control domain is an excellent target for DSSA technology. It is a high payoff area; command and control systems are needed even in the current military climate. (This is particularly true when one recognizes that applications such as drug interdiction fall within the C2 "umbrella".) It is a well-understood area; most of the processing performed in C2 applications is not algorithmically complex. However, C2 applications are very large, and much of this size comes from repeated similar processing, such as parsing hundreds of types of messages. In addition to this commonality within applications, there is much commonality across applications. Multiple C2 systems must handle the same message types, display the same kinds of world maps, etc.

The kinds of commonality in C2 applications are very well-suited to DSSA techniques. In some areas, components can be reused identically; these can be placed in the DSSA component base and highly optimized. In other areas, components will be very similar in nature but differ in the particulars, e.g., message parsing. These areas are a natural fit to the DSSA component generation technology, allowing a table-driven generator to quickly create the needed specific component instances.

3 GTE's Approach

Figure 3-1 illustrates GTE's overall approach to the DSSA program.

Initially, project work will follow two parallel threads. The first will define a software process model appropriate to architecture-driven software development and will develop a toolset to support that process. The second will establish a capability that implements the process for the command and control domain, based on a C2 architecture and a set of reusable C2 components.

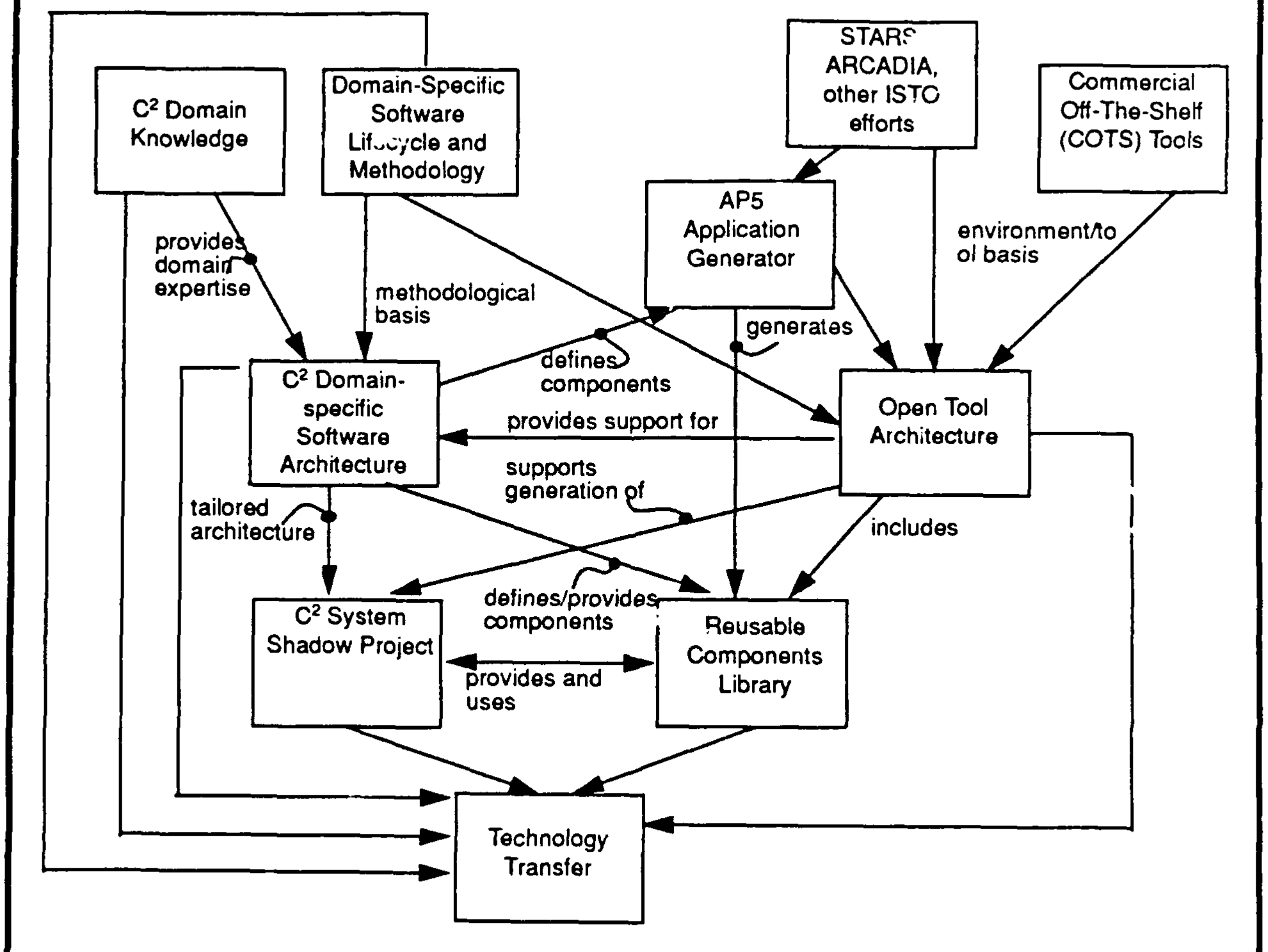

C² Domain Knowledge
Domain-Specific Software Lifecycle and Methodology
STARS ARCADIA, other ISTO efforts
Commercial Off-The-Shelf (COTS) Tools
AP5 Application Generator
provides domain expertise
methodological basis
environment/tool basis
generates
defines components
C² Domain-specific Software Architecture
Open Tool Architecture
provides support for
supports generation of
tailored architecture
includes
C² System Shadow Project
defines/provides components
Reusable Components Library
provides and uses
Technology Transfer

Figure 3-1 GTE's DSSA Approach

The DSSA process model will address all aspects of the software life cycle. It will describe activities for establishing system requirements, developing the software system, and sustaining the system after delivery. The DSSA toolset will support all of these activities, automating them as far as possible. In particular, it will automate system development activities by using the architecture as a template, guiding the selection of available reusable components, and automating the generation of specific required components. The toolset will be constructed insofar as possible from available tools, both commercial products and products of the research community. In particular, it will make use of USC/ISI's AP5 application generator, DARPA/ STARS reuse libraries, and DARPA/Prototech tools. Open tool interfaces will be emphasized to minimize specific tool dependencies, thus making the toolset usable in the widest range of environments.

Fundamental to the C2 DSSA capability is the development of a C2 software architecture. This starts with development of a multi-viewpoint domain model, created through interaction with all elements of the DoD C2 community. The automated Requirements Driven Development (RDD) methodology will be used in model creation. From this, an object-oriented software architecture will be developed. The architecture will tie back to the multi-viewpoint model so that mappings to different views of the domain functional decomposition are apparent. George Mason University's Center for C3I will play a major part in this modeling and consensus-building activity. A base of components conforming to the architecture will then be developed. Many of these will be existing components, perhaps modified to fit the architecture. Others will be automatically generated using AP5.

The DSSA capability will be demonstrated by development of a prototype C2 system, most likely an element of the Army Tactical Command and Control System (ATCCS). An independent metrics/validation task will assess the effectiveness of the approach and gather metrics. The methodology and toolset will be revised based on findings and further necessary research will be identified.

Throughout the program, a technology transfer task will present results in conferences, papers, seminars, and short courses. The George Mason University Center for C3I will serve as a focal point for technology transfer.

4 Application Generation

4.1 The Technology

Application generators are tools that permit software developers to create software application programs in a much higher-level language tailored to the application domain. These programs are automatically translated by the application generator to a lower-level language, thus "generating applications". This greatly reduces the effort required to create working applications, typically by at least an order of magnitude. The benefits are analogous to those achieved by moving from assembly language development to use of standard procedural languages such as FORTRAN, C, and Ada.

Fourth Generation Languages (4GLs) are application generators for DBMS-oriented information system applications. Because 4GLs focus on a narrow class of applications, they can include very powerful constructs that allow software to be developed quickly and easily by those familiar with the application domain. Management Information System (MIS) developers using 4GLs achieve productivity improvements of as much as 50-100 times over traditional (usually COBOL) language users.

Application generators can be (and have been) developed for other types of applications as well. They are best suited to narrow domains, or subdomains of large domains such as C2. Because they require a domain-specific vocabulary for expressing applications, they are generally unique to the domain or subdomain and not easily modified to handle other domains. Creation of an application generator for a particular domain, furthermore, is a significant undertaking. Development of an application generator is most appropriate in domains that are well-understood and in which many different developments perform primarily the same kinds of processing.

4.2 The AP5 Approach

USC Information Sciences Institute (ISI) has developed a capability (called AP5) that supports the development of application generators. AP5 is based on the concept of *relational abstraction.* The application developer identifies abstract data objects and the logical relationship among them. Effectively, the developer has access to a "virtual database" expressed succinctly in terms of the known structure of the domain's data model. Application behavior is then expressed in terms of these data objects, accessing them associatively via queries and modifying them based on values of other objects. This allows the user to concentrate on behavior rather than representation, and provides the power to express that behavior at a very high level.

Providing an AP5 application generator for a particular subdomain requires the development of a domain-specific language for that domain. This is a relatively straightforward task because the language, regardless of domain, involves the same fairly simple set of relation-oriented constructs for expressing data relationships, validations, and actions. It is also a critical task, because the expressive capability of this language is what provides the application generator's power. A translator is then developed to map the language to an underlying program generator, which produces executable procedural code. This is also not too complex, as all languages contain similar constructs. Most of the work is done by the underlying generator. (Currently the system generates LISP; an Ada generator is in development.)

A drawback to many existing application generators is poor efficiency of the generated code. This has, in many cases, made these generators suitable only for developing prototypes. AP5 addresses this problem by allowing the user to specify *annotations* that provide guidance to the translator on desired implementations of specific operations. These annotations can be added incrementally while tuning to achieve desired performance.

AP5 can play a key role in the C2 DSSA program. We anticipate that a number of C2 subdomains will be amenable to this approach. By developing generators for those subdomains we can achieve two major advances in productivity:

- DSSA users can use the generators to create specific components in the subdomain with far less effort.
- DSSA architects can use the generators to create reusable subsystems that can then form part of the component base available to DSSA users.

We have already identified the message handling subdomain as a candidate for AP5 technology; a tentative choice for the next area to tackle is fusion processing.

Figure 4-1 shows the activity flow that will be followed: identifying classes of components (subdomains) to be addressed, based on the architecture; defining domain-specific languages and producing generators; developing annotations to permit optimization; and generating reusable application components.

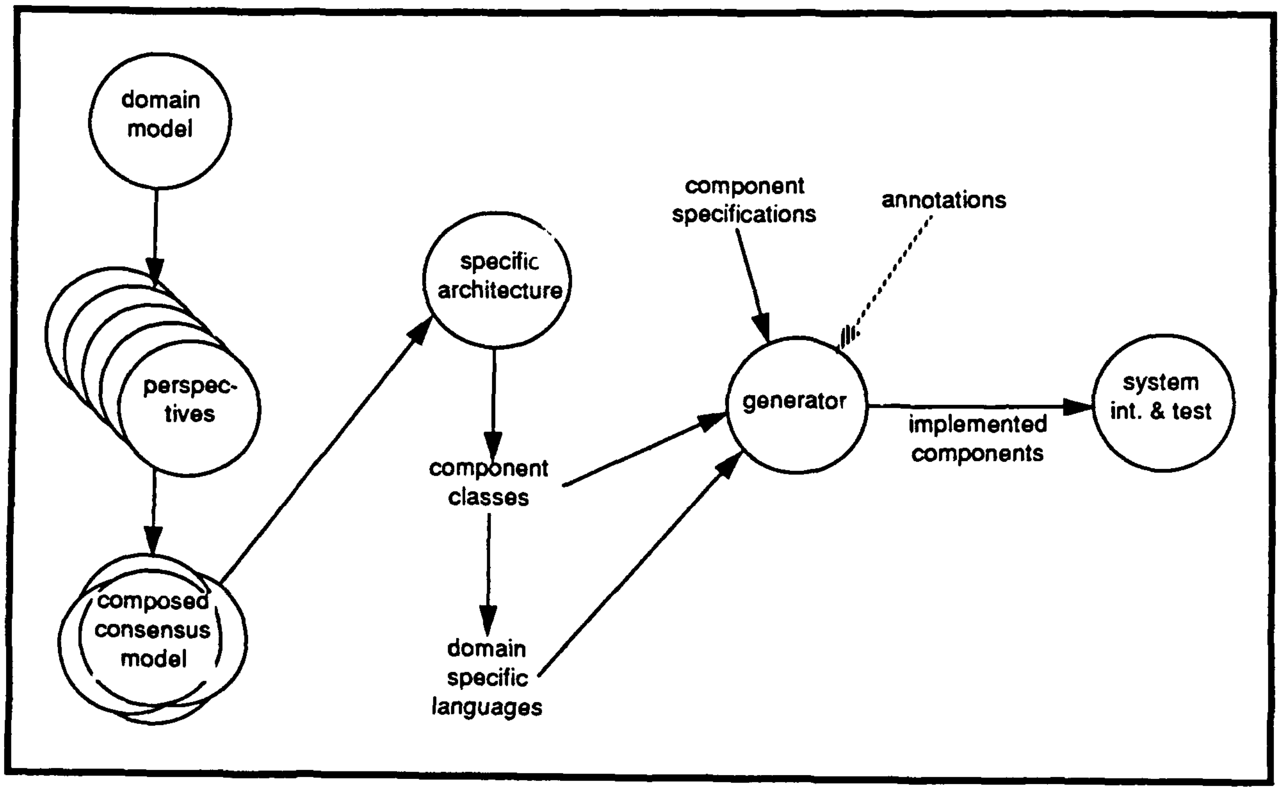


Figure 4-1 DSSA Application Generation Activity Flow

5 C2 Message Handling

As indicated in Figure 5-1, the message handling subsystem is one of the key interfaces between a C2 system and the "outside world". It provides a means of communicating information between different C2 systems and to/from other C2 resources (such as vehicles and weapon installations). Messages may be text or bit streams; we will deal here with text messages. Some text messages are free-form, but most today follow standard prescribed formats; we will deal with formatted messages.

C2 messages are created by humans (on the transmitting side of the interface) according to a written description of the formats. The receiving side parses the message (according to an encoded understanding of the standard format), validates it for correctness, and places the received information in the database for use by other parts of the system (for example, decision support).

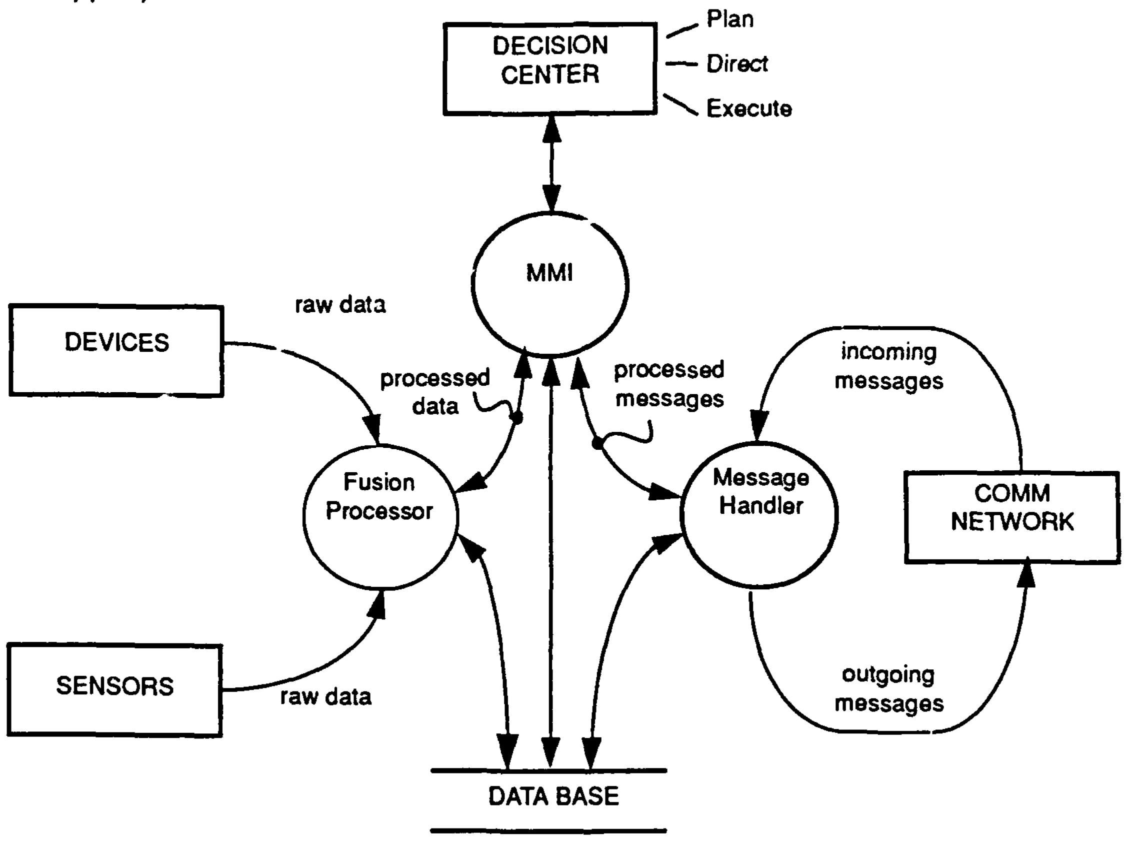


Figure 5-1 C2 System Operations

There are several standard families of messages, such as NATO and JINTACCS messages. Each of these can include several hundred message types; for example, there are approximately 300 NATO message types. (Many types of messages are shared by several message families.) Message formats are described in massive documents using *ad hoc*, non-standard description methods. Typically the descriptions involve much prose. For example, Figure 5-2 shows the description for a single line in one type of message. Furthermore, it is not a complete description; many field descriptions cross-reference to other descriptions.

A message consists of a number of such lines (called *datasets*— may be more than one physical line) grouped together in an *envelope* (which contains from/to information, classification level, etc.). While each type of message can contain only certain kinds of datasets, many are

Data Set ID: MSGID

Fld	Element Descriptive Name	Descript.	M	Edit Rule	Remarks
1	Message Code Name	25 AN	x	1. Must be a member of the approved set of message code words.	
2.	Originator	25 AN	x	1. Must be a plain language address or approved short title	a. Plain language addresses are validated against values found in the references
3.	Message Serial Number	3 N	a	1. Positive integer between the values 001 to 999.	a. May be required for specific messages.
				2. Out of sequence may indicate missing message. See rules for specific msg. code word.	b. Sequence is restarted on 1 Jan each year. May be rolled over when upper limit is reached. c. For Command authorities serial may be validated to maintain order when processing reports.
4.	As-of-Month	3 AN	a	1. Standard abbreviation for month message sent.	a. Required if serial number is used and as-of-DTG not present. b. Not allowed if as-of-DTG not present.
5.	As-of-Year	4 N	o	1. May not be a future year.	a. As-of-Month must be present.

Figure 5-2 Example Message Line Description

optional and their order is generally not prescribed (though there are exceptions). Validity of datasets can depend on other datasets in the message. Each dataset contains a prescribed sequence of *fields*, separated by slashes, with a required order and a well- defined format. Field validity can depend on values in other fields of that dataset as well as in other datasets in the message. Figure 5-3 is an example message (excluding the envelope).

The code involved in writing the software to implement message handling is extensive and error prone. Working from the prose specification, programmers write code to extract each field from each dataset, validate it according to the specified rules, translate it to the appropriate internal representation, build database update transactions, and write to the database. Typically, a single message type can take from 5000 - 100,000 lines of HOL code. The Navy WWMCCS system uses approximately 4 million lines of code to implement 30 message types. Clearly this is a part of C2 system development that should be considered for automation.

6 Automating C2 Message Handling Using AP5

To automate C2 message handling using AP5, we have developed a language specific to the message handling subdomain that provides constructs for specifying message formats, for indicating required validations, and for describing desired database updates.

```
NATOUNCLASSIFIED
SIC: NSR
EXER /OPEN GATE 91//
MSGID /NAVSITREP/CINCIBERLANT/135/DEC/91//
PART /I/HOSTILE//
FORCE /OR523/3/37000N0-012000W3/145/17K/H//
SHIP /OR523A/KARA/-/CG/-/UR//
SHIP /OR523B/KRESTA//
SHIP /OR523C/KRESTA//
SUBTK /OR734/33000N6-010000W1/095/9K/M//
SUB /OR734/TANGO//
PART /II/UNKNOWN/NC//
PART /III/FRIENDLY//
FORCE /CTU 405.1.2/5/420015N2-1333440W8/175/20K//
FORCE /CTU 387.3.2/2/36010N0-004380W5/090/5K//
AMPN /MINE SWEEPING GROUP...//
AIRTK /934/33000N6-010000W1//
AMPN /ONE P-3 SEARCHIN BOX...//
```

Figure 5-3 Example Formatted Message

6.1 Specifying Message Formats

Message formats are described in a simple set language that indicates which datasets are allowed and which are optional for a particular message type. For example,

```
type SPOT =(FORCE), (SHIPTK|AIRTK|AIRCRAFT),
            SHIP
```

would indicate that a SPOT message consists of an optional FORCE dataset, an optional occurrence of one of the SHIPTK, AIRTK, or AIRCRAFT datasets, and a required SHIP dataset.

Message format descriptions can be accompanied by *validations* that indicate which combinations of datasets are valid. For example,

```
type SPOT = (FORCE), (SHIPTK|AIRTK|AIRCRAFT),
            SHIP
   validations
      disallow MSGID.message-serial-number;
      require SHIP.location
      no SHIPTK and no AIRTK requires FORCE;
```

indicates that the message-serial-number field of the MSGID dataset must not be present, the location field of the SHIP dataset must be present, and, if no SHIPTK dataset and no AIRTK dataset is present, the FORCE dataset must be present.

6.2 Specifying Datasets

Dataset formats are described in terms of the fields that make up the dataset and the format of each of those fields. Fields are ordered, so each dataset is characterized by a sequence of fields. Optional fields are indicated by parenthesizing them. Mutually exclusive fields are indicated by alternative bars. As for message formats, dataset descriptions can include validations. For example, a dataset description of a MSGID dataset might be:

```
dataset MSGID = message-code-name (originator)
            (message-serial-number) (as-of-month)
            (as-of-year) (as-of-DTG)
   validations
      as-of-DTG precludes as-of-month;
      as-of-DTG precludes as-of-year;
      as-of-year requires as-of-month;
      message-code-name /= SPOT requires originator;
      message-serial-number and no as-of-DTG
            requires as-of-month;
   field message-code-name = A*26;
   field originator = A*25;
   field message-serial-number = N 3;
   field as-of-month - month;
   field as-of-year = N 4;
   -- as-of-DTG in form: DDHHMMZS        MMMYY
   field as-of-DGT - day, hour, minute, (Z), SUM1, month,
            year;
   field SUM1 = N 1;
   field day = N2;
   field hour = N 2;
   field minute = N 2;
   field month = A 3;
   field year = N 2;
```

6.3 Specifying Database Transactions

The C2 message description language also includes a means for describing the transactions to be carried out for each received message. An example of a segment of such a specification is:

```
{insert msg_Orig_S (ORIGINATOR = PROSIGN.FN,
   MSG_TYPE = MSGID.Code,
   MSG_DTG = sortable_date (ENVELOPE.DTG),
   CLASSIFY = classification_code(ENVELOPE.Sec));
   . . .
```

The database update language also includes tests of field values, so that updates can be conditional on those values, and a capability to allow a sequence of updates to be named and reused in other update instructions. This simple language provides all the power needed to describe the database transactions resulting from received messages.

7 Implications

Clearly, automated generation of message handling software can save greatly on the labor involved in creating such software. A message handling subsystem that requires 4 million lines of HOL code should require less than 1% of that in the message description language.

Perhaps more significantly, there will be little reason to write most of the code more than once. The code required to parse and validate a message of a particular type is not specific to the system being implemented. Once the message specification is developed in the message description language, it can be reused. Minor changes in the specification of required database updates can be easily implemented for individual systems.

An even more far-reaching impact of this work is the development of a precise, unambiguous way of describing message formats. Rather than the *ad hoc* prose descriptions now used in describing message formats, the message description language can be used directly. This will eliminate errors in understanding and correctly implementing message descriptions.

This precise message description mechanism, along with the built-in incentive to reuse message description implementations, will contribute substantially to the development of more error-free message handling subsystems. A major aspect of this benefit is improved interoperability, as systems will no longer be dependent on the programmers' understanding of message formats. All implementations will share a common understanding and be able to interoperate with the full power and precision envisioned for formatted messages.

8 Acknowledgments

The work described in this paper has been supported by the Defense Advance Research Projects Agency through U.S. Army Communications-Electronics Command Contract No. DAAB07-92-C-Q502 and through NASA Ames Research Center Contract No. NCC 2-520.

References

[Balzer 92] Balzer, Bob and Martin Feather, Neil Goldman, Dave Wile, "Proposal for DS Languages for C3 Messages", USC/ISI working paper, 1992.

[Braun 90] Braun, Christine L. and William L. Hatch, "Software Reuse Through CCIS Architecture Standardization", *Proceedings of the 11th AFCEA Europe Symposium and Exposition*, October 1990.

[Hatch 92] Hatch, William, "Example Message Descriptions and Database Transactions", GTE working paper, 1992.

[Ruegsegger 92] Ruegsegger, Theodore, "Domain Specific Software Architectures — Command and Control", briefing slides, CECOM Real-Time/Reuse Technical Interchange Meeting, Ft. Monmouth, NJ, February 1992.

[Wile 90]	Wile, David S., "Adding Relational Abstractions to Programming Languages", *Proceedings of workshop on Formal Methods in Software Engineering*, Napa Valley, CA, May 1990.
[Balzer 85]	Balzer, Robert, "A 15 Year Perspective On Automatic Programming", *IEEE Transactions on Software Engineering*, Nov. 1985
[Cohen 89]	Cohen, Donald, "Compiling Complex Database Triggers", *Proceedings of 1989 ACM SIGMOD* (1989), ACM
[Goldman 92]	Goldman, Neil and K. Narayanaswamy, "Software Evolution through Iterative Prototyping", to appear in the *Proceedings of the 14th ICSE Conference*, IEEE, Melbourne Australia 1992.

Domain-Specific Software Architectures: Distributed Intelligent Control and Communication

Frederick Hayes-Roth, Lee D. Erman, and Allan Terry
Teknowledge Federal Systems, Inc.
Cimflex Teknowledge Corp.
1810 Embaracadero Rd.
Palo Alto, CA 94303
rhayes-roth@teknowledge.com
lerman@teknowledge.com
aterry@teknowledge.com

Barbara Hayes-Roth
Knowledge Systems Laboratory
Department of Computer Science
Stanford University
Stanford, CA 94305
bhr@sumex.stanford.edu

Abstract: We are developing a generic control architecture suitable for use as a single intelligent agent or as multiple cooperating agents. The generic architecture combines a task-oriented *domain controller* with a *meta-controller* that schedules activities within the domain controller. The domain controller provides functions for model-based situation assessment and planning, and inter-controller communication. Typically, these functions are performed by modules taken from a repository of reusable software. In tasks that are simple, deterministic or time-stressed, the modules may be compiled into or replaced by conventional control algorithms. In complex, distributed, cooperative, non-deterministic or unstressed situations, these modules will usually exploit knowledge-based reasoning and deliberative control.

To improve the controller development process, we are combining many of the best ideas from software engineering and knowledge engineering in a software environment. This environment includes a blackboard-like *development workspace* to represent both the software under development and the software development process itself. In this workspace, controllers are realized by mapping requirements into specializations of the generic controller architecture. The workspace also provides mechanisms for triggering applications of software tools, including knowledge-based software design assistants.

This paper explains our general approach, illustrating it in the context of our current demonstration task.[1]

1. The work described in this paper was partly supported by DARPA and the US Army ARDEC, Picatinny Arsenal, under contract DAAA21-92-C-0128.
©1992 IEEE. Reprinted, with permission, from *Proceedings of the 1992 IEEE Symposium on Computer Aided Control System Design*; Napa, California, March 17-19, 1992; pp. 129-136. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the IEEE copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Institute of Electrical and Electronics Engineers. To copy otherwise, or to republish, requires a fee and specific permission.

1 Introduction

We have recently begun a four-year effort to develop a new technology foundation and associated methodology for the rapid development of high-performance intelligent controllers. These controllers will be employed in distributed intelligent control and management (DICAM) applications. Examples of such applications include intelligent highway systems, military command and control systems, and factory floor control systems. Our near-term domain of application is vehicle management systems, where one or more controllers may be employed to control a single vehicle, and these composite controller/vehicles are further aggregated and organized into higher-levels of control and capability. In a military context, for example, a single controller may be used for each subsystem within a tank, each tank system may be controlled by collectively organizing its subsystems, the overall tank may be controlled by another controller that coordinates the tank system controllers, several tanks may combine to form a platoon with its own control level, one or more platoons may form a battalion, and so on.

Our research project is one of several sponsored by DARPA (Advanced Research Projects Agency of the US Defense Department) and the US Army to advance the technology for domain-specific software architecture (DSSA). An overview of our DICAM-DSSA research focus is depicted in Figure 1-1. The actual vehicle management task concerns a howitzer, a tank-like vehicle that aims at more distant targets. The project has four principal focus elements. First, we are formulating a *reference architecture* for intelligent control. Second, we are supporting the construction of applications in a *development workspace* in which system

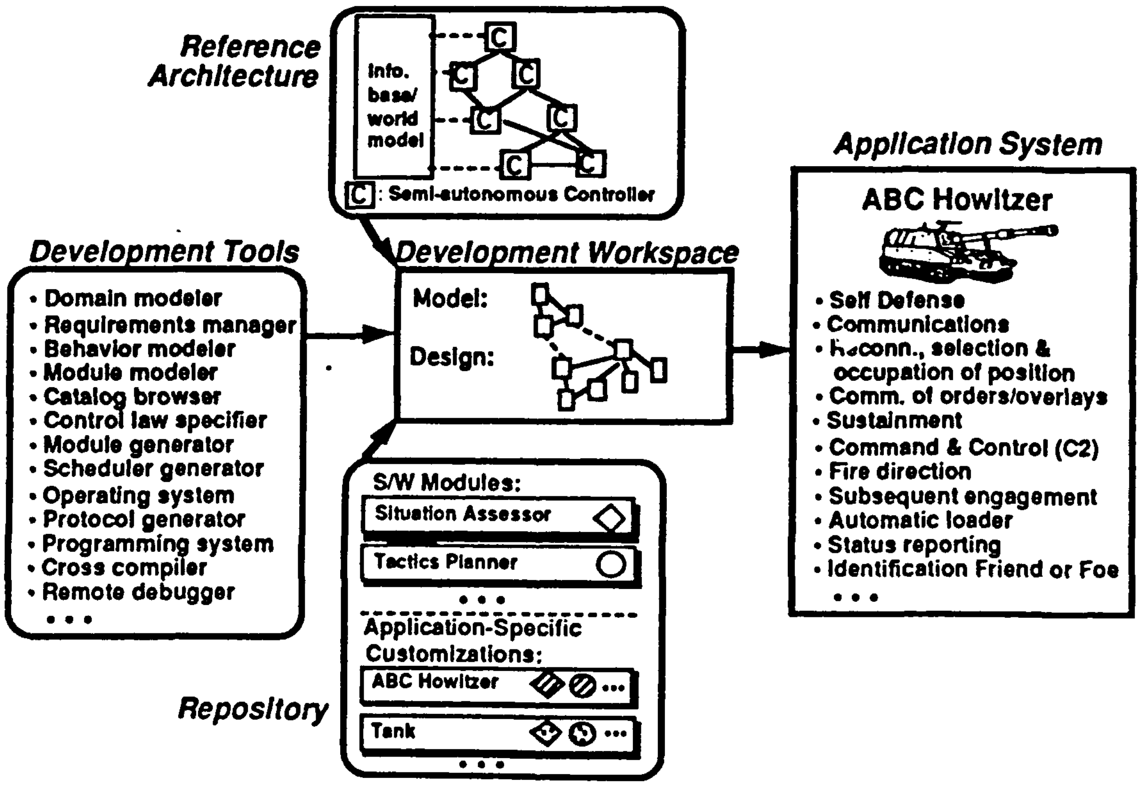


Figure 1-1 Focus Elements of Our DICAM-DSSA Research

requirements are ultimately satisfied by choosing design components that specialize and particularize components of the generic reference architecture. Many of the specialized modules and particular data used to instantiate a design are taken from a *repository*. The entire development process is supported by a rich array of *development tools*, which incorporate numerous techniques from both software engineering (e.g., control law specifiers, code generators, protocols, compilers, and debuggers) and knowledge engineering (e.g., domain modeler, requirements manager, and various knowledge-based design assistants). In short, our DICAM-DSSA fuses knowledge engineering (KE) and software engineering (SE) approaches both within the intelligent real-time control software being developed and in the software development process itself.

Our project integrates KE and SE in three principal ways. First, in order to develop intelligent controllers we need to develop a hybrid control technology that combines key concepts from real-time software engineering with the knowledge-based deliberative reasoning concepts of knowledge engineering. Second, we need to apply concepts, methods and tools from KE directly in the software development process. This paper highlights six principal KE elements involved in this process: knowledge-based models for domain analysis; classification for taxonomies, abstraction and specialization; blackboard methods of incremental problem-solving for system design and development; constraint specification and processing in software requirements management; and knowledge-based expert systems for providing software development and design assistance. The third category of relationships covers applications of SE methods to KE. In particular, we believe that there are many conventional and important SE concepts, methods and tools that need to be applied to the development of knowledge-based intelligent systems. This paper describes seven which are central to our project: real-time systems, database-centered design, hierarchical systems, distributed systems, reference architectures, repositories, and multi-tool software engineering environments.

The paper is organized as follows. Section 2 describes the DICAM framework. Section 3 discusses the kinds of application systems our project will focus on constructing. Section 4 characterizes the development methodology we are supporting. Section 5 describes the development environment we are assembling. Section 6 specifies our current status and plans, and includes a scenario showing the kinds of planned development facilities. Section 7 itemizes the various KE/SE relationships listed in the preceding paragraph. Section 8 briefly relates our work with other similar research. Section 9 summarizes the principal points of the paper.

2 The DICAM Framework

We are developing DICAM simultaneously as a "model" or *framework* for understanding control problems and as an *architecture* and related *environment* for building controllers. There are many reasons why we seek to formulate such a unifying framework. Foremost among these is our belief that the difficult, time-consuming and often unsatisfactory process of controller development would benefit from a more "standard" but flexible approach. Our DICAM framework provides a generic but customizable model of controllers that seems to unify a variety of views and experiences in the control, software and knowledge engineering disciplines.

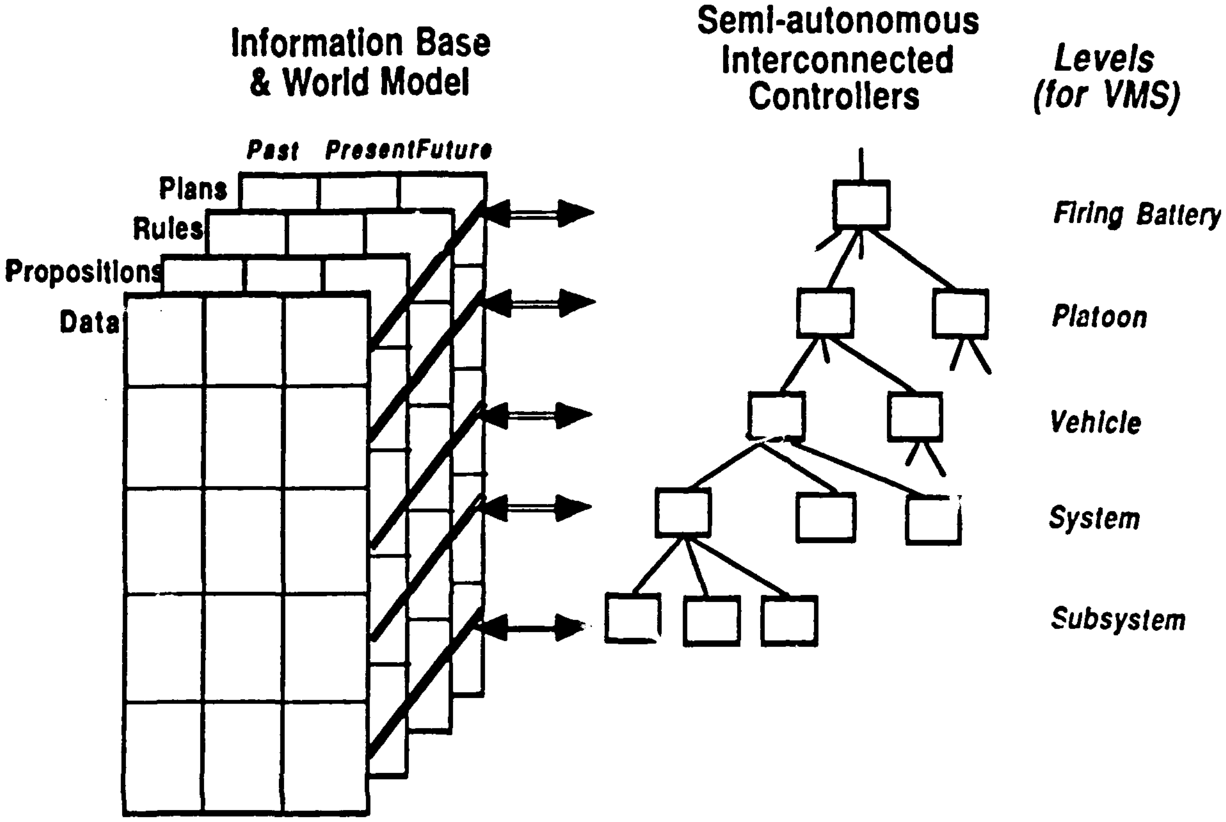


Figure 2-1 The DICAM Reference Architecture

Figure 2-1 illustrates the DICAM Reference Architecture. As with the seven-layer model of OSI, this reference architecture provides a general model of controller applications that prescribes the key system components and their interrelationships. DICAM is closely related to the NASA/NBS reference model for telerobot control systems (NASREM) [Albus 89]. The reference architecture includes two principal components in any distributed intelligent control and management application. First, an *information base and world model* (IB/WM) is a conceptually centralized database/knowledge base that represents the state of the world. It can be viewed as a three-dimensional structure. The first dimension, shown in the y-axis in the figure, stratifies stored information at high to low-levels of aggregation to serve the needs of controllers with corresponding levels of responsibility. The second dimension, shown in the z-axis in the figure, corresponds to the different meta-types of information that must be stored. Here, we have shown four meta-types, termed *data, propositions, rules* and *plans. The data* meta-type includes all types typically representable in a conventional database (e.g., an RDB). *Propositions* and *rules* correspond to the types of information typically stored in an expert system shell or Prolog program. *Plans* include the conditional, generally concurrent, action specifications that controllers generate and execute to make their controlled machinery achieve goals. The third dimension, shown in the x-axis, is that of *time.* Controllers need to look at recent past history, use current state information to make decisions, and forecast future expected situations to determine whether behavior changes are required. The specific elements of an IB/WM required for an application are specified by means of an *IB schema*, analogous to a database schema. The specific abstract datatypes (ADTs) contained in the IB schema are used to standardize the representation of information that is shared among or communicated between controllers.

The second principal component of the DICAM reference architecture is a collection of semi-autonomous interconnected controllers. These controllers are differentiated in terms of the scope of behavior they address, the resources they control, and the time frame spanned by their decisions. "High-level" controllers typically address overall coordination of numerous individuals or organizations of individuals. These top-level controllers are concerned with long time frames and delegate nearly all tasks to their subordinates for execution. "Low-level" controllers, in applications such as manufacturing, vehicle management or robotics, usually have millisecond time constants and have tight coupling of sensor feedback to actuators through servo-control loops. The intermediate levels provide distinct controllers for each natural cluster of concerns. The time constants for decisions between adjacent levels, typically, vary by a factor of 10 ([Albus 89], [Newell 91]). In a space application that requires seven levels of hierarchy, for example, the lowest-level controllers may have a millisecond response requirement, and the top-level controller may plan and control activities spanning a few days.

Figure 2-2 depicts the internal structure of an individual controller in the reference architecture. The controller is actually divided into two separate but interrelated components called the *domain controller (DC)* and the *meta-controller (MC)*. The DC contains several modular functions and prescribes how they interact using dataflow conventions. The functions include sensing, input filtering, situation assessment, planning, plan assumption analysis, execution and effector activation. Each function is shown with a corresponding rectangle. Situation Assessment (SA) interprets incoming sensor information and determines the identity of environmental objects, their location and relative motion, and so forth. In nearly all systems today, SA is performed using a combination of templates, rules and scripts in a knowledge processing approach. These programs follow closely the path originally developed in Hearsay-II [Erman 88]. Methods of planning today are performed in less systematic or routine ways. Typically, however, knowledge engineers work with experts to identify critical goals such as accident avoidance or maximum velocity, and then define parameterized plans whose execution would achieve the corresponding goals. In real time the best generic plans are selected and then *customized* or *particularized* to instantiate the plan with the current situation as context. Typically, the plans prescribe a sequence of actions to be performed over time, and there may be multiple concurrent action streams for different effectors or subordinates. Plans generally continue as long as no major changes in the situation arise that negate their critical assumptions. Reports from subordinates detail the outcome of the various subgoals that were delegated during plan execution. The controller tracks the results to determine whether continuing the plan is appropriate or changes or failures have made replanning necessary

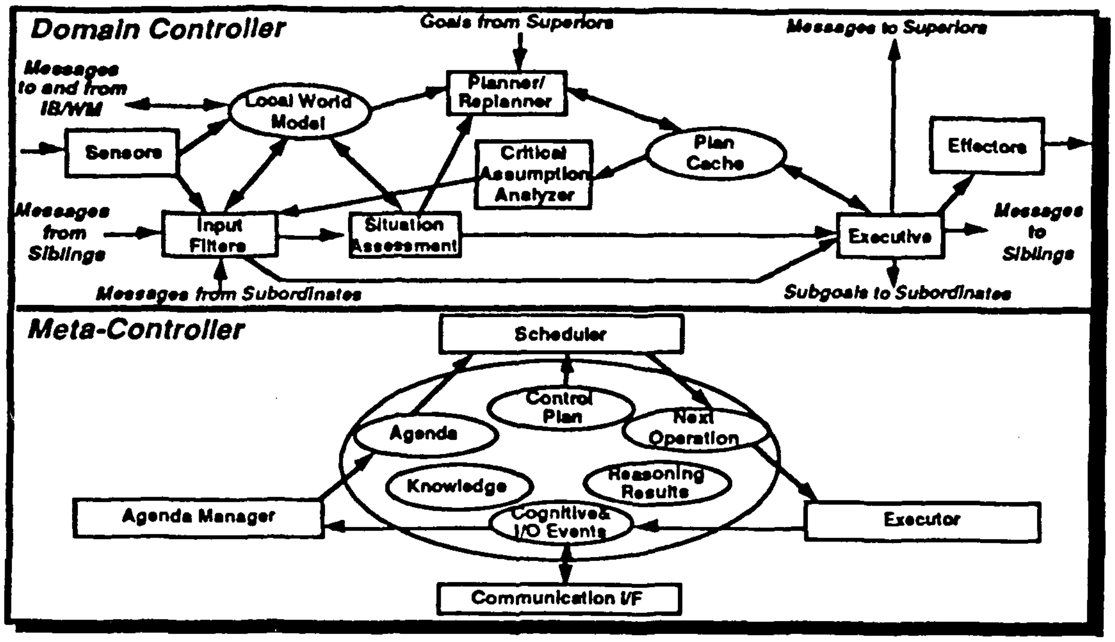


Figure 2-2 Individual Controller Reference Architecture:
Domain Control & Meta-Control

Persistent storage is provided by state objects, shown here as ovals. In particular, the local view of the integrated IB/WM is updated by the sensing, filtering, and situation assessing functions. Proposed plans and executing plans are maintained in a plan cache. The critical assumption analyzer determines which presuppositions of the cached plans need to be continually verified or, equivalently, which critical assumptions make the plans vulnerable. These assumptions are used to drive input analysis. Several messages flow into and out of the DC. The inputs include messages received from a superior controller specifying goals for the controller, messages from sibling controllers at the same level (such as another vehicle in the same group), and messages from subordinates, typically reporting on the outcomes of their efforts. Outputs include subgoals assigned to subordinates for delegated execution as well as messages to siblings, for example, to report on current plan execution objectives or status or to request operating resources.

Although this general DC structure has proved effective in applications such as the Pilot's Associate ([Smith 89], [Lark 90]) and robotics [Becker 89], dataflow programs in general exploit only weak knowledge about when to execute functions. The general rule is to compute any function when all of its inputs are available. However, there are often too many possible instantiations to execute all simultaneously, or even with a small delay. Thus, in situations where more knowledge is required to achieve excellent results with scarce resources, a meta-level of control is required ([Garvey 89], [Hayes-Roth 85], [Hayes-Roth 90a]). Our meta-controller is based on the knowledge-based scheduler of the BB1 blackboard system. This controller utilizes three basic functions to determine on a cyclical basis which pending action is best to execute next: an agenda manager to store and evaluate pending actions; a scheduler to determine the next action based on the degree of fit between goals of a control plan and actions pending on the agenda; and an executor, which gives control to the selected actions.

The DICAM architecture provides several appealing features that can simplify and improve both knowledge engineering and software engineering. We consider five of these briefly:

- The DICAM architecture provides a standardized or generic control framework that can be customized for a wide variety of applications.
- The model is recursive, or *fractal,* in that the same approach to control can be applied at various levels, with different time scales and different domains and scope.
- It designates several specific modular capabilities that combine to produce intelligent control. This makes it easier to develop libraries of reusable controller functions, with as much domain-specialization as is useful. Furthermore, it makes it more likely that the field will formulate and adopt generic approaches to SA and planning tasks, which would improve both the quality of the software and make possible improved task-specific KE environments for such tasks as knowledge acquisition and KB validation.
- It highlights the importance of information sharing and a corresponding IB schema for building integrated intelligent controllers. This should facilitate emergence of standard datatypes for such items as situation models and plans. Because successful development of intelligent controllers requires a combination of effective KE and SE, and requires integration of functions both within individual controllers and between multiple semi-autonomous controllers, precise ADTs, messages, and protocols are highly desirable. Thus, formulation and adoption of an IB schema is a key to advancing SE and KE for intelligent control.
- The DICAM framework suggests a means of unifying conventional control engineering with knowledge-based AI approaches to control. Where the former addresses fast, closed-loop, deterministic, algorithmic solutions to relatively simple and well-specified control problems, the latter is concerned with deliberative, non-deterministic, heuristic solutions to complex and open-ended control problems. We believe that the best solution to any control problem will require a particular mixture of these complementary general approaches. We expect the reference controller, consisting of the DC and MC, to be able to model and specify all such mixtures of control approach. Then, with appropriate compiling techniques, one framework for control could be used to generate high-performance controllers for a wide variety of applications.

3 Controllers in Application: Automated & Mixed Initiative

Although our own research project is focused on technology for producing controllers, it is the applications of controllers that are the end use. In other words, our technology will be evaluated ultimately in terms of the quality of control applications that it helps make possible.

There are two broad classes of applications that we are concerned with: (1) automated or "computer-based" control and (2) human-in-the-loop or "mixed initiative" control. Because the automated case is simpler, we elaborate it further here before considering the mixed initiative case.

3.1 Automated Control

In automated control, the controller is a computer-based system, and it is solely responsible for control. Control of complex systems is divided among cooperating controllers as shown in Figure 3-1.

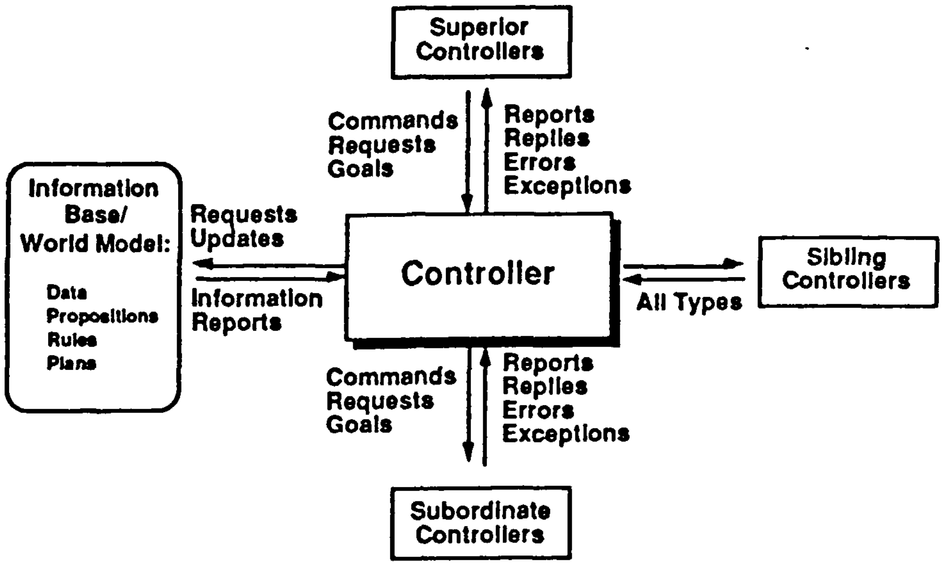


Figure 3-1 Controller Interactions

Communications upward and downward are composed of two basic classes of messages:

- Commands, requests and goals that are inputs to the module.
- Reply, result, error or exception messages sent from the module to the sender from whom it originally received its own task assignment.

Note that controllers are generally persistent. They take on tasks that are temporary and continue until the tasks are completed, regardless of interruptions.

For an autonomous control system to be successful, it need only achieve high performance on various objectives such as risk avoidance, throughput or travel time, etc. Success or failure on the task can be attributed directly to the effectiveness of the individuals and their cooperation

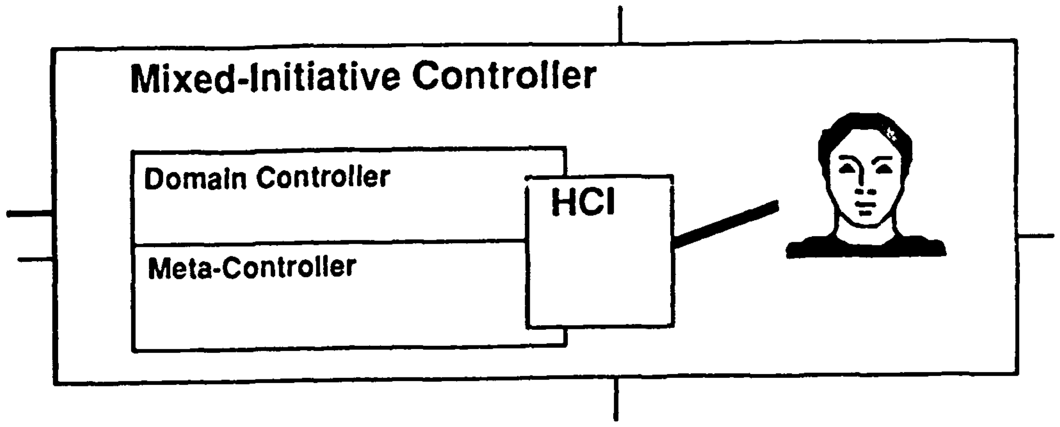


Figure 3-2 Controller Architecture Augmented for Human Component Interface

3.2 Mixed Initiative Control

In contrast with an automated controller, the mixed initiative controller with a "person in the loop" is more complicated, and the metrics for system performance are generally more subjective. Figure 3-2 illustrates how we are currently integrating the human into our generic control approach. In place of the automated control comprising only the DC and MC, the mixed-initiative controller also includes a human-computer interface (HCI) module and a human (usually just one per controller). The HCI determines what information to supply from the computer to the person and how to interpret the information provided by the human's actions. Mixed-initiative controllers can vary over several additional dimensions corresponding to: (1) the distribution of authority between person and machine; (2) the distribution of tasks; (3) the bandwidth of communication between the two; (4) the comparative intelligence levels of the two; and (5) the principal value sought from the computer system, from active idea generation and improved information presentation to more passive decision assessment or error monitoring.

Depending on the specific values on these dimensions pertinent to a particular application, the HCI will be appropriately specialized. Typical in applications like those of the Pilot's Associate, where the HCI was called the "pilot-vehicle interface (PVI)", and our howitzer demonstration task, the HCI is a combination of expert, advisor, monitor, assistant and clerk. In both applications, the human is the ultimate and only authority, but the human generally delegates authority for routine decision-making to the computer system during training sessions and pre-flight or pre-battle mission initialization or set-up activities. Furthermore, because policies or practicalities may dictate that only humans can make some kinds of decisions, our architecture must be capable of modeling systems in which humans directly perform some DC or MC functions. In this way, a full model of the mixed-initiative controller would show a partitioning of DC and MC functions and an assignment of some of them to the human as processor. The methodology we are developing supports this complication explicitly by separating the specification of functionality from processor allocation decisions. More generally, particular components within the domain controller and meta-controller may require replacement or customization to support human interaction and implement the HCI.

In short, humans are "in the loop" for three basic reasons. First, organizations today for historical reasons are largely defined in terms of human roles. In these contexts, humans are per-

In short, humans are "in the loop" for three basic reasons. First, organizations today for historical reasons are largely defined in terms of human roles. In these contexts, humans are perceived as the active controlling agents and current practices are based on human approaches, skills, and policies. Computers are not seen as *team members*, generally, so the computer-based controllers are used in limited supporting ways. Second, humans are almost everywhere designated as the responsible decision-makers. Therefore, computer-based controllers in many such contexts are perceived primarily as decision-support systems. Third, humans are in many cases the ideal information processors where information is subjective, visual, or potentially critical. Therefore, systems must be built around these strengths, focusing on making the computer non-interfering and complementary. Because of these important reasons, the DICAM architecture and associated environment must be capable of representing, supporting and developing intelligent control systems which have arbitrary mixtures of human and computer-based initiative, cooperation, and authority.

4 The Development Methodology

Our basic methodology for development of DICAM applications consists of a blackboard-like environment where the "blackboard" is a *development workspace* and the "knowledge sources" are system developers augmented by a wide variety of computer-based tools, including some expert systems that are capable of autonomous development activity.

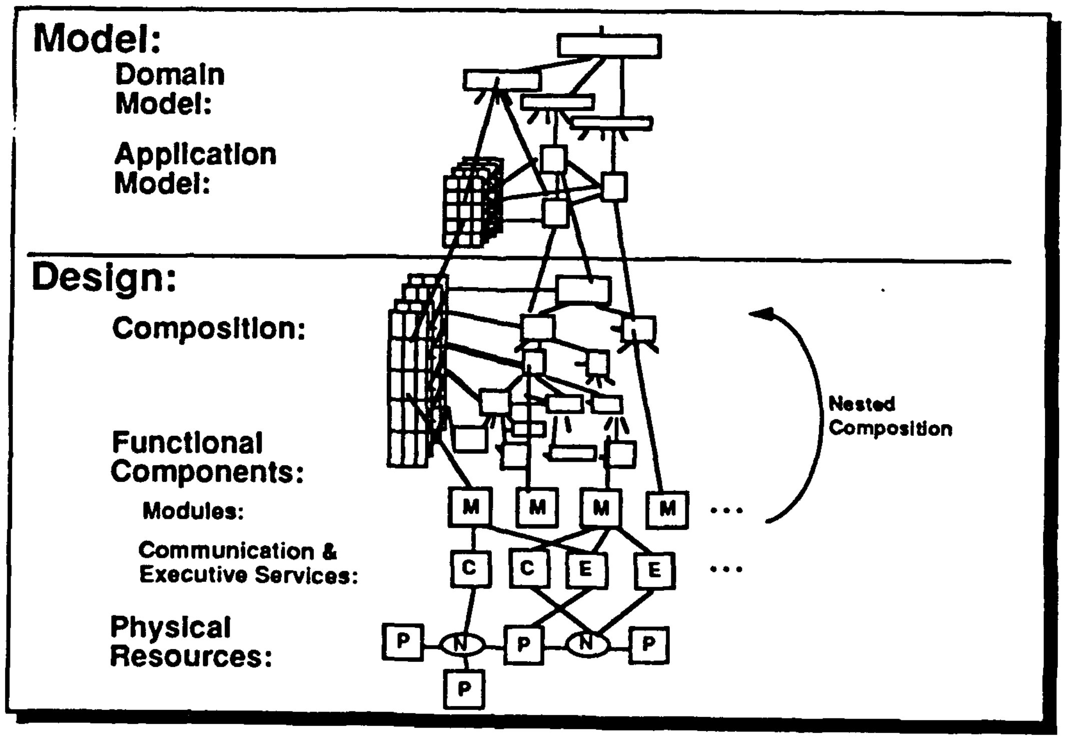


Figure 4-1 The Development Workspace

4.1 Development Workspace

The development workspace contains a representation of the emerging system being developed incrementally over time. Its elements represent nearly independent decisions or specifications, linked into a "web" of mutually supporting decisions that both specify the system design and justify it. We have combined three lines of research in formulating this development workspace. First, we have drawn on the blackboard model and opportunistic reasoning ([Erman 80], [Hayes-Roth 79], [Hayes-Roth 86]) as an organizing methodology for incremental design and development processes. Second, we have adopted the emphasis of the domain analysis and domain-specific architecture approach to software specification, reuse and rapid development [Prieto-Díaz 91]. Lastly, we have adopted and generalized the approach of module-oriented programming from our previous research on ABE ([Erman 1988], [Hayes-Roth 89], [Hayes-Roth 91]). This includes the ideas of recursive modular composition, distributed control through message passing using ADTs, system construction through module composition, and system realization by deferred binding of processors to modules.

Specifically, the workspace provides a multi-faceted, multi-level representation of DICAM software applications. It provides means for describing the domain model, i.e., the general characteristics of the task and environment in which the vehicle or machinery will operate. The general domain model is then augmented with specialized information about the specific application being built, such as how many vehicles, the distances to be travelled, the specific threats and so forth that the application will address.

At a lower, more concrete level, the workspace provides means for representing the functional components and the physical resources that make up the controlled system, and it describes how the functional components are composed and how they are implemented using specific processors, communication capabilities or other machinery.

In addition, the workspace provides means for representing the status of the software development process, including the history of activities and characteristics of the current overall development.

As is typical of blackboard systems, the workspace provides means of representing decisions and using state changes to trigger the invocation of appropriate tools. Decisions in this workspace range from abstract characterizations of components such as requirements or goals to particular specifications, including detailed functional characterization or specific software or hardware packages that realize the required capability. We have not yet settled upon final or formal representation sublanguages for each level, but are considering various alternatives that are being suggested in other groups' efforts to conceive potentially standardizable descriptions of modules and module interfaces (e.g., the DARPA module interconnect formalism, the DoD STARS repositories, etc.). Regardless of which specific formalism is used, the description of modules must include input/output datatypes, function characterizations, implementation requirements, domain assumptions, and performance metrics. When making a design decision, the developer specifies (or the development system infers) some or all of these attributes along with his or her name and some rationale. As in all blackboard systems, decisions are changeable, and multiple competing decisions may coexist. Ultimately, those decisions that form the best coherent "web" win: these decisions constitute the overall system specification, from requirements to implementation, which particularizes the domain and application models.

4.2 Development Methods and Tools

We intend to support a wide variety of development processes. In fact, the most general model we support is *opportunistic design,* where developers move freely around in the workspace of decisions as interesting situations arise. Typical decisions are to add a requirement to a component specification, to derive a new requirement at a lower level deemed necessary or desirable to helping satisfy some requirement at a higher level, and to implement a function by selecting a module from the repository. Also, we anticipate that most systems will be designed by redesigning old systems, so we provide means for storing, browsing, and retrieving old designs from a repository. The components of the library correspond to the types of objects previously discussed: DC & MC components and whole systems; domain and application models and components thereof; and requirements on, specifications for, and implementations of functional modules and compositions thereof. Similar library services are available to support creation and reuse of abstract datatypes (ADTs) that are used by the IB/WM schema and within intermodule messages. In short, the developers can assemble a system from old or new parts, potentially in any order they wish.

Most organizations, however, will want to suggest or impose a prescriptive process model (a *process plan*) upon the development process. This would determine which concerns would need to be addressed first, and might constrain all decisions to accord with some set of rules. We will support this by allowing organizations to implement their own process models using a pattern-directed system that monitors the process state and suggests or enforces use of particular tools when corresponding key events occur. This means, in effect, the prescriptive process model is the control plan for the development process. This allows us to support any methodology explicitly.

Other aspects of the methodology are too numerous to describe in detail; however, Table 1 presents these and their key elements concisely.

Aspect	Elements
Opportunistic Design	Multiple levels and representations Abstract to particular characterizations Incremental decisions Linked decisions form design web Prescriptive process models permitted Humans and computer tools cooperate
Controller Architecture	Generic modules Flexible, tailorable controllers Schema of ADTs for IB/WM Message processing using ADTs and intermodule protocols Distributed control Fractal control model
Information Base/World Model	Shared data managed by IB/WM Conceptually centralized, single-copy, but allows physical partitioning Typically distributed Time response must satisfy requirements ranging from sub-millisecond to a few seconds Different levels of aggregation Different meta-types: data, propositions, rules, plans Temporally organized and continually renewing
KB Design Assistants	Mini-expert systems watch process state and advise user at key events Tool-use expert systems help humans apply development tools
Repository	Stores and uses partial matches to retrieve "components" at any level Components classified in taxonomy from generic to particular Domain-specific customizations available to particularize generics
Engineering Foci	Domain modeling Requirements engineering Knowledge engineering, about DICAM and DICAM development tools and methods Performance objectives, measurement and attainment
Component Characterization	Goals & Constraints Models: Behavior, timing, functionality Interfaces Datatypes Module partners Conversation types Protocols Messages to other devices Resource/environment prerequisites

Table 1 Aspects of the Development Methodology

5 Development Support Environment

We are assembling an Application Development Support Environment (ADSE) for DICAM applications. It is depicted in Figure 5-1. In addition to the Development Workspace, the ADSE contains several other principal features which we describe briefly in the following paragraphs.

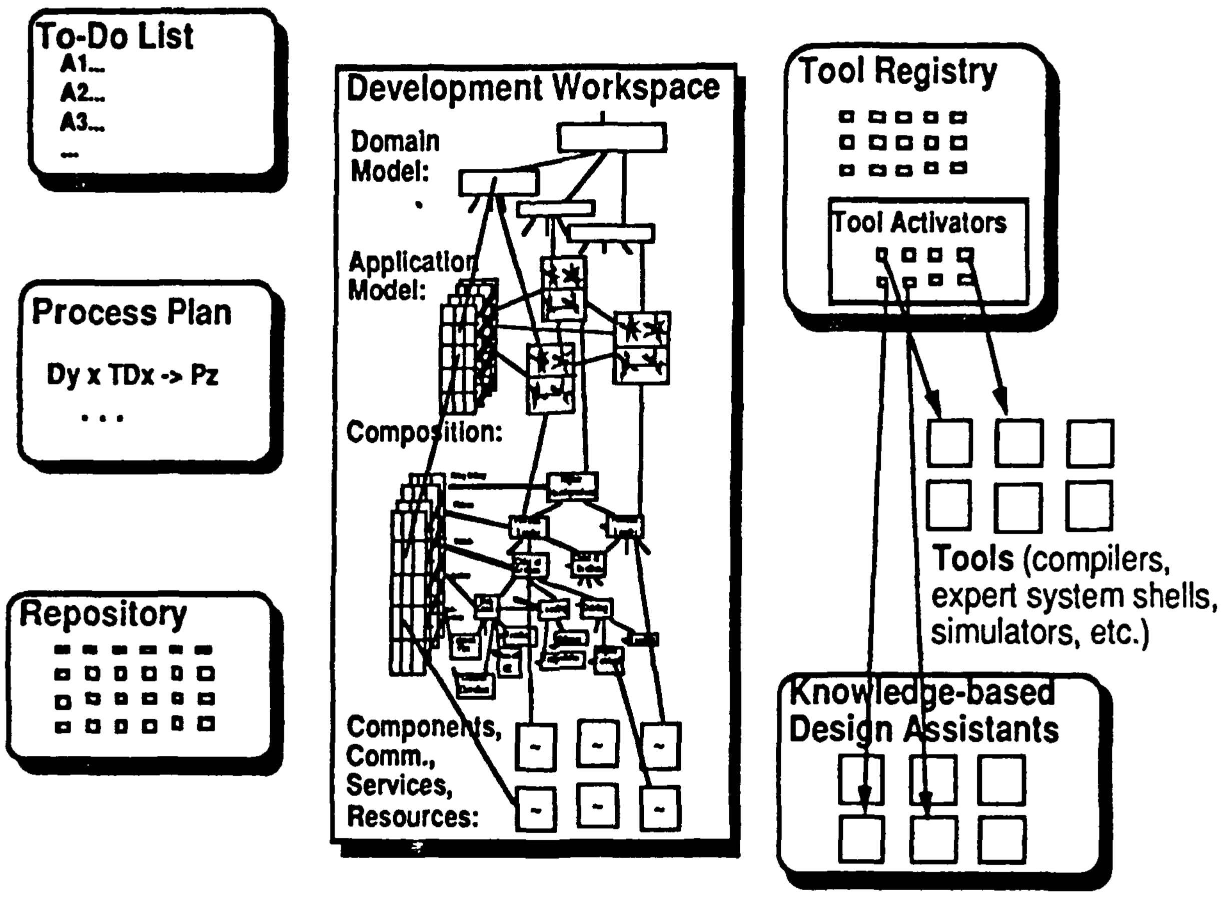


Figure 5-1 The Application Development Support Environment (ADSE)

A *To-Do List* is provided that keeps an agenda of pending tasks for the software developers. The To-Do list is a central interface between the developer and the development support environment. Actions on this list are generated either by the Process Plan, by Knowledge-Based Design Assistants (KBDAs), or by developers themselves. As with blackboard systems, actions are triggered (i.e., enabled) when the state of the Workspace matches a pattern of interest. In some cases, as specified in the Process Plan (see next), a triggered action is executed automatically; in other cases, it is up to the developer to choose if and when to initiate the action.

A *Process Plan* is supported that effectively maps patterns of interest found in the Development Workspace and the current To-Do List into proposed actions. The proposed actions (shown as Pz) in the figure might include any of the following: make a specific design decision; apply a particular tool to a particular design component with certain parameters; raise the priority on doing one pending task over others, etc. Our plan is to support a wide variety of SE methodologies by providing a general mechanism for representing and implementing corresponding process plans.

A *Repository* of reusable components is provided that stores, classifies, and searches for previously used Development Workspace structures. Typically these include reusable domain models, application characteristics, generic function modules, specific implementation modules, and data to customize or particularize generic functions for specific application domains.

A *Tool Registry* provides mechanisms for enrolling software development tools, describing their required inputs and associated outputs in terms of patterns that match characteristics found in the Workspace or Process Plan and, finally, providing *Tool Activators* that can automate or semi-automate invocation and application of tools. The tools consist of compilers, generators, simulators, expert system shells, etc.

Lastly, the ADSE incorporates specialized tools called KBDAs that provide knowledge-based assistance in to the software development process. These tools can include, for example: requirements analyzers that suggest appropriate reusable components; redesign advisors that suggest ways to modify an existing design in light of a change in requirements or capabilities; and intelligent interfaces that set up and run complex tools to assist a developer in generating or analyzing some code.

Aside from the overall project focus on DICAM applications, which affects the types of tools, models, and compositions we construct and the kinds of knowledge appropriate for the repository and KBDAs, our ADSE should prove generally applicable to software development.

To implement the ADSE we are using a number of "off-the-shelf" technologies. Chief among these are: ABE [Hayes-Roth 91], as an integration environment for tools, a composition framework for modular, real-time applications, and a catalog and classification system for the reuse repository; BB1 as an incremental workspace, process model interpreter, and agenda manager; M.4, a commercial expert system shell, for building the KBDAs; and the Requirements Manager (RM), a DARPA-sponsored software product for collecting, managing, and evaluating application requirements and validating application designs against requirements [Fiksel 90]. We are also evaluating many other commercial and research SE tools for use in the ADSE [cf. NIST 91].

6 Status and Plans

We are currently applying the general approach described in this paper to demonstration problems chosen from defense applications. As an example, consider the task of achieving intelligent control of field artillery systems, such as mobile howitzers. Howitzers, like other military vehicles, are self-propelled, mobile vehicles with offensive guns. Their primary mission is ground-based artillery shelling of over-the-horizon targets. They are very similar to tanks, armored personal carriers, and helicopters in general information processing terms. Thus, all military vehicles of this sort share elements of the domain model, but differ increasingly as these models and the corresponding application model become detailed.

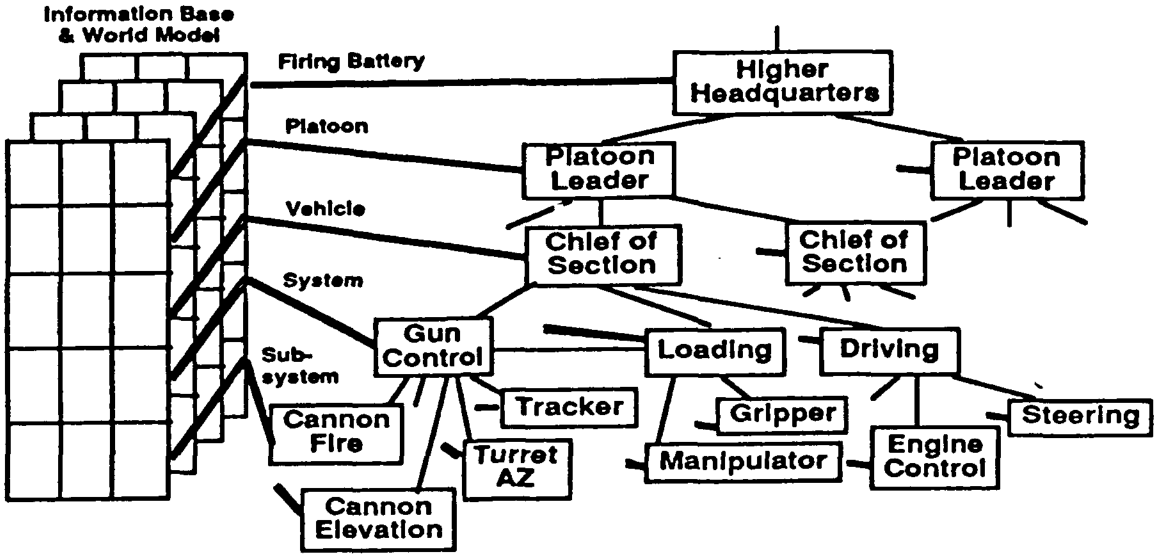


Figure 6-1 Specializing the Generic DICAM Controller Structure for a Particular Howitzer

The general DICAM architecture is specialized for Army Vehicle Management Systems (see Figure 6-1) by the selection of levels: battalion, battery, platoon, vehicle (section), system, subsystem. Here it is further specialized for the hypothetical "ABC Howitzer" by the selection of functional controllers and their relationships. Each group of ABC howitzers is headed by a Platoon Leader who reports to higher headquarters. The Chief of Section of each vehicle reports to the Platoon Leader and is responsible for the Gun Control, Loading, and Driving functions.

The Gun Control, Loading, and Driving functions might be individually manned or automated. The functions below them are typically not separately manned. At this level of description, there is no distinction between automated and mixed-initiative controller functionalities.

Following the domain-specific approach, after developing the generic domain model, the next task for system developers is to elaborate the application model. The current task application model is illustrated in Figure 6-2. The figure shows an enumeration of desired functionalities associated with each level of control. The vertical lines connecting several functions indicate generic functions that appear repeatedly across different control levels, such as tasking subordinates with subgoals or performing external and system status analyses as part of situation assessment. These functionalities are also common across the analogous components in other vehicles: tanks, missile launchers, infantry fighting vehicles, etc. Thus, there are two levels of functional similarity:

- across different components within a vehicle, as shown here, and
- across the similar components in different vehicles.

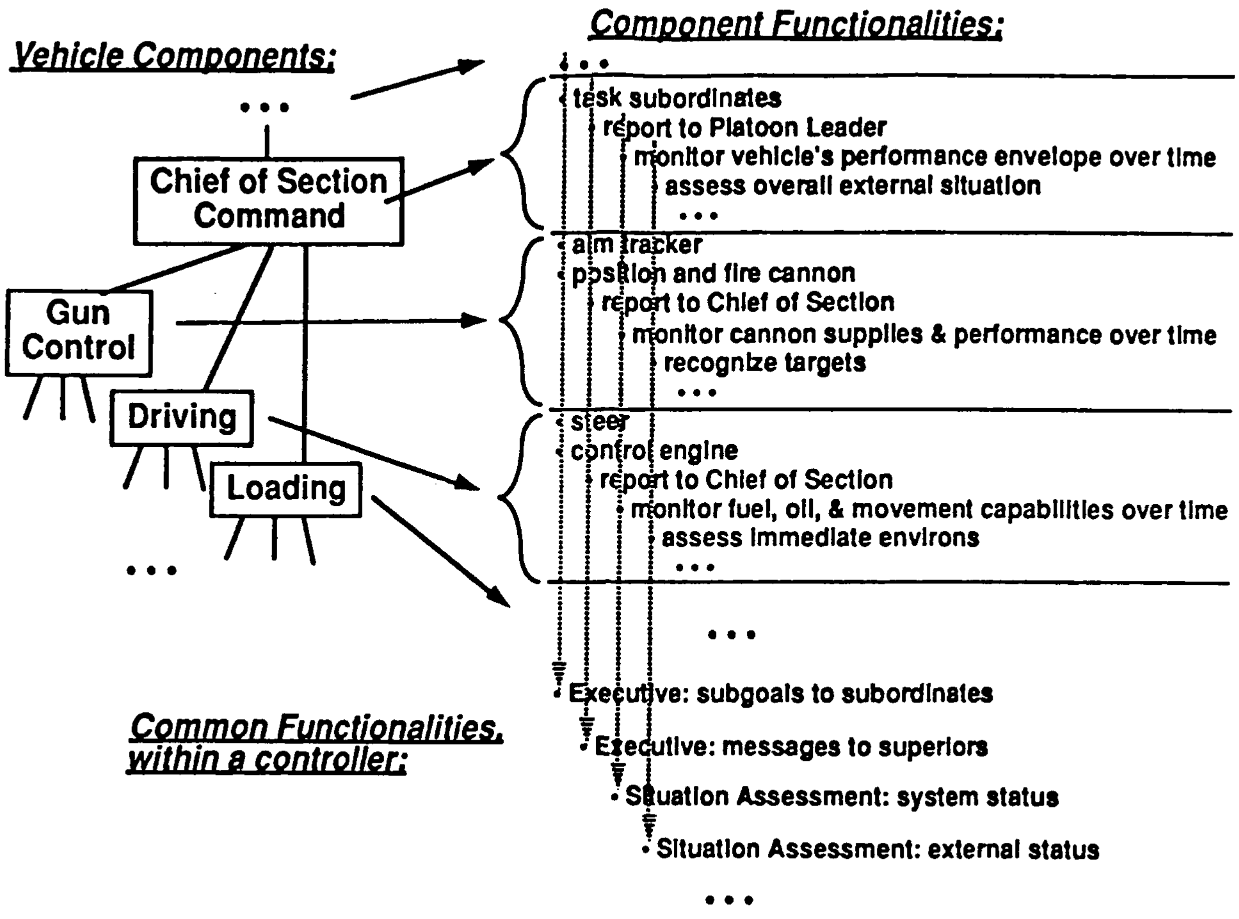

Component Functionalities:
Vehicle Components:
Chief of Section Command
Gun Control
Driving
Loading
task subordinates
report to Platoon Leader
monitor vehicle's performance envelope over time
assess overall external situation
aim tracker
position and fire cannon
report to Chief of Section
monitor cannon supplies & performance over time
recognize targets
steer
control engine
report to Chief of Section
monitor fuel, oil, & movement capabilities over time
assess immediate environs
Common Functionalities, within a controller:
Executive: subgoals to subordinates
Executive: messages to superiors
Situation Assessment: system status
Situation Assessment: external status

Figure 6-2 Informal Task Application Model for a Portion of the Howitzer Example

To convert the informal task analysis into a more formal, explicit application model, the system developer selects from among generic functionalities in the reference architecture, specializing and customizing them for the particular needs of the target vehicle. This is illustrated in Figure 6-3. In that figure we see, for example, that the functions of the Chief of Section in the areas of planning, situation assessment and meta-control are mapped into specific functional requirements. These specifications include a requirement that site selection for moving the vehicle be completed within 25 minutes, that future fuel levels be estimated with less than 10% error, that evaluations by situation assessment of candidate sites be completed in less than a minute, etc.

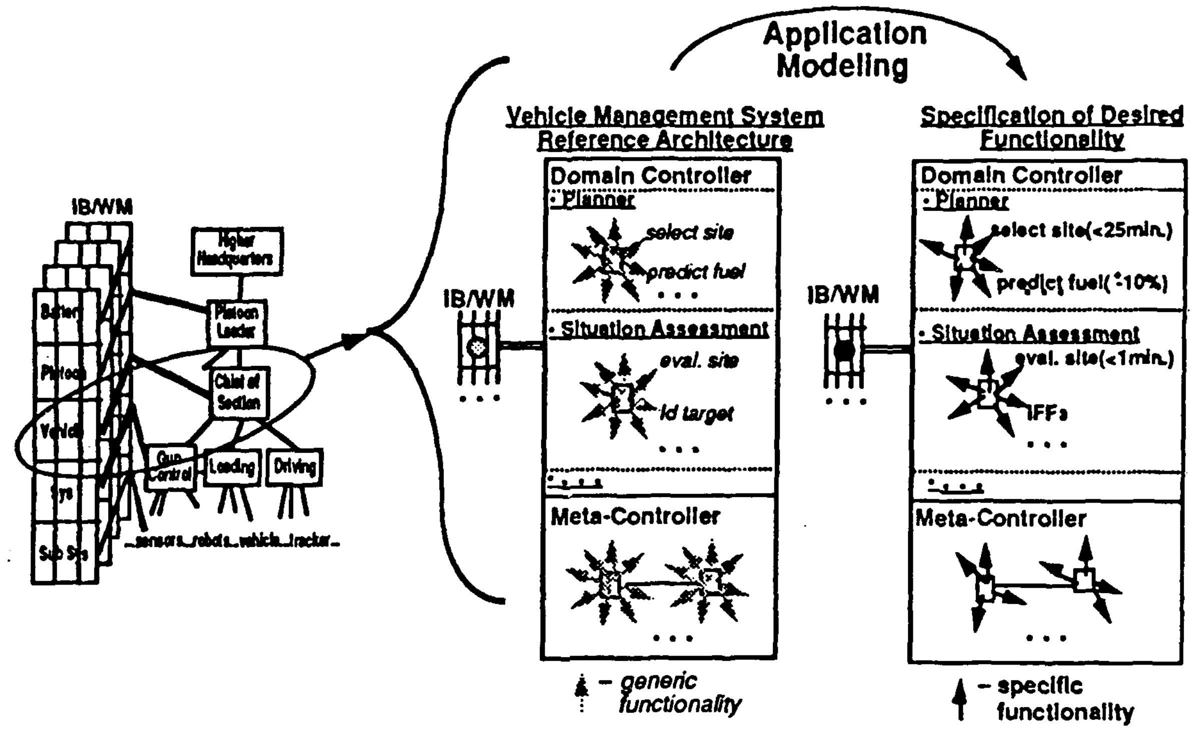


Figure 6-3 Building the Application Model

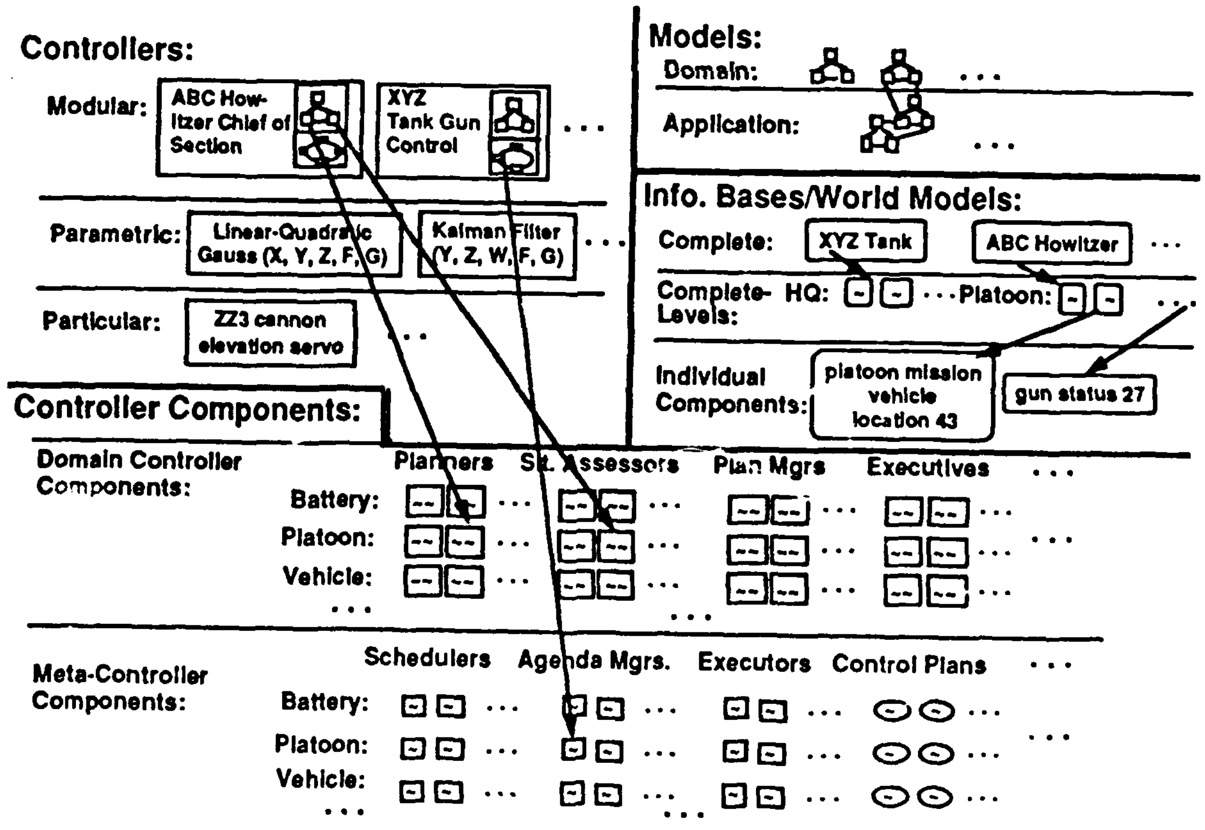


Figure 6-4 Partial Layout of the Repository

Our approach toward organizing the repository is illustrated in Figure 6-4. Classifications are maintained for controllers themselves, including modular, parametric and particular implementations; for domain and application models; for information bases at various levels of completeness; and for components of controllers including modules of domain controllers and meta controllers.

6.1 A Partial Development Scenario

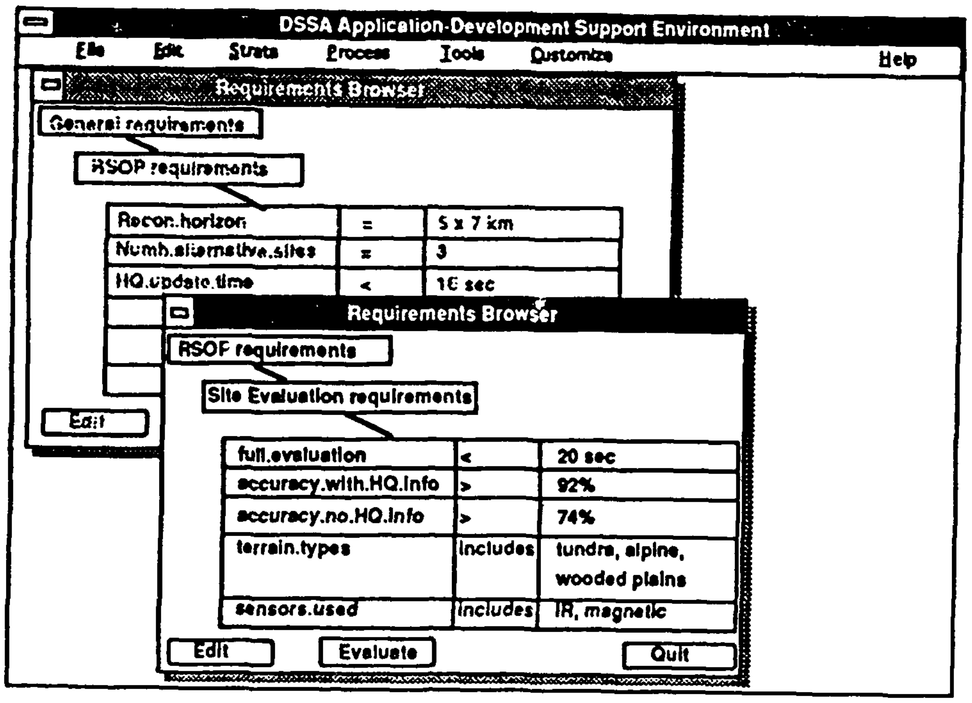
Our research on the DICAM-DSSA program began late in 1991 and will continue through 1995. Our current emphasis is on assembling an initial infrastructure for the ADSE, developing a taxonomy of controllers to help classify and organize controller elements in the repository, initial software development for the hybrid controller itself, and domain analysis and design for a DICAM-DSSA demonstration in the howitzer domain. We now present a brief tour of some of the initial elements in this demonstration, with the following overall sequence:

- The scenario starts in the midst of an on-going design project
- The user changes some requirements, then evaluates them to see where the design needs work
- These changes cause tasks to be posted to the To-Do list
- The user gets advice on one of these tasks from a design advisor
- The advisor helps in the selection of a more capable processor and also helps in propagating the effects of the decision
- The user decides to fix another problem by finding an appropriate module from some other vehicle's design
- The user queries the repository for this module, examines the alternatives, and selects one

The goal of this scenario is to give a flavor of the ADSE by showing the use of various tools and typical interactions. This brief scenario assumes the user is designing the ABC Howitzer section chief's decision support system by modifying that of a missile launcher and borrowing relevant pieces from a tank design. Design is focused at this point on the reconnaissance, selection, and occupation of position (RSOP) and its Site Assessment module.

Figure 6-5 shows the Requirements Manager in use for specifying and tracking the hierarchical requirements for the RSOP's Site Evaluation capability. The user can evaluate which requirements are satisfied using a variety of more or less sophisticated evaluation methods, including simulation and analysis.

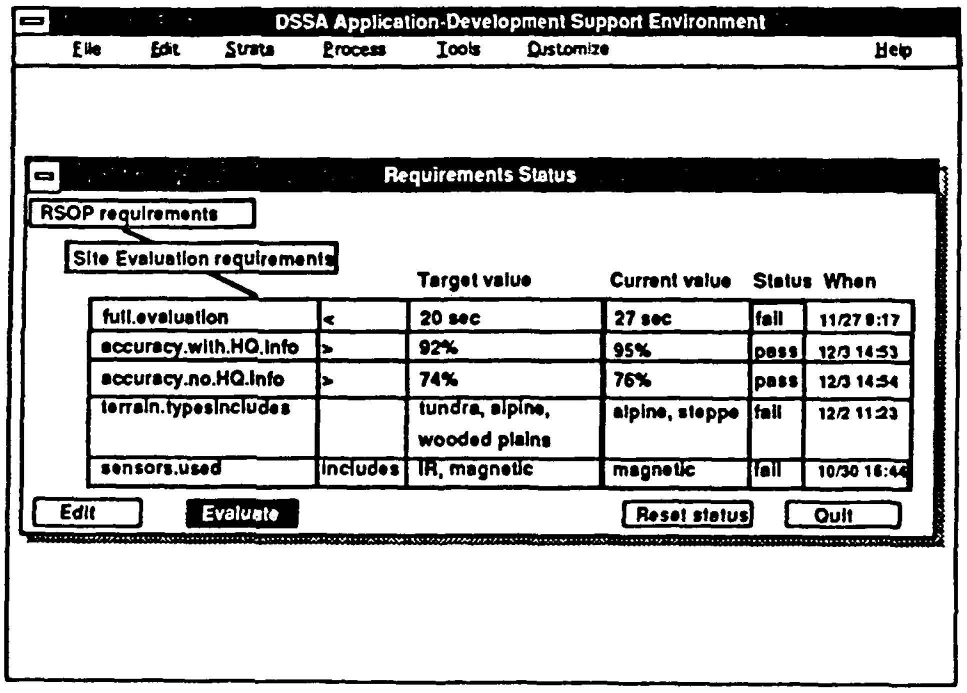
In Figure 6-6, each requirement has a status showing the result of most recent evaluation, and the time of that evaluation. Here the user selected a Site Evaluation module from another design and checked to see what doesn't work. Among other problems, it looks like it is too slow as is. When a requirement fails evaluation, the requirements manager can automatically post a task to the To-Do list to effect some change that will allow the requirement to be satisfied.



Recon.horizon	=	5 x 7 km
Numb.alternative.sites	=	3
HQ.update.time	<	10 sec

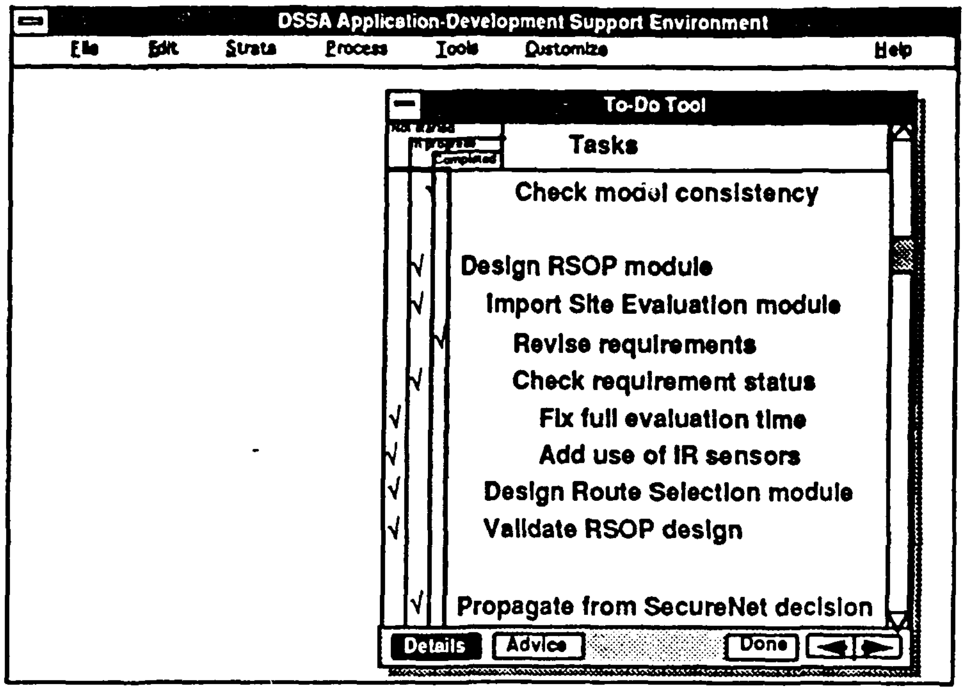
full.evaluation	<	20 sec
accuracy.with.HQ.info	>	92%
accuracy.no.HQ.info	>	74%
terrain.types	includes	tundra, alpine, wooded plains
sensors.used	includes	IR, magnetic

Figure 6-5 Browsing RSOP Requirements



		Target value	Current value	Status	When
full.evaluation	<	20 sec	27 sec	fail	11/27 9:17
accuracy.with.HQ.info	>	92%	95%	pass	12/3 14:53
accuracy.no.HQ.info	>	74%	76%	pass	12/3 14:54
terrain.typesincludes		tundra, alpine, wooded plains	alpine, steppe	fail	12/2 11:23
sensors.used	includes	IR, magnetic	magnetic	fail	10/30 16:44

Figure 6-6 Evaluating a Requirements Group



Check model consistency

Design RSOP module

Import Site Evaluation module

Revise requirements

Check requirement status

Fix full evaluation time

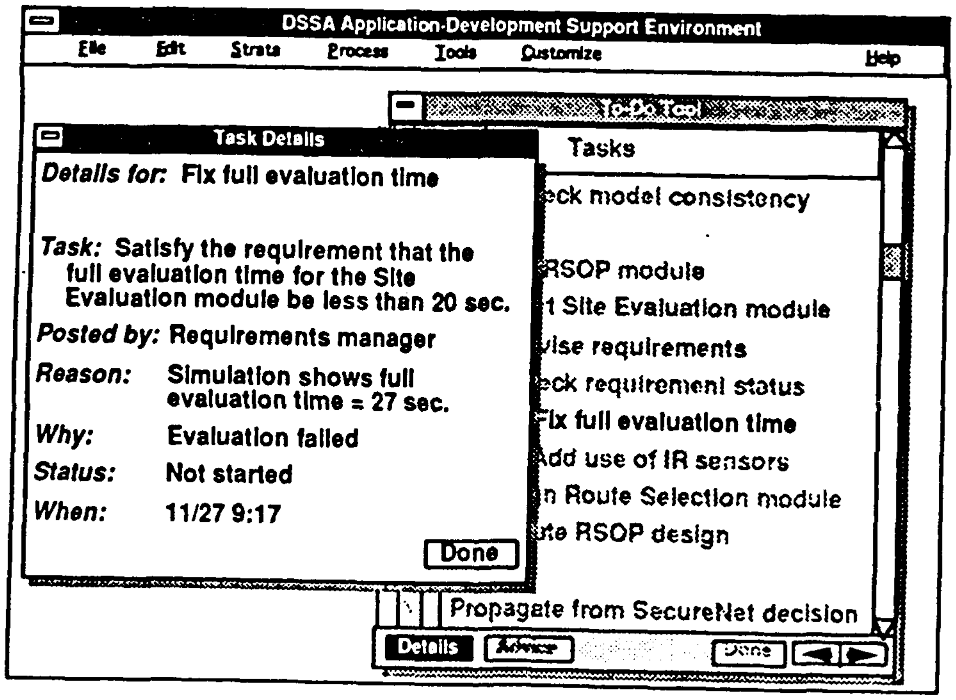
Add use of IR sensors

Design Route Selection module

Validate RSOP design

Propagate from SecureNet decision

Figure 6-7 Checking the To-Do List



Details for: Fix full evaluation time

Task: Satisfy the requirement that the full evaluation time for the Site Evaluation module be less than 20 sec.

Posted by: Requirements manager

Reason: Simulation shows full evaluation time = 27 sec.

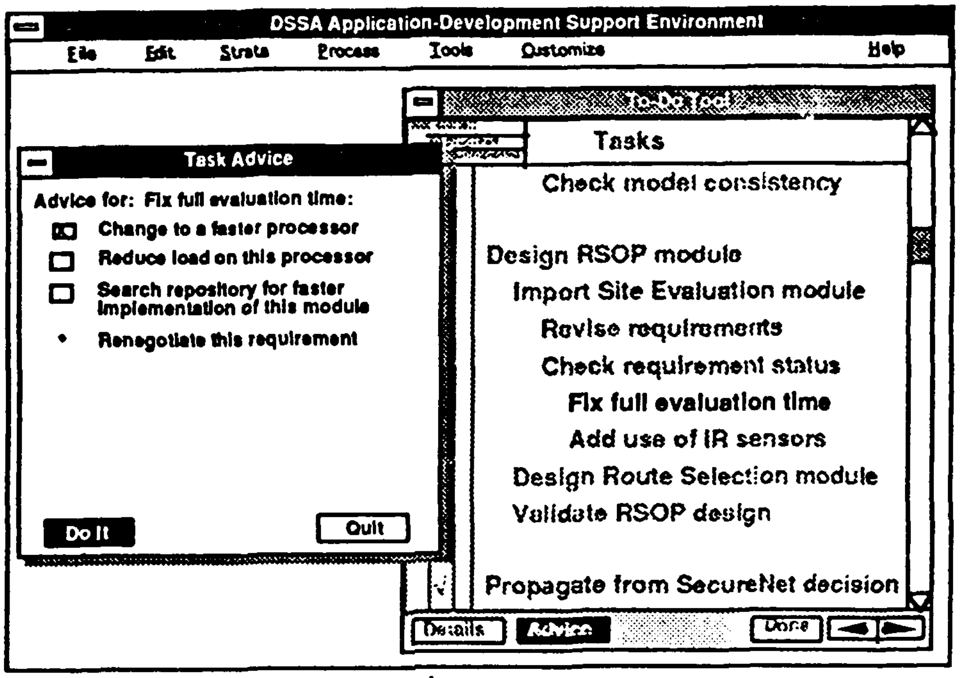
Why: Evaluation failed

Status: Not started

When: 11/27 9:17

Done

Figure 6-8 Getting More Details on the Task



Advice for: Fix full evaluation time:
- [✓] Change to a faster processor
- [ ] Reduce load on this processor
- [ ] Search repository for faster implementation of this module
- • Renegotiate this requirement

Tasks

Check model consistency

Design RSOP module
- Import Site Evaluation module
  - Revise requirements
  - Check requirement status
    - Fix full evaluation time
    - Add use of IR sensors
- Design Route Selection module
- Validate RSOP design

Propagate from SecureNet decision

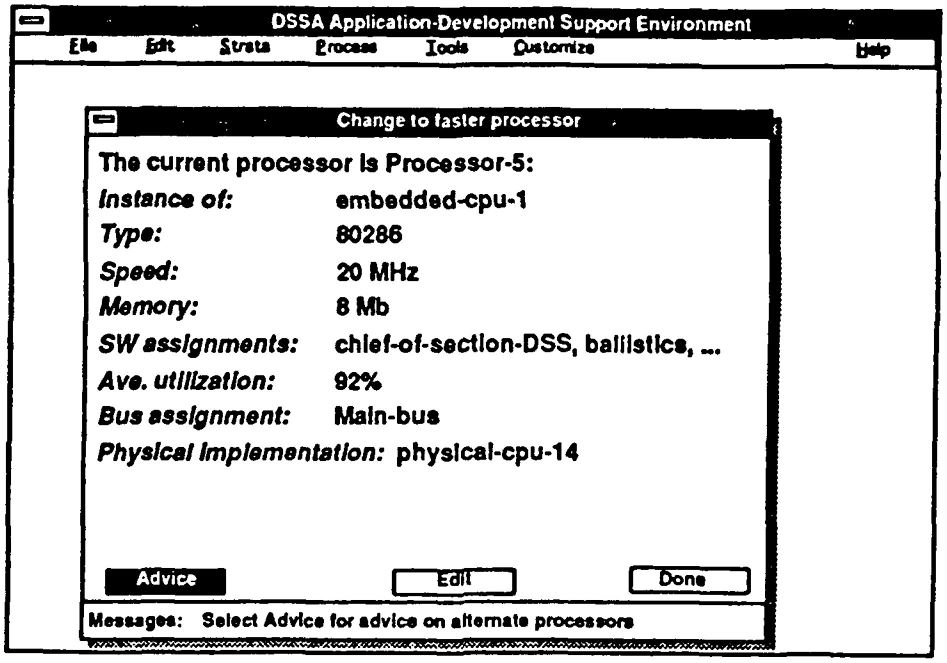
Figure 6-9 Getting Advice on How to Complete the Task

Figure 6-7 illustrates the To-Do list, a central interface for the user and ADSE to communicate focus and open tasks. The interface uses an outline metaphor. Tasks have three possible status values: not started, in progress, done. The user can button a task and ask for details or for advice on how to do it. At any point the user can edit the list and change status values; so can tools in the ADSE. Note that a task to fix the failed full evaluation time requirement from Figure 6-6 has been posted.

In Figure 6-8, the user has asked for more details on the new task. This display shows a textual description along with reasons for its being posted. The user can edit this information when appropriate.

Figure 6-9 illustrates what happens when the user asks for advice on how to satisfy the task. This might be stored text or a KBDA might compute what can be done. The **Do It** button fires up ones the user has selected. The absence of a box means the ADSE cannot help work on this task. **Do It** for **Change to a faster processor** puts the user in the editor and allows definition or editing of a new processor specification.

Figure 6-10 shows what happens when a process program or *script* executes. This script begins by showing the user what processor is currently specified and invites the user to request advice, which the user does. In this case advice is provided by a KBDA which specializes in selecting appropriate processors

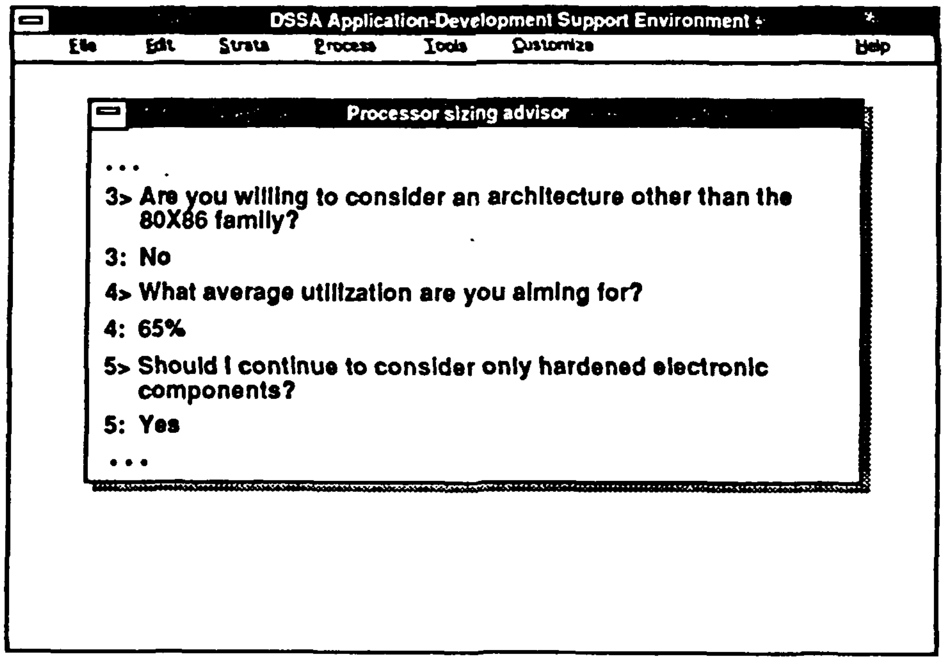

DSSA Application-Development Support Environment
File Edit Strata Process Tools Customize Help
Change to faster processor

The current processor is Processor-5:

Instance of:	embedded-cpu-1
Type:	80286
Speed:	20 MHz
Memory:	8 Mb
SW assignments:	chief-of-section-DSS, ballistics, ...
Ave. utilization:	92%
Bus assignment:	Main-bus
Physical implementation:	physical-cpu-14

Messages: Select Advice for advice on alternate processors

Figure 6-10 Script for Fix Shows Existing Processor


DSSA Application-Development Support Environment
File Edit Strata Process Tools Customize Help
Processor sizing advisor

...

3> Are you willing to consider an architecture other than the 80X86 family?

3: No

4> What average utilization are you aiming for?

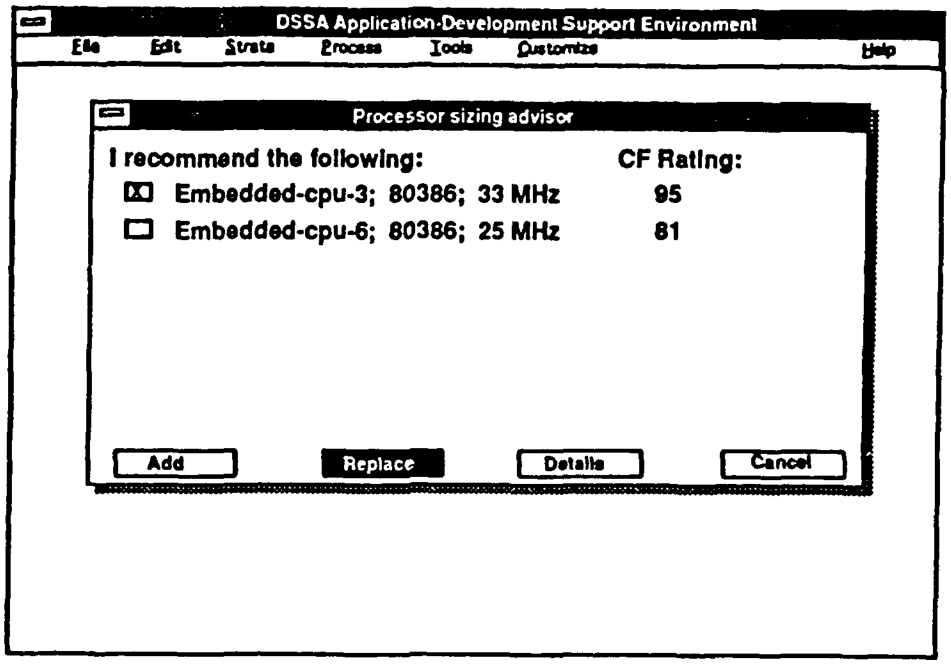
4: 65%

5> Should I continue to consider only hardened electronic components?

5: Yes

...

Figure 6-11 KBDA Conducts a Dialog with the User


DSSA Application-Development Support Environment
File Edit Strata Process Tools Customize Help
Processor sizing advisor
Add Replace Details Cancel

I recommend the following: CF Rating:

[✓ Embedded-cpu-3; 80386; 33 MHz] 95

Embedded-cpu-6; 80386; 25 MHz 81

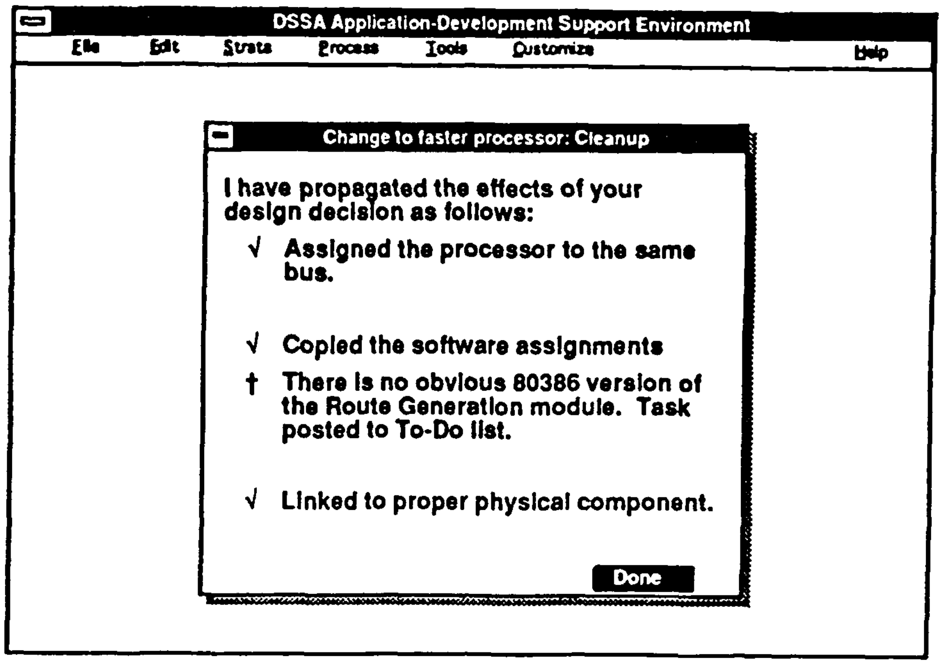
Figure 6-12 User Selects from the Choices Offered

In Figure 6-11, the processor specialist KBDA begins by first collecting and evaluating user requirements and design flexibilities. Then it queries the repository to see which candidate processors it can recommend.

Figure 6-12 shows that the KBDA finds two candidate processors for this user, and it presents each along with a rating of goodness. The "CF" is a "certainty factor" computed by the underlying expert system shell. The user selects the first one listed as best fitting the design objectives and replaces the old processor with this new one.

In Figure 6-13, we see how another KBDA helps manage the many possible effects of a design decision. This KBDA helps the user by propagating the more mechanical effects, copying the contents of the old processor's attribute values into corresponding ones in the description of the new one. It notices one of the modules assigned to the processor has no obvious implementation version for the 386. This is posted to the To-do list for resolution.

The KBDA's results show up in the to-do list in Figure 6-14. The user has made progress on the task of fixing evaluation time by adding a new, faster processor for the application. This also spawned some new subtasks. A new task to find a 386 version of the route generation module has been posted, even though it is not an explicit child of the "fix processor speed" task. Since the processor has been selected, the task is characterized quite specifically. The user still need to re-evaluate requirements and to test that this design change really works.


DSSA Application-Development Support Environment
File Edit Strata Process Tools Customize Help
Change to faster processor: Cleanup
Done

I have propagated the effects of your design decision as follows:

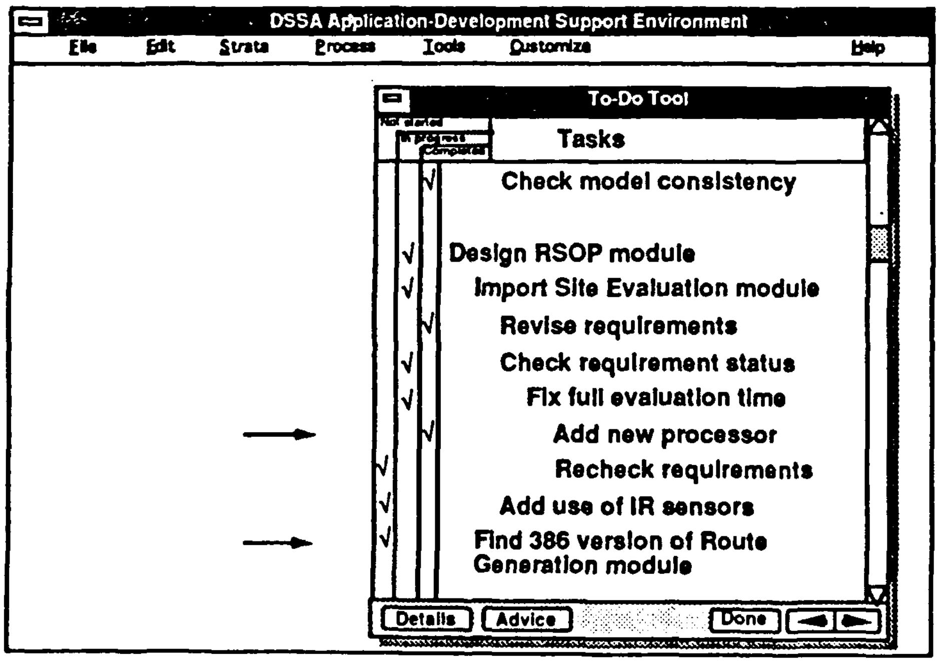
√ Assigned the processor to the same bus.

√ Copied the software assignments

† There is no obvious 80386 version of the Route Generation module. Task posted to To-Do list.

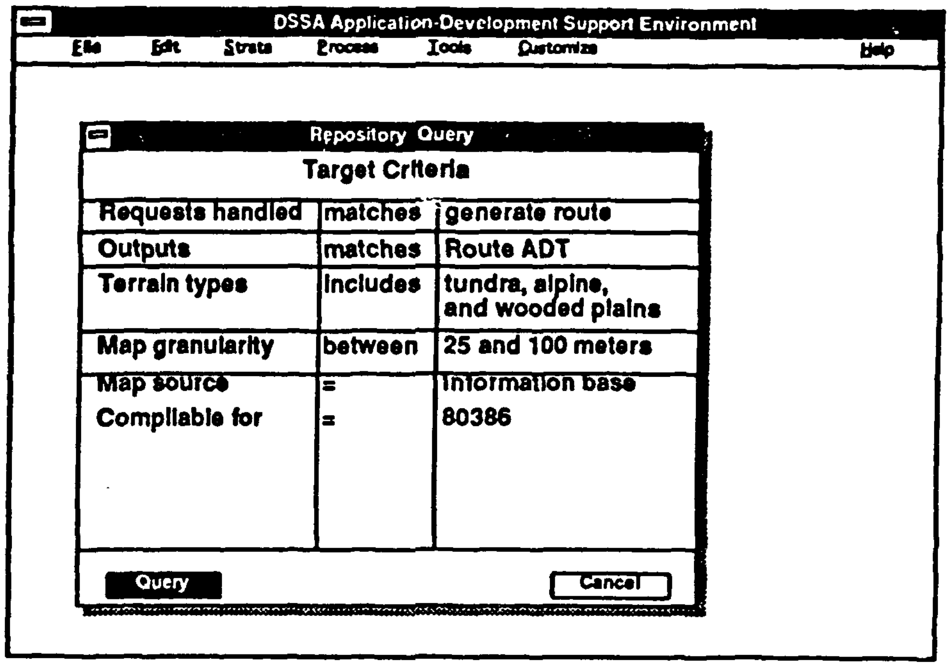
√ Linked to proper physical component.

Figure 6-13 Script Provides Some Design Cleanup Actions


DSSA Application-Development Support Environment
File Edit Strata Process Tools Customize Help
To-Do Tool
Not started
In progress
Completed
Details
Advice
Done

Not started	In progress	Completed	Tasks
		√	Check model consistency
	√		Design RSOP module
	√		Import Site Evaluation module
		√	Revise requirements
	√		Check requirement status
	√		Fix full evaluation time
		√	Add new processor
√			Recheck requirements
√			Add use of IR sensors
√			Find 386 version of Route Generation module

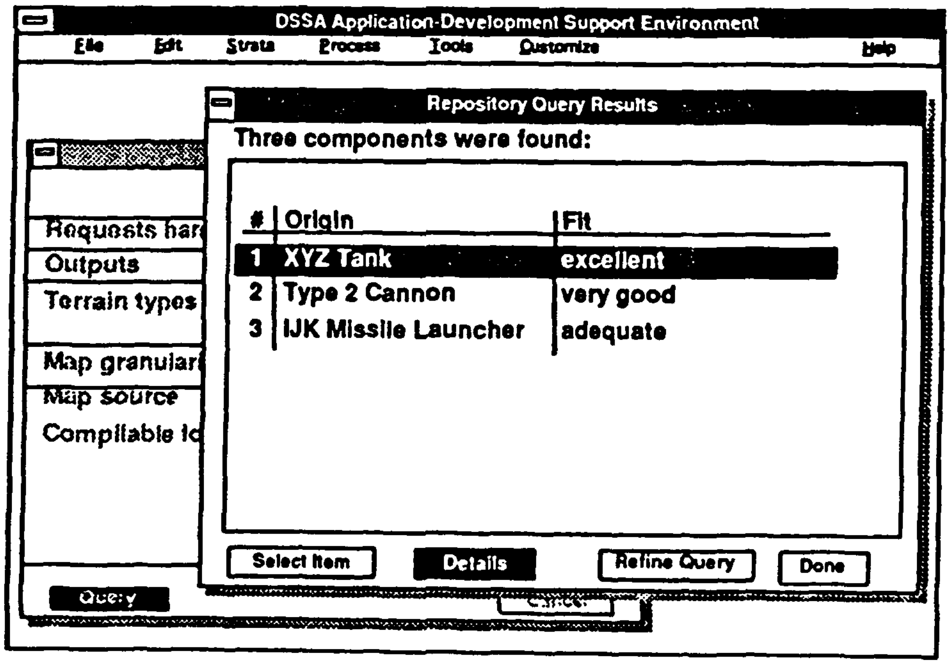
Figure 6-14 Results of a KBDA Reflected in the To-Do List


DSSA Application-Development Support Environment
File Edit Strata Process Tools Customize Help
Repository Query
Query
Cancel

Target Criteria

Requests handled	matches	generate route
Outputs	matches	Route ADT
Terrain types	includes	tundra, alpine, and wooded plains
Map granularity	between	25 and 100 meters
Map source	=	information base
Compilable for	=	80386

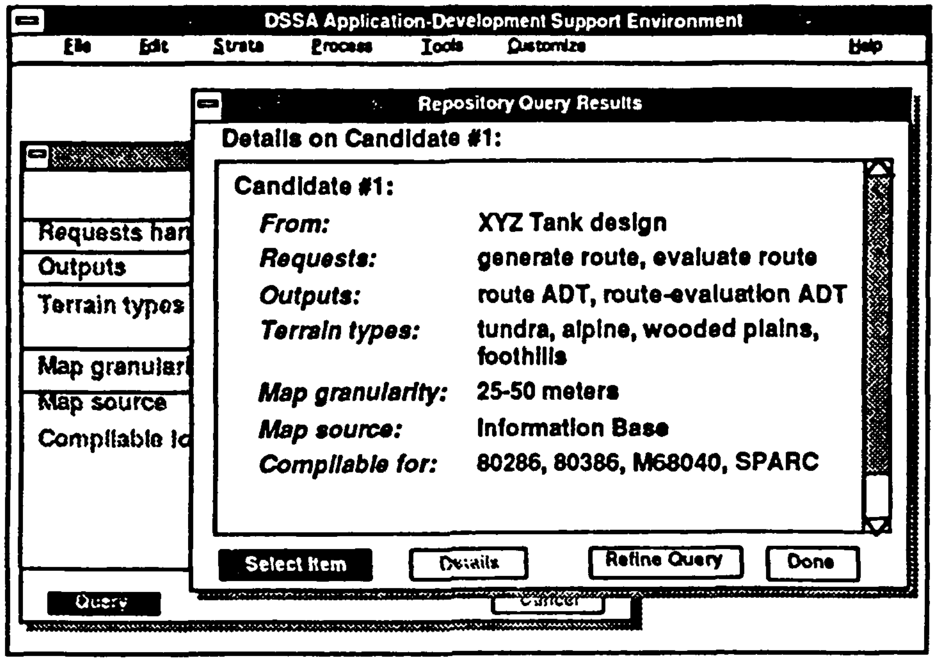
Figure 6-15 User Queries Repository for Possible Modules



Three components were found:

#	Origin	Fit
1	XYZ Tank	excellent
2	Type 2 Cannon	very good
3	IJK Missile Launcher	adequate

Figure 6-16 Summary of the Matches Found



Details on Candidate #1:

Candidate #1:

From:	XYZ Tank design
Requests:	generate route, evaluate route
Outputs:	route ADT, route-evaluation ADT
Terrain types:	tundra, alpine, wooded plains, foothills
Map granularity:	25-50 meters
Map source:	Information Base
Compilable for:	80286, 80386, M68040, SPARC

Figure 6-17 User Browses RSOP Requirements

The KBDA didn't find any "obvious" matches for the route generation module. The user decides in Figure 6-15 to query the repository for modules that might work from other vehicles. Figure 6-16 shows that the query found three hits and lists them along with its rating of their fit.

Finally, in Figure 6-17, the user looks at details of the top-rated candidate, which apparently looks excellent. The user selects it for inclusion in the current design and subsequently integrates this selection into the overall design web.

7 Mappings Between KE And SE In DICAM

Our research focuses on distributed intelligent control and management. Although our particular demonstration will initially be in the area of military vehicles, all of which share "move-and-shoot" domain models, we expect the technology and methodology will generalize to other intelligent control domains, such as factory management and control systems, which is a principal area in which we wish to leverage these results.

Intelligent control requires a blend of both SE and KE technology and methods. The traditional SE approach to control has been autonomous, real-time, closed-loop, and deterministic. The traditional KE approach has been toward human-in-the-loop, deliberative, non-real-time, problem-solving systems. This project requires integrating these two approaches to produce hybrid control which is deliberative and real-time, operates at different levels of granularity, and spans many levels of concern from high-level objective setting to low-level fast servo-controlled behaviors.

While the preceding paragraphs addressed the relationships between SE and KE *within* the DICAM applications, the two technologies must also be combined to support the *development process*. As previously discussed, we are using techniques and tools that originated in KE to support the incremental development process, including the formulation and development of DC and MC components, blackboard-like environment, knowledge-based advisors, and the domain and application models. On the other hand, we are using techniques and tools that originated in SE to support the tool integration, real-time systems specification, requirements management, domain analysis, and software engineering environments composed of myriad tools.

A key objective in this program is the reuse of software components. Given that the strategy for achieving this rests on the use of a reference architecture, generic module specifications, and domain-specific customizations, it appears that reuse is precisely the application of knowledge engineering to software engineering. That is, we seek to acquire and encode knowledge about software components that makes it possible to automate software engineering decisions.

8 Related Research

The DICAM-DSSA program integrates and extends research in many subareas of both SE and KE. We briefly mention some of the research related to our work in a few of the key areas.

Our efforts to develop a standard framework for hybrid control that combines conventional and AI approaches, uses a conceptually centralized information base and world model, and has a fractal or recursive, hierarchical structure synthesizes the following related results. Albus and his colleagues at NIST developed the NASREM hierarchical model of control which has multiple levels and a shared world model ([Albus 88], [Albus 89], [Albus 90]). The hierarchical, message-passing organization of robot control was developed by numerous investigators ([Becker 89], [Miller 91], [Schoppers 87], [Zeigler 90]). Our work is informed by conventional control theory and practice ([Bollinger 88], [Franklin 80]), AI approaches to intelligent control ([Boureau 89], [Collinot 90], [Erman 90], [Garvey 87], [Garvey 89], [Hayes-Roth 90a], [Hayes-Roth 90b], [Hayes-Roth 90c], [Hayes-Roth 90d], [Hayes-Roth 89], [Washington 89]), and various approaches to distributed control ([Erman 73], [Coleman 90], [Herget 90]).

Our work relies upon and extends much earlier work in distributed situation assessment ([Lesser 80], [Steeb 81]).

Intelligent control has been a focus of investigations in both manufacturing ([Murdock 90], [Pardee 90]) and planning ([Darwiche 89], [Fox 91], [Hayes-Roth 79], [Hayes-Roth 88], [Washington 90]), especially as used in mixed-initiative control systems such as the Pilot's Associate ([Smith 89], [Smith 88], [Telles 88]).

Our approach to developing software designs by finding good partial matches to requirements relies on previous work on these topics ([Fiksel 90], [Hayes-Roth 79], [Prieto-Díaz 91]).

Related work on design as assembly ([Bhansali 91], [Hayes-Roth 86]) is also highly relevant. These themes are central to the recent great interest in domain analysis and software reuse ([Brooks 87], [DARPA 90], [Prieto-Díaz 91]).

Software engineering environments ([Erman 88], [Hayes-Roth 89], [Hayes-Roth 91], [NIST 91]) are a major topic of interest that overlaps and informs our work. The specific requirements of real-time system specification and construction (cf. [Lark 90]) have also influenced our overall approach. The particular approach, that of incremental development, we are supporting here is motivated in part by its superiority over more rigid, waterfall approaches [Boehm 88].

Knowledge-based assistants for planning, design and engineering ([Bhansali 91], [Daube 89], [Fiksel 89], [Fox 91], [Frederick 90], [Nicklaus 88]) are a central element of our ADSE. Although software development advisory systems are still rare, the need for them and the conception of them is widely perceived ([Floyd 71], [Green 83], [Moriconi 79], [Waters 85]).

9 Conclusions

The DICAM-DSSA project we have described exploits and combines many facets and techniques of both SE and KE. By focusing on intelligent control applications, we are forced to combine and relate the largely complementary perspectives the two fields apply to control systems, namely, formal and algorithmic vs. rich and knowledge intensive. Furthermore, by focusing on the need to build systems more quickly and reliably, we have needed to apply KE techniques directly to the domain of SE. This is most apparent in the development of reusable domain and application models, the blackboard-like approach to the representation and control of the software development process itself, the classificatory organization to the reuse repository, and the provision of software development advisors (KBDAs). In order to address the requirements for building controllers for realistic, critical applications, we have needed to provide a SE methodology and environment that is somewhat traditional in its support for real-time systems engineering and the application of many, diverse software tools. To meet that requirement we have formulated an application development support environment (ADSE) capable of specifying and realizing practical controllers using both conventional and knowledge-based tools. Furthermore, the ADSE provides means for implementing varic s software development processes and assisting developers with such tools as a To-Do assistant and other KBDAs.

References

[Albus 88] Albus, J. S. "System description and design for multiple autonomous undersea vehicles", National Institute of Standards and Technology, Tech. Note 1251, 1988.

[Albus 89] Albus, J. S., McCain, H. G., and Lumia, R. "NASA/NBS standard reference model for telerobot control system architecture (NASREM)", National Bureau of Standards, Tech. Note 1235, 1989.

[Albus 90]	Albus, J. S. "Outline for a theory of intelligence", National Institute of Standards and Technology, 1990.
[Becker 89]	Becker, J. M. "The generic control level: a unifying view", *Proc. ROBEXS '89*, Palo Alto, CA, 1989.
[Bhansali 91]	Bhansali, S. and Nii, H. P. "KASE: An integrated environment for software design". Report No. KSL 91-73. Knowledge Systems Lab, Department of Computer Science, Stanford, 1991.
[Bollinger 88]	Bollinger, J. G. and Duffie, N. A. *Computer control of machines and processes.* Reading, MA.: Addison-Wesley, 1988.
[Boehm 88]	Boehm, B. "A spiral model of software development and enhancement", *Computer*, 21(5), May 1988, 61-72.
[Boureau 89]	Boureau, L. and Hayes-Roth, B. "Deriving priorities and deadlines in real-time knowledge-based systems", *Proc. of the IJCAI Workshop on Real-Time AI Systems*, Detroit, 1989.
[Brooks 87]	Brooks, F. "No silver bullet: Essence and accidents of software engineering", *Computer*, 20(4), April 1987, 10-19.
[Clancey 85]	Clancey, W. "Heuristic classification", *Artificial Intelligence*, 27(3), 1985, 289-350.
[Coleman 90]	Coleman, N. "An emulation/simulation environment for intelligent controls", in Herget, C. J. (ed.) *Proceedings of the Workshop on Software Tools for Distributed Intelligent Control Systems*, Lawrence Livermore National Laboratory report CONF-9007134, July 1990.
[Collinot 90]	Collinot, A. and Hayes-Roth, B. "Real-time control of reasoning: Experiments with two control models", *Proc. of the DARPA Knowledge-Based Planning Workshop*, San Diego, 1990.
[DARPA 90]	DARPA *Proceedings of the Workshop on Domain-Specific Software Architectures*, Hidden Valley, PA, July 1990.
[Darwiche 89]	Darwiche, A., Levitt, R. E., and Hayes-Roth, B. "OARPlan: Generating project plans in blackboard systems by reasoning about objects, actions, and resources", *Journal of Artificial Intelligence in Engineering Design, Automation, and Manufacturing*, 1989.
[Daube 89]	Daube, F. and Hayes-Roth, B. "A case-based redesign system in a mechanical domain", *Proc. of the International Joint Conference on Artificial Intelligence*, Detroit, 1989.

[Erman 90]	Erman, L. D. (ed.) "Intelligent real-time problems solving (IRTPS): DARPA/AFOSR/NSF Workshop report", Cimflex Teknowledge Corp., Tech Report TTR-ISE-90-101, 1990.
[Erman 73]	Erman, L. D., Fennell, R. D., Lesser, V. R., & Reddy, D. R. (1973). System organizations for speech understanding: Implications of network and multiprocessor computer architectures for AI. In *Proc. 3rd Inter. Joint Conf. on Artificial Intelligence*, (pp. 194-199). Reprinted in *IEEE Trans. Computers*, C-25 (1976), 414-421.
[Erman 80]	Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, R. "The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty", *Computing Surveys* 12(2), June 1980, 213-253.
[Erman 88]	Erman, L. D., Lark, J. S., and Hayes-Roth, F. "ABE: An environment for engineering intelligent systems", *IEEE Transactions on Software Engineering*, 14(12), December 1988.
[Fiksel 89]	Fiksel, J. and Hayes-Roth, F. (1989). "Knowledge systems for planning support". *IEEE Expert*, 4(3), 1989, 16-24.
[Fiksel 90]	Fiksel, J. and Hayes-Roth, F. "A requirements manager for concurrent engineering in printed circuit board design and production", *Proc. of the Second National Symposium on Concurrent Engineering*, Morgantown, WV, February 1990.
[Floyd 71]	Floyd, R. "Toward interactive design of correct programs". *IFIP*, Ljubljana, Yugoslavia, 1971, 7-11.
[Fox 91]	Fox, M. S. (1991). Knowledge-based logistics planning: its application in manufacturing and strategic planning (Final Technical Report No. RL-TR-91-241). Rome Laboratory, Air Force Systems Command.
[Franklin 80]	Franklin, G. F. and Powell, J. D. *Digital Control of Dynamic Systems*, Addison-Wesley, 1980.
[Frederick 90]	Frederick, D. K., James, J. R., Antoniotti, A., and Nitta, H. "A second-generation expert system for computer-aided control system design", DKF Consulting Service, Inc., Ballston Lake, NY, 1990.
[Garvey 87]	Garvey, A., Cornelius, C., and Hayes-Roth, B. "Computational costs versus benefits of control reasoning", *Proc. of the Annual Conference of the American Association for Artificial Intelligence*, Seattle, 1987.
[Garvey 89]	Garvey, A. and Hayes-Roth, B. "An empirical analysis of explicit vs. implicit control architectures", in Jagannathan, V. and Dodhiawala, R. T. (eds.), *Current Trends in Blackboard Systems*, Academic Press, 1989.

[Green 86] Green, C., Luckham, D., Balzer, R., Cheatham, T. and Rich, C. "Report on a Knowledge-based Software Assistant". Kestrel Institute, Palo Alto, CA., 1983. Reprinted in C. Ri[illegible]n & R. C. Waters (eds.), *Readings in Artificial Intelligence and Software Engineering,* Morgan Kaufmann, Los Altos, CA, 1986.

[Hayes-Roth 88] Hayes-Roth, B. "Blackboard architecture for control", *Artificial Intelligence,* 26, pp. 251-321, 1985. Reprinted in: Bond, A. and Gasser, L. (eds.), *Readings in Distributed Artificial Intelligence,* Morgan Kaufmann Publishers, Inc., 1988.

[Hayes-Roth 90a]Hayes-Roth, B, "Architectural foundations for real-time performance in intelligent agents", *Real-Time Systems: The International Journal of Time-Critical Computing,* 2(1/2), 1990, 99-125.

[Hayes-Roth 90b]Hayes-Roth, B. "Adaptive intelligent systems: Architecture and experimental applications", *Proc. of the DARPA Knowledge-Based Planning Workshop,* San Diego, 1990.

[Hayes-Roth 90c]Hayes-Roth, B. "Dynamic control planning in intelligent agents", *Proc. of the AAAI Symposium on Planning in Uncertain, Unpredictable, or Changing Environments,* Palo Alto, 1990.

[Hayes-Roth 90d]Hayes-Roth, B. "Making intelligent systems adaptive", In K. VanLehn (ed.), *Architectures for Intelligence.* Lawrence Erlbaum, 1990d.

[Hayes-Roth 79] Hayes-Roth, B. and Hayes-Roth, F. "A cognitive model of planning", *Cognitive Science,* 1979, 3, 275-310. Reprinted in A. Collins and E. E. Smith (eds.), *Readings in Cognitive Science: A Psychological and Artificial Intelligence Perspective.* Morgan Kaufmann, 1988; and in J. Allen, and J. Hendler, and A. Tate (eds.), *Readings in Planning,* Morgan Kaufmann, 1990.

[Hayes-Roth 79] Hayes-Roth, B., Hayes-Roth, F., Rosenschein, S., and Cammarata, S. "Modeling planning as an incremental, opportunistic process", *Proc. of the International Joint Conference on Artificial Intelligence,* 1979. Reprinted in R. Engelmore and A. Morgan (eds.), *Blackboard Systems.* Addison-Wesley, 1988.

[Hayes-Roth 86] Hayes-Roth, B., Johnson, M.V., Garvey, A., and Hewett, M. "Applications of BB1 to arrangement-assembly tasks", *Journal of Artificial Intelligence in Engineering,* 1986.

[Hayes-Roth 89] Hayes-Roth, B., Washington, R., Hewett, R., Hewett, M., and Seiver, A. "Intelligent monitoring and control", *Proc. of the International Joint Conference on Artificial Intelligence,* Detroit, 1989.

[Hayes-Roth 79] Hayes-Roth, F. The role of partial and best matches in knowledge systems. In D. A. Waterman & F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems* (pp. 557-574). New York: Academic Press, 1979.

[Hayes-Roth 91] Hayes-Roth, F., Davidson, J.E., Erman, L.D. and Lark, J.S. "Frameworks for developing intelligent systems: The ABE systems engineering environment", *IEEE Expert*, June 1991.

[Hayes-Roth 89] Hayes-Roth, F., Erman, L. D., Fouse, S., Lark, J. S., and Davidson, J. "ABE: A cooperative operating system and development environment", in M. Richer (ed.), *AI Tools and Techniques*, chapter 12. Ablex Publishing, Norwood, NJ, 1989. Reprinted in A. Bond and L. Gasser (eds.), *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, 1988, 457-488.

[Herget 90] Herget, C. J. (ed.) *Proceedings of the Workshop on Software Tools for Distributed Intelligent Control Systems*, Lawrence Livermore National Laboratory report CONF-9007134, July 1990.

[Lark 90] Lark, J. S., Erman, L. D., Forrest, S., Gostelow, K. P., Hayes-Roth, F. and Smith, D. M. "Concepts, methods, and languages for building timely intelligent systems". *Real-Time Systems: The International Journal of Time-Critical Computing,* 2(1/2), 127-148, May 1990.

[Lesser 80] Lesser, V. R., & Erman, L. D. "Distributed Interpretation: A model and experiment". *IEEE Trans. on Computers*, C-29(12), 1144-1163, 1980. Reprinted in A. Bond and L. Gasser (eds.), *Readings in Distributed Artificial Intelligence*, Morgan-Kaufman, 1988, 120-139.

[Miller 91] Miller, D. J. and Lennox, R. C. "An object-oriented environment for robot system architectures", *IEEE Control Systems*, February 1991, 14-23.

[Moriconi 79] Moriconi, M. "A designer/verifier's assistant". *IEEE Trans. on Software Engineering* 5 (4), July 1979, 387-401.

[Murdock 90] Murdock, J.L. and Hayes-Roth, B. "Intelligent monitoring and diagnosis of semiconductor manufacturing", *Proc. of the Fifth Annual SRC/DARPA CIM-IC Workshop*, Berkeley, 1990.

[Newell 91] Newell, A. *Unified theory of cognition.* Cambridge, MA: Harvard, 1991.

[Nicklaus 88] Nicklaus, D. J., Overton, K. S.,Tong, S. S., & Russo, C. J. Knowledge representation and technique for engineering design automation. In J. S. Kowalik & C. T. Kitzmiller (eds.), *Coupling symbolic and numerical computing in expert systems.* New York: Elsevier, 1988, 67 76.

[NIST 91] NIST ISEE Working Group and ECMA TC33 Task Group on the Reference Model. "Reference model for frameworks of software engineering environments". NIST Special Publication 500-201. Dec. 1991. Also published as Technic .l Report ECMA TR/55 (2nd ed.), European Computer Manufacturers Assn.

[Pardee 90] Pardee, W.J., Shaff, M.A., and Hayes-Roth, B. "Intelligent control of complex materials processes", *Artificial Intelligence in Engineering, Design, Analysis, and Manufacturing*, 4, 55-65, 1990.

[Prieto-Díaz 91] Prieto-Díaz, R. and Arango, G. (eds.). *Domain Analysis and Software Systems Modeling.* IEEE Computer Society Press, 1991.

[Schoppers 87] Schoppers, M. "Universal plans for reactive robots in unpredictable environments", *Proc. of the International Joint Conference on Artificial Intelligence*, Seattle, 1987.

[Smith 89] Smith, D. M. and Barnette, J.N. "Pilot's associate processing requirements". *AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference*, Seattle, WA, July 1989.

[Smith 88] Smith, D. M. and Broadwell, M. M. "The pilot's associate: an overview", *Proc. of the Eighth Annual Workshop on Expert Systems and their Applications.* Avignon, France, May 1988.

[Steeb 81] Steeb, R., Cammarata, S., Hayes-Roth, F., Thorndyke, P. W., and Wesson, R. B. Distributed intelligence for air fleet control. Technical Report R-2728-ARPA, The Rand Corporation, Santa Monica, CA, 1981. Reprinted in Bond, A. and Gasser, L. (eds.), *Readings in Distributed Artificial Intelligence*, Morgan-Kaufman, 1988, 90-101.

[Telles 88] Telles, D. and Wasson, J. "Rotorcraft pilot's associate", *Proc. of the American Helicopter Society Annual Forum*, Washington, D.C., June 1988.

[Washington 89] Washington, R. and Hayes-Roth, B. "Input data management in real-time AI systems", *Proc. of the International Joint Conference on Artificial Intelligence*, Detroit, 1989.

[Washington 90] Washington, R. and Hayes-Roth, B. "Abstraction planning in real time", *Proc. of the AAAI Symposium on Planning in Uncertain, Unpredictable, or Changing Environments*, Palo Alto, 1990.

[Waters 81] Waters, R.C. "The Programmer's Apprentice: A Session with KBEmacs". *IEEE Trans. on Software Engineering*, 11 (11), Nov. 1[illegible] 1296-1320.

[Wesson 88] Wesson, R., Hayes-Roth, F., Burge, J., Stasz, C., and Sunshine, C. "Network structures for distributed situation assessment". *IEEE Transactions on Systems, Man and Cybernetics January,* 1981. Reprinted in Bond, A. and Gasser, L. (editors), *Readings in Distributed Artificial Intelligence,* Morgan-Kaufman, 1988, 71-89.

[Zeigler 90] Zeigler, B.P. *Object- oriented simulation with hierarchical, modular modules, Intelligent agents and endomorphic systems.* San Diego: Academic Press, 1990.

Domain-Specific Software Architectures for Intelligent Guidance, Navigation, & Control

Ashok Agrawala
University of Maryland
Department of Computer Science
College Park, Maryland 20742
agrawala@cs.umd.edu

James Krause and Stephen Vestal
Honeywell Systems
Research Center
3660 Technology Drive
Minneapolis, Minnesota 55418
krause@src.honeywell.com
vestal@src.honeywell.com

Abstract: Honeywell and the University of Maryland are developing a domain-specific software architecture for intelligent (adaptive) guidance, navigation and control for aerospace applications. Some distinguishing aspects of our domain are a need to adapt to a variety of specialized target hardware systems, requirements for high reliability and system certification, and increasing demands for functional integration and high performance computing. Our approach exhibits three major themes: an extensive reliance on formal models, a provision of multiple views corresponding to multiple areas of skills and requirements, and an open toolset and layered architecture.[1]

1 Introduction

Honeywell and the University of Maryland are currently developing a software architecture for intelligent (adaptive) guidance, navigation and control as part of the DARPA DSSA program. In addition to providing a classification of operations and data types appropriate for our domain, our architecture will also provide standard interfaces (standard control and data flow mechanisms) that will facilitate component reuse.

An important aspect of our program is that the architecture itself will be amenable to automatic analysis, configuration and population. Thus, abstract and easily manipulable representations for our architecture, together with tools to analyze, configure and populate it, are important products of our program.

Another important aspect is that we must address multiple system requirements. Real-time performance, hardware requirements, and reliability must also be considered, not just GN

1. Supported by DARPA/ONR Contract No. N00014-91-C-0195.
©1992 IEEE. Reprinted, with permission, from *Proceedings of the 1992 IEEE Symposium on Computer Aided Control System Design*; Napa, California, March 17-19, 1992; pp. 110-116. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the IEEE copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Institute of Electrical and Electronics Engineers. To copy otherwise, or to republish, requires a fee and specific permission.

functionality. The software architecture must be based on technologies from multiple areas, just as engineers from many disciplines are involved in designing and building an actual control system.

We have identified three central themes that we believe will allow us to achieve our goals. First, we are making heavy use of formal models in the development of our architecture. Second, we are providing multiple views corresponding to different skills and requirements. Third, we are utilizing an open toolset in conjunction with a layered architecture.

1.1 Formal Models

The first major theme of our approach is a reliance on formal models to the maximum extent possible. Examples of formal models that play an important role in our program are systems of differential and difference equations, scheduling theory and combinatorial optimization, Markov processes, etc. Our approach is to derive the software architecture from formal models rather than attempt to construct approximate models for some convenient or pre-existing software architecture. One of the major research goals we are addressing is to do this in a way that satisfies two conditions.

First, a software architecture must represent an integration of multiple formal models. This is complicated by the need to configure an architecture. Neither formal models nor architectures apply to a single problem instance. Instead, they are both parameterized in a number of ways. For example, a control system might be parameterized by linear state space operators, number of processes and their frequencies, number of target system processors, hardware fault rates, etc. Parameters may be more complicated than individual values and may extend to structural aspects of a system, for example the interconnect or data flow pattern between a set of processes. When an architecture is configured according to the parameters and structures of one formal model, it is necessary that this be done in a way that preserves consistency with all the other applicable formal models and specifications.

Second, a software architecture should be as precisely modeled by each applicable formal model as possible. Our goal extends beyond the use of formal models as design guides. To the maximum extent possible, there should be a precise mapping between elements of the final software and elements of each formal model. For example, it should ideally be possible to map each machine instruction to the parameter of the real-time scheduling model that accounts for that instruction's execution time. Another way to describe this is that we are striving towards a methodology for formal verification in-the-large.

We anticipate a number of benefits from the use of formal models. In addition to architecture configurability and verification, the notation associated with a formal model provides a starting point for machine-processable specifications and representations. Because of the precise relationships between software architectures and formal models, the analytic predictions made using the formal models should be highly accurate. This will enable rapid design evaluation and trade-off studies and should lead to significant improvements in system quality.

Many formal models have associated optimization techniques that can be used to automatically synthesize good designs (e.g., synthesis and rate monotonic scheduling).

1.2 Multiple Views

The second major theme of our approach is to provide multiple views into an architecture that correspond to standard engineering disciplines, requirements categories, and groups of related tools and models. Actual systems are complex and require the skills of many disciplines, not just those traditionally associated with the field of controls. Computer system engineers, safety and reliability engineers, software engineers, as well as program managers and administrators, must all cooperate in order to produce an acceptable system. We can identify two areas in which we have goals that we believe will provide significant benefits.

One of our research goals in this area is to develop uniform user interfaces and specification and design languages within each view. Associated with each view are a variety of design synthesis, design analysis and evaluation, and code selection and generation tools. A common user interface with a common design and specification capture language would greatly facilitate more extensive design and analysis tasks within a view.

A second research goal is to facilitate the exchange of evaluation and trade-off information between views in a way that enables multi-disciplinary concurrent engineering and integrated product development. Analysis information obtained in one view may be of great interest in another view, especially when presented in a way that is meaningful and relevant in that view. For example, information about scheduling feasibility obtained by the computer system engineer, such as allowable changes in process frequencies and execution times, are also of interest to the control engineer. More elaborate capabilities than information sharing are desirable, such as allowing an engineer to "can" a system-specific analysis in some process programming or script language in a way that allows many design questions asked by specialists in other areas to be answered automatically.

1.3 Open, Layered Architecture

The third major theme of our approach is to provide an open toolset and a layered architecture. Generally equivalent tools should be substitutable for one another. The architecture should be layered in such a way that groups of tools correspond to architecture layers, where toolsets and layers can be added or removed for each particular project as needed.

Many vendors provide many tools to perform design synthesis and analysis, and hopefully many run-time modules will increasingly be drawn from software repositories. Rather than tie ourselves to a particular toolset, our goal is to facilitate the integration of new tools and modules into the overall product. In some cases, different sites will have a preference among roughly similar tools (e.g., MATLAB versus XMath versus HoneyX). In other cases, internal and/or highly specialized tools may be required for certain markets or projects (e.g., multi-body dynamics and flexible structures).

Domains, like systems, are hierarchical. Guidance, navigation and feedback control can be viewed as subdomains within the overall field of control science, just as control implementation can be viewed as a subdomain within the overall field of embedded computing. One of our major goals is to develop a toolset and architecture that is extensible to related domains. In particular, it should be possible to add new views, toolsets, and layers to our IGN architecture (e.g., display management, signal and image processing, and diagnostics).

2 Architecture Views

We will first walk through a simple design scenario to help better explain our concept of view before outlining the set of views we are assembling. Figure 2-1gives an example of two views we are working to provide into our domain-specific software architecture: guidance, navigation and control; and resource scheduling and allocation. We begin our scenario with a control engineer using a GN view to develop a control algorithm.

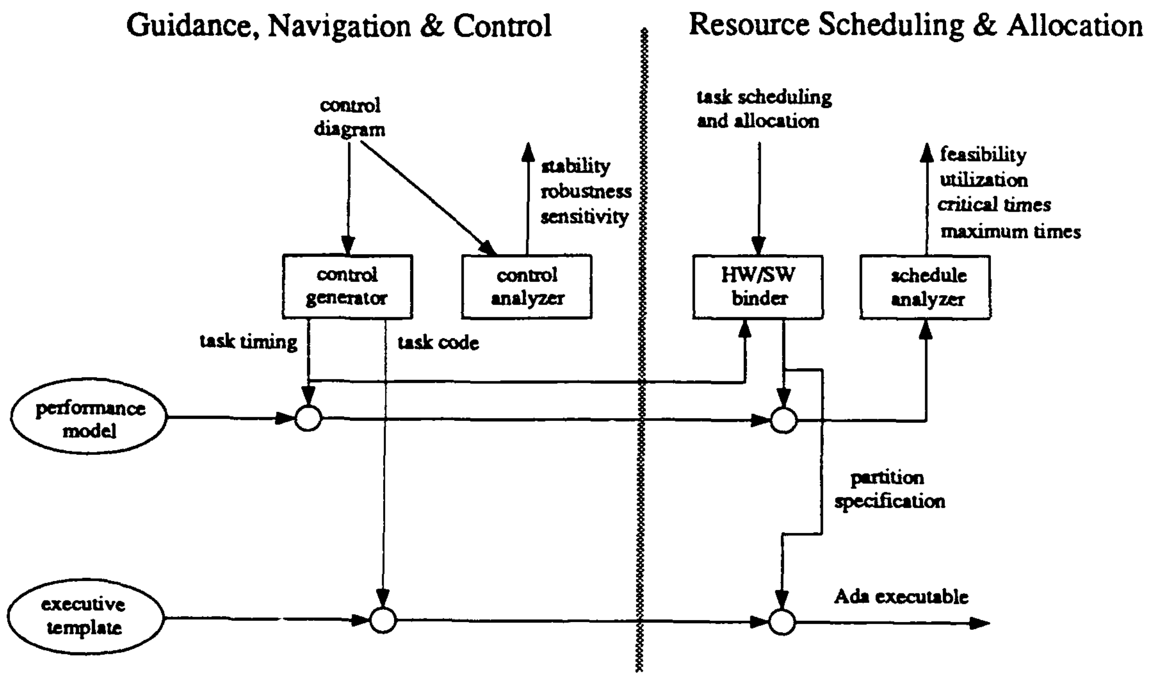


Figure 2-1 Example Architecture Views

The control engineer develops a plant model and controller design expressed in some convenient specification language, most likely a mixed visual/textual language. Synthesis, analysis, and simulation tools are available to help develop and evaluate the plant model and controller design. Stability, performance, and robustness criteria are typical concerns at this point in the development process. When the control engineer is satisfied with the preliminary controller des gn, toc are available to generate software to implement the control functions and to gener timing and data flow specifications.

The machine representation of an overall software architecture may be viewed as a set of logically distinct configurable specifications or templates, ignoring for the moment the specific representation and consistency maintenance techniques used. In the example of Figure 2-1, the timing data is used to configure a performance model, and the data flow specifications and software are used to configure and populate a run-time executive template. These then become available in the resource scheduling and allocation view for use by the computer system engineer.

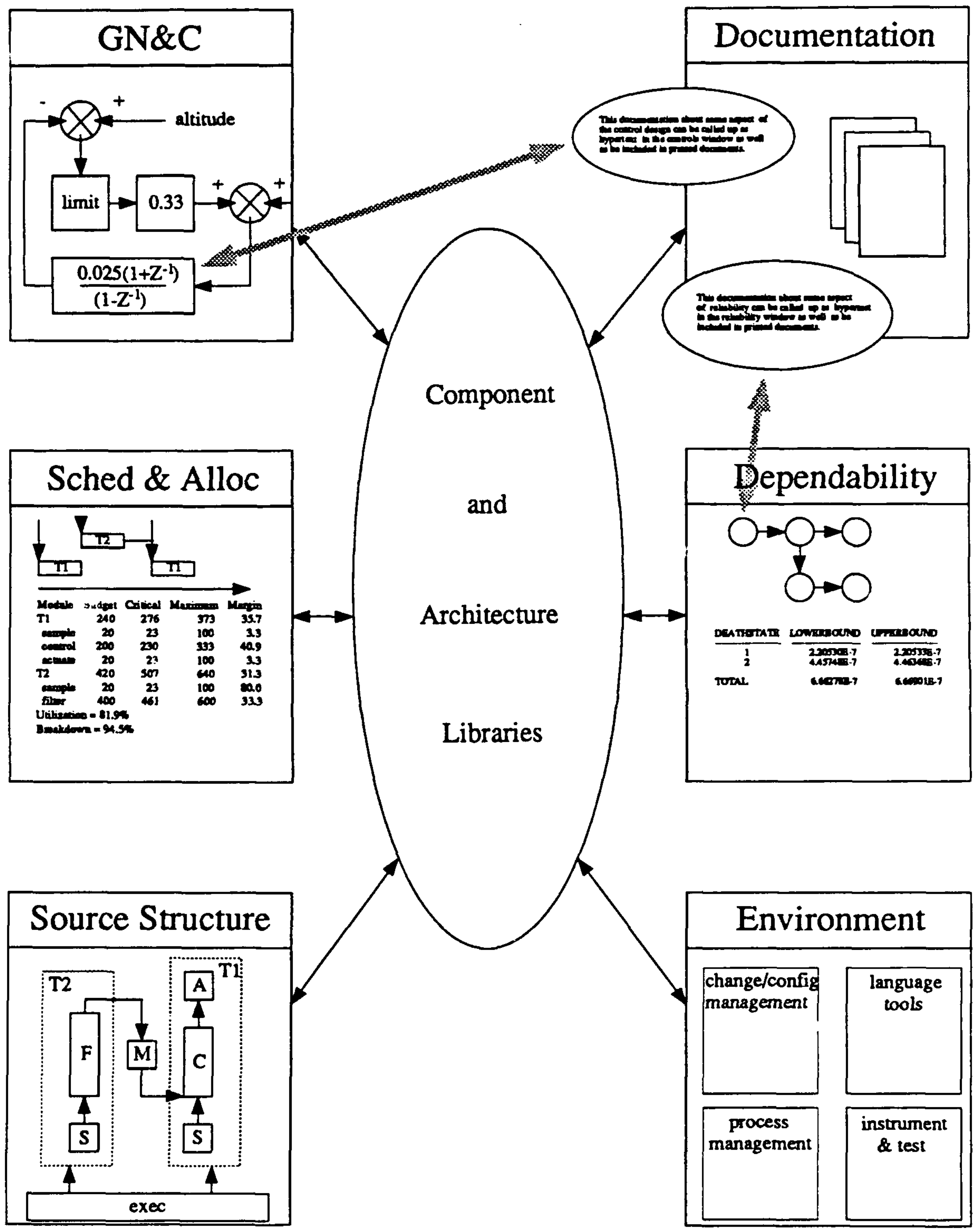
The computer system engineer employs a variety of scheduling and combinatorial optimization tools. For example, simulated annealing might be used to allocate processes to processors and data flows to hardware interconnects, and rate monotonic scheduling might be used to schedule the processes assigned to each particular processor. Analysis tools provide an indication of scheduling feasibility and processor utilization. Analysis tools can also provide useful sensitivity analysis information, such as how much a process frequency or execution time can be changed and still preserve feasibility (or must be changed to achieve feasibility). This latter information is also of great interest to the control engineer, and when relayed back to the GN view provides a basis for rapid, multi-disciplinary design iteration.

Figure 2-2 shows six views that are useful in the development of IGN software. Each of these views consists of an appropriate specification and design capture interface together with appropriate synthesis, analysis, and generation tools. All these tools access a common architecture and component library, which in the final product may be some mixture of integrated workspaces, persistent storage managers, representation-to-representation translators, and consistency maintenance tools. The first two of these views have already been introduced.

The source structure view will allow the software engineer to examine and compose the software at the source (e.g., Ada) level, providing such standard capabilities as cross-referencing, graphical display of data and control flows, etc. These capabilities are characteristic of existing Computer Aided Software Engineering (CASE) toolsets.

What we plan to provide in our documentation view is management of hypertext/hypermedia capabilities that are pervasively provided throughout the interfaces for all views. For example, it should be possible for the control engineer to call up a hypertext card to document a particular transform in a particular diagram, or the software engineer to call up a hypertext card to document a particular data flow in a particular package. This view should also contain capabilities to search, extract and format information in various ways (e.g., 2167A documents).

Dependability includes not only reliability in the face of hardware faults (i.e., classical hardware fault-tolerance), but also notions of security, testing, certification, plant model validation, and any other verification validation issues. Like documentation, dependability provides a centralized view of activities that may occur in other views. For example, different forms of testing will be conducted in the GN, the resource allocation and scheduling, and the source structure views, where all testing might be tracked and traced to various requirements within this view.


GN&C
altitude
limit
0.33
0.025(1+Z-1) / (1-Z-1)
Documentation
This documentation about some aspect of the control design can be called up as hypertext in the controls window as well as be included in printed documents.
This documentation about some aspect of reliability can be called up as hypertext in the reliability window as well as be included in printed documents.
Component and Architecture Libraries
Sched & Alloc
T1
T2
Dependability
Source Structure
T2
T1
A
F
M
C
S
exec
Environment
change/config management
language tools
process management
instrument & test

Module	Budget	Critical	Maximum	Margin
T1	240	276	373	35.7
sample	20	23	100	3.3
control	200	230	333	40.9
actuate	20	23	100	3.3
T2	420	507	640	31.3
sample	20	23	100	80.0
filter	400	461	600	33.3

Utilization = 81.9%
Breakdown = 94.5%

DEATHSTATE	LOWERBOUND	UPPERBOUND
1	2.205308E-7	2.205338E-7
2	4.457448E-7	4.463468E-7
TOTAL	6.662788E-7	6.669018E-7

Figure 2-2 Architecture Views

The overall product must be usable in a variety of development environments, targeted to a variety of embedded computers. Such characteristics as language processing tools, software repositories, change and configuration management, and process management procedures and practices, will inevitably vary from site to site. We include this view to emphasize its importance within an overall development toolset and the need to flexibly adapt to varying environments.

There are substantial numbers of pre-existing tools in all these views. The goal of our project is not to reimplement this or that tool, but rather to develop a software architecture for IGN that integrates all these aspects. In some sense, the details of tool selection and tool infrastructure will be driven by the software architecture, just as architecture development will be driven by the various formal models and requirements.

3 Architecture Layers

Figure 3-1 shows some proposed layers in our IGN architecture. In our system, a layer carries the traditional connotation of a block of software that may but need not be present in the final system. We also wish this term to connote the associated views and tools used to specify, analyze, and generate that layer of application software.

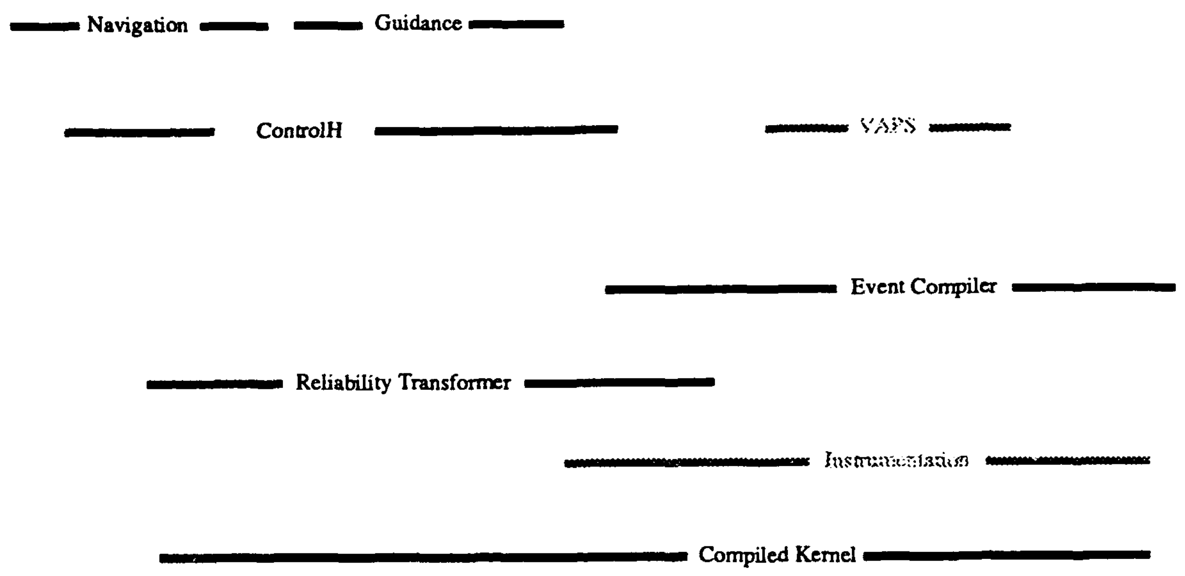


Figure 3-1 Architecture Layers

3.1 Guidance, Nav ControlH

Our initial approach is based on a common specification and design language called ControlH. ControlH provides all the basic operations and operators required for traditional control system specification, as well as features required for adaptive and nonlinear control. As illustrated in Figure 3-2, this language is intended to provide a common interface for both synthesis and analysis, simulation, and code generation tools. The preliminary language design calls for mixed visual and textual entry, and in greatest generality ControlH could be termed a user interface rather than a language in the traditional sense.

We plan to build the subdomains of navigation and guidance on top of the basic ControlH capabilities. Our current approach is to make the ControlH layer extensible. Navigation and guidance will be provided by integrating specialized capture, synthesis, and analysis tools and specialized run-time module implementations into the basic ControlH toolset and module library. This will inevitably impact the ControlH interface to some extent, and one of the research issues we must deal with is defining ControlH in a way that allows extensions to be made in a regular manner without propagating major changes throughout the language (e.g., new menu options, new operator attributes, and new predefined operators and operand types).

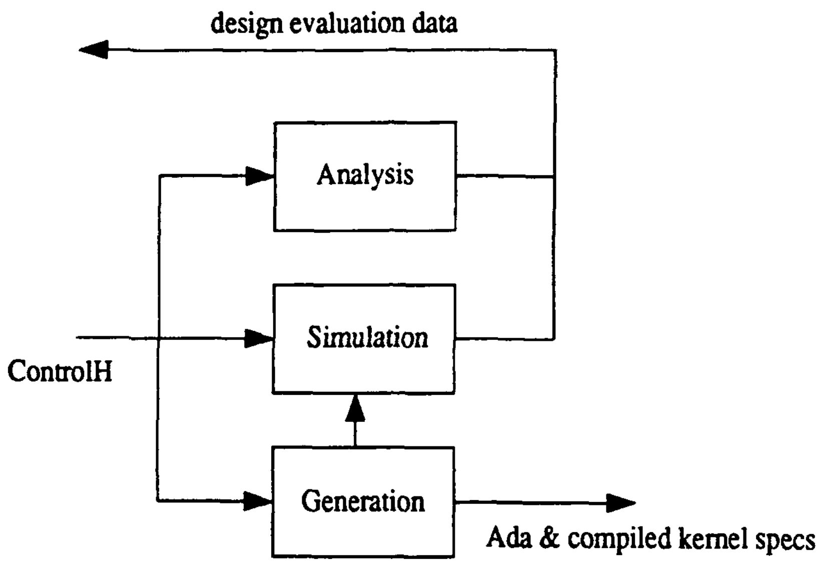


Figure 3-2 ControlH Specification and Interface

Note that simulation uses the same source modules generated for the final system. That is, the simulator supports the same functional component interfaces as the real-time, reliable kernel. There are two important consequences of this. First, the simulations utilize the final operational flight software for the control functions, eliminating an important source of inconsistency and error in current practice. Second, the plant model (or portions thereof) can be used to generate code that is executed in real-time, facilitating hybrid testing of the actual digital control system.

3.2 Kernel, Reliability Events

The lowest level of our architecture is a compiled kernel. This layer provides secure, hard real-time, fault-tolerant scheduling and communication for periodic and aperiodic processes in multi-processor systems. The associated toolset supports a domain-specific module interconnect language. That is, users identify source modules to be included in a system and specify how those modules are to be scheduled, how they communicate, and how they are bound to various hardware resources in the target. The toolset performs hard real-time schedule feasibility and sensitivity analysis; timing, data, and control security verification; reliability analysis; and then automatically assembles the modules into the final system, generating all necessary scheduling and communication code automatically.

Statically generating the kernel code provides two major advantages. First, it eliminates the need for many run-time services that must otherwise be provided by an executive (e.g., process creation and message routing). This can lead to significant improvements in efficiency as well as reductions in the amount of flight software that must be certified. Second, it facilitates a static analysis and verification of kernel behavior.

The reliability analysis performed by the compiled kernel toolset is based in part on a specification of the redundancy management approach used. There are several viable approaches for redundancy management (e.g., error masking and reconfiguration) and several alternative consensus protocols to select from (e.g., triple modular redundancy and Byzantine protocols). We plan to provide a tool to help transform a nonredundant specification into a redundant one and assist in performing the necessary trade-offs.

Discrete Event Dynamic Systems (DEDS) design and analysis is becoming increasingly important in modern control systems. There are a number of models for DEDS, each of which serves a different purpose. The lower layers of our system are primarily concerned with the para-functional requirements of reliability and performance, and we are basing our initial event processing and analysis layer on queueing theory and queueing network models.

Eventually, all real-time scheduling and communication, discrete event and mode change processing, and hardware redundancy management are likely to be provided using a largely uniform user interface, analogous to the way we hope to obtain navigation and guidance capabilities as extensions to ControlH. In the short term, this is complicated by the need to use specialized forms of specification for queueing systems and redundancy management schemes. An active area of research for us will be to better integrate reliability, real-time resource allocation and scheduling, and discrete event specification and analysis, within a common conceptual framework and user interface.

3.3 Domain Extension

One of our goals is to provide as extensible a product as possible. This may be accomplished through the addition of new analysis tools and source modules within a view, as we observed for ControlH. This may also be accomplished by the addition of entirely new layers and views. Figure 3-1 shows two such layers that do not fall cleanly within our domain of IGN but for which existing tools can be easily integrated with our DSSA product.

The Virtual Application Prototyping System (VAPS) is a commercial tool used to rapidly prototype cockpit display systems. The VAPS code generator is being modified to produce operational Ada flight software, and such a tool is easily integrated into the overall DSSA system.

Like many organizations, we have an instrumentation system to support real-time testing, debugging and performance measurement in multi-processor targets. As with the layers and views of our DSSA product, instrumentation is specified using a high-level experimentation language that is compiled into the necessary event and data monitoring code. There are also data collection and analysis tools. Such tools can provide powerful development support when integrated into a DSSA product.

4 Conclusion

Within our domain there are a number of generally accepted formal models, as well as a number of analysis tools, for GN real-time scheduling and resource allocation, event processing, and hardware fault-tolerance and reliability. What has been lacking is a common software architecture that supports requirements in all these fields, is subject to accurate analysis, can be automatically configured and populated, and serves as a core around which a complete integrated development system for IGN software can be built.

Domain-Specific Software Architectures for Hybrid Control

Richard Platek and James H. Taylor
ORA Corp ıc'ion
301 A Dates D, /e
Ithaca, NY 14850-1313
richard@oracorp.com
jim@oracorp.com

Abstract: The ORA-Cornell University team in the DARPA DSSA project is concerned with the domain of hierarchical, distributed, intelligent, hybrid control for nonlinear systems. By "hybrid" we mean that the resulting system is comprised of both digital and analog elements. The motivation for this focus is as follows: Software for linear control is a fairly mature field with several commercial CAD environments in use containing facilities for automatic generation of code. These environments can be viewed as moderately successful instances of DSSA. The vendors do provide some capability for designing, analyzing, and implementing nonlinear controls, but this is generally *ad hoc* and rudimentary in form. Our team is developing both a mathematical theory of nonlinear hybrid control and corresponding prototype software tools which support, where possible, the automatic generation of control laws and real-time software implementations. This theory is being tested on real physical control problems, and a CAD environment is being developed to encapsulate the theory and make it available to control engineers. The basis of the environment is a flexible package to simulate, analyze and visualize hybrid systems behavior in terms of dynamical systems with discontinuous vector fields. CAD tools for design will be integrated with this package.[1]

1 The Domain

As part of the DARPA Domain-Specific Software Architecture (DSSA) program a team consisting of ORA Corporation and Cornell University's Mathematics Sciences Institute (MSI) is investigating an environment for the design, implementation and evaluation of hierarchical, distributed, intelligent, hybrid control. The effort is being monitored by Dr.Norman Coleman's laboratory at the U.S. Army's Armament Research, Development and Engineering Center (ARDEC).

The present stress in the effort is on *hybrid* control, by which we mean an integrated approach to continuous physical devices (mechanical, electrical, hydraulic, etc.) being controlled by discrete computational units (digital cpus). The customary approach to digital control of physical

1. The work described in this article was partially funded as a part of the DARPA-sponsored DSSA Project under Contract No. DAAA21-92-C-0013, for US Army AMCCOM, Picatinny Arsenal, NJ 07806-5000.
©1992 IEEE. Reprinted, with permission, from *Proceedings of the 1992 IEEE Symposium on Computer Aided Con' ' System Design*; Napa, California, March 17-19, 1992; p. 137. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the IEEE copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Institute of Electrical and Electronics Engineers. To copy otherwise, or to republish, requires a fee and specific permission.

processes is to either take the world's point of view or the computer's point of view. In the former, the control problem is solved mathematically using continuous math and the solutions are implemented using a digital computer to calculate approximations to the continuous controls. These approximations are generally close enough for practical purposes. In the latter case, the continuous world is replaced by a sampled world anu the problem is viewed entirely as an exercise in discrete math. In *hybrid* control, one attempts to study the problem without either reducing the discrete to the continuous or the continuous to the discrete.

2 The Approach

At the present time the mathematical theory does not exist to analyze or design hybrid control problems rigorously. Most textbooks in control theory present parallel chapters on continuous and discrete control with, for example, the Laplace transform in the former being replaced by the z transform in the latter. On the other hand, one can claim that in practice control engineers deal with hybrid control all the time. It is not unusual in the history of control theory that practice leads the development of theory. The flyball governor was introduced in the early nineteenth century as a piece of effective technology. The theory behind it was first explored in 1868 by James Clerk Maxwell in his paper "On Governors", which established the field of mathematical control theory [Maxwell 68]. Similarly, a mathematical theory of hybrid control which deals with mixed logico-differential equations should establish a rationale for what practitioners dealing with the digital control of continuous processes have learned to do through experiment and tuning.

The development of the mathematics of hybrid control under this effort is proceding at Cornell and ORA. The Nerode-Kohn paper at the recent IEEE Symposium on Computer-Aided Control System Design reports on some findings [Nerode-Kohn 92]; other work under this effort is reported in Nerode's "Hybrid Systems: A Survey and Models" [Nerode 92] and Yakhnis' "A Concurrency Games Approach To Hybrid Systems". In the latter paper, a game-theoretic approach to hybrid control is outlined which will support automated extraction of winning strategies (i.e., effective control laws) from game formulations of the control problem.

Another innovative approach underpinning our effort is the use of nonlinear dynamical systems theory to support the design and analysis of hybrid controls. To support this, we are working to adapt simulation tools from this area toi solve hybrid control problems. In the last two decades, the mathematics of dynamical systems has made significant progress in both the worlds of pure and applied science [Guckenheimer 83]. Under the provocative name of "chaos" this mathematics has caught the popular imagination and its vocabularly is entering common educated discourse [Stewart 89]. In addition, the problem of controlling chaos is attracting wide scale attention in the scientific world ([Ott 90], [Bradley 92]).

In our project we are adapting and further enhancing the package *dstool* which has been developed at Cornell under the direction of John Guckenheimer and his students. This tool is available by Internet ftp from Cornell and we are presently changing it so that it can be used to analyze and simulate hybrid dynamical systems. Principal changes are the integration of trajectories defined by discontinuous vector fields and the extension of algorithms for determining fixed points, bifurcations, and the onset of chaotic behavior. This is done by adopting a manifold-descriptive approach to phase space; that is, the phase space is covered by patches with coordinate transformations when patches overlap.

Finally, there have been recent advances in the area of frequency-domain methods for the effective *synthesis* of nonlinear controllers that hold substantial promise as a means of automatically generating control algorithms and software for nonlinear hybrid control ([O'Donnell 91], [Taylor 90]). In addition, we have recently conceived of an extension to this approach that can be used as a way to synthesize fuzzy controllers. These approaches will be integrated with the ORA hybrid control environment to assess their usefulness as DSSA generators.

In order to focus our early efforts we are applying our emerging nonlinear control and hybrid system theories to the ARDEC test fixture called the Advanced Testbed 1000 (ATB1000) \cite{mattice}. This is a physical simulation (scale model) of a gun with a flexible barrel which can be aimed and is subject to base motion and recoil disturbances. The control problem is to point the tip of the barrel using actuators attached to the barrel's base. Motion pictures of the cannon on the Apache helicopter show that during firing the tip of the cannon appears to behave quite chaotically. Since it is the tip which determines the targeting of the gun the Apache is known as an "area weapon" as opposed to a point weapon. The ability to control its firing would greatly enhance the military value of the Apache. The ARDEC test fixture is designed to physically simulate situations like the Apache helicopter.

3 Conclusion

The creation of methods and software tools for the automatic generation of instances of DSSAs for nonlinear hybrid control problems holds great potential for reducing the cost of real-time embedded control software for practical applications. The work under the DSSA Program in general and this contract in particular should help to realize this promise.

References

[Bradley 92] Elizabeth Bradley. "Mathematics and Computation: Controlling Chaos". *Second International Symposium on Artificial Intelligence and Mathematics (Autonomous Control Workshop)*, 1992.

[Guckenheimer 83]John Guckenheimer and Philip Holmes. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Springer-Verlag, New York, Berlin, Heidelberg, Tokyo, 1983.

[Mattice 91] Michael Mattice. "Inertia Wheel Fixture (Problem)", *1991 Automatic Control Conference*, pages 2992-2993, 1991.

[Maxwell 68] J.C. Maxwell. "On Governors", *Proc. Royal Soc. London*, volume 16, pages 270-283, 1868.

[Nerode 92] Anil Nerode. "Hybrid Systems: A Survey and Models", *Second International Symposium on Artificial Intelligence and Mathematics (Autonomous Control Workshop)*, 1992.

[Nerode-Kohn 92]Anil Nerode and Wolf Kohn. "An Autonomous Systems Control Theory", *1992 IEEE Symposium on Computer-Aided Control System Design*, 1992.

[O'Donnell 91] James R. O'Donnell, Jr. and James H. Taylor. "CAE Tools for Nonlinear Systems Analysis and Design Based on SIDFs", *IFAC/IMACS Symposium on CAD in Control Systems*, pages 69-74, 1991.

[Ott 90] Edward Ott, Celso Grebogi, and James A. Yorke. "Controlling Chaos", *Chaos: Proceedings of a Soviet-American Conference (American Instutute of Physics)*, 1990.

[Stewart 89] Ian Stewart. *Does God Play Dice? The Mathematics of Chaos*, Basil Blackwell Inc, 1989.

[Taylor 90] James H. Taylor and James R. O'Donnell, Jr. "Synthesis of Nonlinear Controllers with Rate Feedback Via SIDF Methods", *1990 Automatic Control Conference*, pages 2217-2222, 1990.

[Yakhnis 92] Alexander Yakhnis. "A Concurrency Games Approach to Hybrid Systems, *Second International Symposium on Artificial Intelligence and Mathematics (Autonomous Control Workshop)*, 1992.

Application of ProtoTech Technology to the DSSA Program

Frank C. Belz
TRW Systems Integration Group
TRW R2/2062
One Space Park
Redondo Beach, CA 90278
belz@trwarcadia.sod.trw.com

David C. Luckham
Stanford University
ERL 456
Stanford CA 94305
luckham@anna.stanford.edu

James M. Purtilo
University of Maryland
Department of Computer Science
College Park, Maryland 20742
purtilo@cs.umd.edu

Abstract: In DARPA's ProtoTech Program, exploration of the prototyping process and innovative technologies to support that process has led to an emphasis on (i) multilingual prototyping, (ii) component-based architectures, (iii) architecture evolution, and (iv) rapid derivation of prototypes from pre-existing components and specifications of behavior and connectivity requirements. These emphases match needs within the DSSA program; consequently a team of ProtoTech participants (TRW, Stanford, and Maryland) is focused on bringing early maturing technology from the Prototyping Program into the DSSA program. For example, it is unifying the technology (developed at Stanford for modelling, prototyping, and analyzing prototypes of distributed, time-sensitive systems), with the Polylith module interconnection technology (developed at Maryland for building multi-lingual distributed applications out of pre-existing components). This DSSA team is using such ProtoTech technologies as RAPIDE and Polylith to model and prototype architectures under development by the other, domain-based DSSA teams. Using the same modelling techniques on all these domains, we are seeking to identify commonalities between the DSSA teams' products. Such commonality can increase the results from the DSSA program by suggesting the development of common technology to be used by most or all teams where appropriate. This process of identifying commonalities is interwoven with the process of transferring the ProtoTech technology into use by the other DSSA teams.[1]

1 Introduction

Prototyping has an essential role in the development of domain-specific software architectures (DSSAs) and in the new reuse-based software development processes and technology that will result from their use. Successful prototyping enables rapid evaluation of hypothesized architectures and proposed implementations. This is important during the initial formulation of

1. This work is supported by the Defense Advanced Research Projects Agency and the Office of Naval Research under Contract N00014-91-J-0173.

domain-specific architectures, and even more important in the evolution of those architectures and components satisfying them. Prototyping technology in common use today, however, is unlikely to be sufficient for the new demands of DSSA-based development. The ProtoTech Program is developing prototyping technology expected to live up to those demands, among others, and thereby to contribute to the full realization of the DSSA Program goals.

The ProtoTech Program is a collaborative effort involving several teams of industrial and academic participants, described elsewhere in these proceedings. The ProtoTech teams have been cooperatively exploring alternate hypotheses regarding both the nature of prototyping processes and the kind of technological innovation that can most effectively improve our ability to conduct prototyping processes. These investigations have led to an emphasis in the ProtoTech community on multilingual prototyping, component-based architectures, evolution of architectures, and rapid derivation of executable versions of prototypes both from pre-existing components and from specifications of behavior and connectivity requirements. These concerns correspond, point by point, with needs within any DSSA-based approach to software development.

Some of the ProtoTech investigations are already capable of improving the prototyping practice, others will begin bearing fruit in the near future, and still others are likely to achieve their first real impact in a few years. While several participants in the ProtoTech program are also members of domain-based DSSA teams, the DSSA team consisting of three members of the CPL/CPS community, TRW, Stanford and the University of Maryland, is attempting to bring selected early-maturing ProtoTech technologies to bear in multiple DSSA team efforts and, through the process of interacting with those teams, to identify commonalities among them that can be exploited in the DSSA program.

2 Our Research Hypothesis

The DSSA program is based upon the observations that (i) distinct software applications can have common architectures, (ii) such common architectures can enable efficient reuse of components across such applications, and (iii) such common architectures are most easily recognized in specific application domains, in part because the body of widely understood concepts for a particular problem domain helps to overcome substantial differences in the representations of the applications.

We hypothesize that the kinds of generalizations that enable the identification of common architectures within a problem domain can be applied to identify (sub)architectures that are common to applications in several domains.

We are testing that hypothesis by employing technology from the ProtoTech program to represent architectures and to analyze them. Specifically, the Prototyping Language and System [Belz 90], developed by the Stanford/TRW team and the Polylith Software Bus [Purtilo 92], a module interconnection system developed by the University of Maryland, emphasize the evolutionary development of application architectures as part of the process of prototyping applications. These are the principal technologies from the ProtoTech program that we are using to:

- identify common architectural components among the architectures of the domain-focussed DSSA contractors,
- define modular components, separating interfaces from (possibly multiple) implementations,
- specify component interfaces, including where appropriate specification of functional behavior, concurrent behavior consisting of synchronization and intercommunication, and timing requirements,
- specify the topology of distributed dataflow and control of communication between components,
- identify generic components from which different components of separate domain architectures can be derived by well defined development paths,
- deliver a specialized version of an architecture description language and system (derived from RAPIDE and Polylith) for modelling and evaluating DSSA architectures,
- show how this architecture description system can serve as a Module Interconnection Facility sufficient to support the interoperability requirements of the DSSA architectures, i.e., provide an ability to define architectures whose components are implemented in multiple languages (such as Ada, C++ and VHDL),
- show how this technology could be incorporated in tools developed under the DSSA program for eventual inclusion in software development environments,
- provide feedback to the ProtoTech Program to ensure that its technology is applicable to the DSSA Program.

If we successfully demonstrate our hypothesis, we expect the consequences to include an improved ability within the DSSA program to generate architecture-based application components and tools, with a reduced risk that the program will generate several isolated domains, each with incompatible tools to support rapid application development. The latter risk is currently the state of practice with fourth generation technology and its amelioration is a significant, if implicit, goal of the DSSA program.

3 The Enabling Technology

Among the ProtoTech technologies, our work will rely extensively on the Prototyping Language and Toolset, the Polylith module interconnection technology, and the results of ProtoTech's Module Interconnect Formalism Working Group (MIF WG) efforts.

The RAPIDE Prototyping Language and Toolset

RAPIDE is a prototyping language with supporting tools that provide features for specifying and building prototypes of distributed time-sensitive systems. Many of the concepts embodied in have evolved from prior work at Stanford ([Augustin 90], [Luckham 87], [Luckham 90]).

Two versions of RAPIDE have been defined: RAPIDE 0.2, a preliminary proof-of-principal version, with tools; and RAPIDE 1.0, a full prototyping language, designed as a language framework which can be tailored to meet domain-specific requirements as needed. RAPIDE has the following features:

- **Module Specifications**: Module interface specifications contain:
  - general properties of module operations, independent of any implementation details, so that varying implementations are possible. These properties include definition of values returned by module functions, timing behavior of module operations, and exception propagation situations.
    These specifications are equally suitable for software modules and for hardware modules. They permit a variety of currently separate techniques to be used in specifying DSSA's, including input/output specifications, finite state machine transitions, and interval timing specifications.
  - Requirements needed from the using environment in order for a module to operate effectively; these include generic parameter restrictions, and other reuse requirements.
    These specifications, which deal primarily with reuse and reconfiguration of modules, provide critical data for a variety of tools supporting configuration of architectures and reuse of components.
- **Communication Architectures**: Communication architectures define how the modules in a system communicate, what kinds of data, timing and data throughputs are allowed, and the synchronization or independence of operation between modules. In general, communication architectures can be dynamic, or variable during the system operation.
- **Design Hierarchy**: RAPIDE 1.0 provides powerful and simple features based on object-oriented inheritance [Mitchell 91], and mapping constructs to capture design hierarchy. Mappings are a new language concept, and are based on the concept of patterns of events. This feature is used to capture design decisions (i.e., designer knowledge) involving hierarchical refinement of components. Using pattern mappings it is possible to specify in RAPIDE how a component module, which has been specified abstractly at one level in a system design, is expanded into a detailed architecture at a lower design level. Design hierarchy is a critical and often used tool in architecture specification. One such example is the refinement of a communication system into layers of protocols, each layer being more and more detailed and hardware-specific.
- **Executable Prototypes**: allows prototype systems to be implemented in a simple concurrent rule-based language, or RAPIDE 1.0) in other languages such as Ada or VHDL. Implementations of both modules and communication architectures are supported. The primary purpose is two-fold: (i) to build executable models rapidly for analysis of requirements, and to support gradual evolution of systems from either early prototypes or one application to another. RAPIDE allows implementations of components to be introduced into the system one component at a time, and tested in the overall system design framework containing prototypes or implementations of all other modules.

- **Derivation Histories**: The RAPIDE support system will keep a history of how various versions of module components and architectures are built out of previous versions. This facility will be based on an object-oriented inheritance feature of the language. It is a powerful facility in reconfiguring systems to meet new requirements while at the same time ensuring against various kinds of errors.
- **Tools for Analytical Prototyping**: The present RAPIDE system is based on the model of distributed computations as partially ordered sets of events (Posets) [Meldal 91]. The ordering between events expresses whether an event caused another event or whether two events occurred independently. Timing of events by means of timestamps is also included in RAPIDE computations. According to modern theory of concurrent computation, the Poset model provides the most accurate information about behavior of a distributed system, and has demonstrable advantages over the use of linear traces. The RAPIDE toolsuite provides graphical capabilities for manipulating and viewing partially ordered and timed computations, and also tools for automatically checking specifications against such computations. Other toois are currently being designed and built to support: (i) analysis of architecture specifications using subtype and inheritance information,
(ii) reconfiguration of architectures to meet changes in requirements, including, for example, selection of modules for reconfiguration of an architecture, (iii) test data generation from specifications to support the use of DSSA's to test specific applications, (iv) transition from RAPIDE systems to specific Ada implementations.
- **Processes of Evolutionary System Development:** The RAPIDE 1.0 language and support system is designed to support processes of gradual evolution of systems. The type system captures the development paths of both components and architectures. It provides a basis for checking some aspects of the correctness of such reuse activities as component replacements. The runtime analysis tools based on Posets with timing provide a capability to analyze changes in communication, synchronization, and timing, and compatibility with requirements. The design hierarchy features allow low level detailed implementations to be compared for consistency with high level specifications.
The sum total of these capabilities is a powerful support system for developing rigorous processes of system evolution and refinement based on behavioral prototyping and analysis in conjunction with formal specifications.

Status of Prototyping in RAPIDE

The capabilities described above are supported by an experimental prototype toolset. Experiments demonstrating the ability to prototype in RAPIDE a wide variety of small example systems from differing domains have been carried out. Examples of small systems include a communication protocol, a telephone PBX, a disk controller, a satellite communication scheduler, and simple hardware devices including a 16-bit CPU, and examples drawn from other DSSA projects: a helicopter on-board flight program from the IBM avionics-domain effort, and the controller/meta-controller model from the CIMFLEX/Teknowledge effort. The architectures of some of these examples involve three or four levels of hierarchical refinement. Development of a set of guidelines for evolutionary prototyping is underway.

Polylith

Our planned experiments entail comparing and evaluating software that has been developed for specific domains by several contractors. Success in these experiments requires that we have a practical way to integrate such software. This requirement directly corresponds to one from the ProtoTech program, where the focus is on rapid configuration of prototyping apparatus. Available software components must be easy to use, without regard for their implementation language or original execution platform.

The Polylith software interconnection system at the University of Maryland helps meet these requirements for ProtoTech ([Callahan 91], [Purtilo 88], [Purtilo 91a], [Purtilo 91b], [Purtilo 92]). Polylith provides:

1. a module interconnection language (MIL) for developers to express their software configurations abstractly.
2. a packaging system for analyzing configurations and then generating all stubs, build commands and other interfacing structures according to the developer's abstract interfacing decisions.
3. a run-time system providing various forms of communication support for a configuration's components, especially those from diverse languages and platforms. This run-time system is referred to as an implementation of the 'software bus', described below.

A version of Polylith has been distributed for use within the ProtoTech community, and is now being used as the carrier for many of the community's ideas. The version of Polylith that has resulted from evolution within ProtoTech will form a new baseline for use in our DSSA activities.

Whether for prototyping or for experimenting with heterogeneous domain-specific software, the critical problem within interconnection is clear: current ad-hoc methods for interconnecting software forces programs to become intricately coupled with their environment. The components of a realistic application are not cohesive, since they must serve many functional and non-functional requirements at the same time. Examples of non-functional requirements that apply to application components are coercion of data representation and relocation of the data by available media. These requirements accumulate primarily due to limitations in the interconnection system, such as the mixing of host platforms or implementation languages. This coupling limits reuse of components within prototyping environments, and would impede DSSA experiments with components having diverse origins.

To solve this problem, we postulated a new software organization to encapsulate communication and data transformation tasks. This is the software bus organization, where an 'abstract bus' is the specification of interfacing properties to be encapsulated, and a 'run-time bus' is a program that implements the developer's encapsulated decisions. This allows developers to decouple the implementation of interfacing requirements from the treatment of functional requirements. Thus programmers are able to code without having to pay constant attention to constraints imposed by the underlying architectures, language processors or communication media. Such constraints cannot be completely ignored, but our primary result is that they can be isolated for independent treatment without loss of performance at run time. Moreover, once the application has been built for one execution environment, then tailoring the interface needs for execution in other environments is a separate and automated activity.

Describing the direct relation between arbitrary software components is difficult. Describing how an arbitrary component relates to the abstract bus is easier, and, once this is done, developers have a common basis for relating that component to others that have been similarly specified. The Polylith system demonstrates our software bus organization (currently on Unix platforms), and continues to evolve as the ProtoTech Module Interconnection Formalism (MIF) Working Group converges towards specification of a 'prototyping bus' suitable for use within the community. For DSSA, we will adopt the software bus organization to serve both pragmatic interconnection needs and abstract module comparison activities.

The Module Interconnection Formalism Working Group

The ProtoTech program achieves a substantial interaction through technology-specific working groups. Among the most important to DSSA is the Module Interconnection Formalism Working Group (MIF WG). This Working Group is charged with developing and formalizing emerging community consensus on desirable models and technologies for module interconnection of incrementally-developed systems of the future.

The MIF WG must address a very broad set of services, requirements, and scenarios. Examples of these issues are value passing mechanisms, inter-language integration, inter-machine integration, transmission of values having abstract types, pointer data, stream data, synchronization, generalized exception handling, implementation of call-back mechanisms, implicit versus explicit invocation, and performance considerations.

The need for a component-based approach is clear, and there appears to be a general consensus within the MIF WG that a high-level notation for describing the components and their interconnections is needed. There is also consensus that formalisms are needed that support definition of interconnection service layer, the "prototyping bus". In some cases, the Polylith interconnection system has been used to prototype these capabilities to enhance the discussion.

Our approach, using the Polylith software interconnection system in union with is entirely compatible with the consensus emerging from the MIF working group.

4 Melding Research and Technology Transfer

Our effort is both a research and a technology transfer task. We are interweaving both aspects of our effort, making each work on behalf of the other.

Achieving technology transfer is a complex process, involving management of many organizational, interpersonal, and technical factors [Przybylinski 91]. Our approach emphasizes the commitment process, which we merge with our experimental method. The commitment process, as described in Przybylinski, Fowler, and Maher, has three phases: Information Transition (contact, awareness, understanding), Pilot Test (trial use), and Technology Transition (adoption, institutionalization). Correspondingly, we have a three-phase model of technology transfer. For each domain-based DSSA team we will go through these three phases; we will overlap our interaction with the teams over time.

Phase 1: "In-house" investigations. In this phase we determine the applicability of the ProtoTech technology to the needs of the domain-based project. Our team exerts the effort involved; we develop executable architecture descriptions that serve as illustrative prototypes addressing architectural issues the domain-based team is grappling with. A precondition for this investigation is a prior identification on the part of the domain-based team that there is some potential applicability of the technology. Expected positive postconditions include recognition by the domain-based team that the technology is in fact applicable, and worth investing in further.

Phase 2: Collaborative investigation. In this phase we demonstrate that the technology is applicable, and begin the adaptation of the domain-team process to include the technology. The heart of the activity is pilot architecture prototype development, in which both our team and the domain-based team share the effort of using the prototyping technology in the pilot. Preconditions include satisfactory completion of Phase 1, management commitment on the domain-based team, available resources, and appropriate timing. Expected positive postconditions include increased commitment to the technology by the domain-based team, improved understanding by our team of the commonality of the pilot architecture with those of other teams, and useful feedback to the ProtoTech community about the requirements on prototyping technology imposed by DSSA-based development approaches.

Phase 3: Client usage. The purpose of this phase is to integrate the use of the technology in the domain-based team's process. The activity is selected and conducted solely by the domain team. This phase requires mature, robust technology, successful Phase 2 investigation, and management commitment. Expected post-conditions include improved acceptance, understanding and feedback.

The principal research objective, to identify important commonalities among the approaches of the separate domain-based teams, is well served by the three-phase model. Phase 1 creates intimate involvement with the team, their architectural approach and artifacts. It also starts the process of identifying common elements of the teams' approaches by using a common representation technology for the designs and architectures from the domain-teams. By phase 3, all parties have a very clear pictures of the architectures, the technology we have provided to represent that architecture, and the commonalities among the architectures.

References

[Augustin 90] L. Augustin, D.C. Luckham, B. Gennart, Y. Huh and A. Stanculescu, *Hardware Design and Simulation in VAL / VHDL.* Kluwer Academic Publishers, October 1990, 322 pages.

[Belz 90] F.C. Belz and D.C. Luckham, A New Approach to Prototyping Ada-Based Hardware/Software Systems, in *Proceedings of the ACM Tri-Ada Conference*, Baltimore. December 1990.

[Callahan 91] J. Callahan and J. Purtilo, *A Packaging System for Heterogeneous Execution Environments*, IEEE Transactions on Software Engineering, June 1991.

[Luckham 87] D.C. Luckham, F.W. von Henke, B. Krieg-Brückner and O. Owe, *ANNA — A Language for Annotating Ada Programs*, Springer-Verlag Lecture Notes in Computer Science, LCNS No. 260, July 1987, pp. 1-140.

[Luckham 90] D.C. Luckham, *Programming with Specifications. An Introduction to Anna, A Language for Specifying Ada Programs.* Springer-Verlag Texts and Monographs in Computer Science Series. October 1990, pp. 1-421.

[Meldal 91] S. Meldal, S. Sankar and J. Vera, Exploiting Locality in Maintaining Potential Causality, in *Proceedings of the 1991 Symposium on Principles and Practice of Parallel Programs (PPoPP).*

[Mettala 92] E.K. Mettala, J.R. James, N.P. Coleman, E.J. Gallagher, Jr., R.L. Harris and J.G. Smith, Domain Specific Software Architectures: Government Needs and Expectations, in *Proceedings of the Symposium on Computer-Aided Control System Design*, Napa, California, March 1992.

[Mitchell 91] J.C. Mitchell, S. Meldal and N. Madhav, An Extension of Standard ML Modules with Subtyping and Inheritance, in *Proceedings of Conference on Principles of Programming Languages*, 1991.

[Przybylinski 91] S.M. Przybylinski, P.J. Fowler and J.H. Maher, Software Technology Transition, Tutorial presented at the 13th International Conference on Software Engineering, Austin, TX, May 12, 1991.

[Purtilo 88] J. Purtilo, D. Reed and D. Grunwald, Environments for Prototyping Parallel Algorithms, *Journal of Parallel and Distributed Computing*, vol. 5, pp. 421-437, 1988.

[Purtilo 91a] J. Purtilo and P. Jalote, An Environment for Developing Fault Tolerant Software, *IEEE Transactions on Software Engineering*, Vol. 17, February 1991, pp. 153-159.

[Purtilo 91b] J. Purtilo and P. Jalote, An Environment for Prototyping Distributed Applications, *Computer Languages*, Vol. 16, no. 3/4, 1991, pp. 197-207.

[Purlilo 92] J. Purtilo, *The Polylith Software Bus,* to appear in *ACM Transactions on Programming Languages and Systems.* Currently available as University of Maryland TR-2469.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
4. PERFORMING ORGANIZATION REPORT NUMBER(S CMU/SEI-92-SR-9	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute — 6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office
6c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213	7b. ADDRESS (City, State and ZIP Code) ESD/AVS Hanscom Air Force Base, MA 01731
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office — 8b. OFFICE SYMBOL (if applicable) ESD/AVS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003
8c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO 63756E; PROJECT NO. N/A; TASK NO N/A; WORK UNIT NO. N/A

11. TITLE (Include Security Classification)
LTC Erik Mettala and Marc H. Graham, eds.

12. PERSONAL AUTHOR(S)

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM TO	June 1992	85 pp.

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse of necessary and identify by block number)

Avionics Software
Command and Control Software
Control Software
Domain-Specific Software Development
Software Architecture
Software Reuse

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The DARPA Domain Specific Software Architecture Program (DSSA) is a five-year effort that has been active since July 1991. This document contains an overview of the work being done in the program as of July 1992.

Software architectures serve as frameworks for software reuse. Domain-specific software architectures also serve as a common language in which domain engineers can discuss, understand and teacn the principles of their craft.

There are six independent projects within the DSSA program. Four of these projects are working in specific,

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED SAME AS RPTDTIC USERS ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution
22a. NAME OF RESPONSIBLE INDIVIDUAL John S. Herman, Capt, USAF	22b. TELEPHONE NUMBER (Include Area Code) (412) 268-7631 — 22c. OFFICE SYMBOL ESD/AVS (SEI)

ABSTRACT —continued from page one, block 19

military-significant domains. Those domains are Avionics Navigation, Guidance and Flight Director for Helicopters; Command and Control; Distributed Intelligent Control and Management for Vehicle Management; Intelligent Guidance, Navigation and Control for Missiles. In addition, there are two projects working on underlying support technology. Hybrid (discrete and continuous, non-linear) Control and Prototyping Technology.

This report contains brief descriptions from each project and an overview.

END