

DARTS: A DOMAIN ARCHITECTURE FOR REUSE IN TRAINING SYSTEMS

Robert G. Crispen, Brett W. Freemon, K. C. King, and William V. Tucker
Boeing Defense & Space Group
Huntsville, Alabama

ABSTRACT

The dynamics involved in the training system marketplace of today are dictating the need for major changes in the way organizations specify, develop, and maintain training systems. One of the key areas affected by these changes is the system and software architecture of training systems. This is evidenced by the increased attention that has been placed on architectures by recent initiatives (e.g. Structural Model, Mod Sim, STARS, DIS, ARPA DSSA, etc.). There are many reasons for this emphasis, not the least of which is a desire to produce training systems at the least possible cost while providing faster time to market and higher quality. An architecture for training systems can be a framework to enable cost reduction, reusability, and standardization.

We derive a set of attributes which we believe characterize a "good" software architecture. We discuss an architecture developed by Boeing Defense & Space Group, the Domain Architecture for Reuse in Training Systems (DARTS) and evaluate DARTS against these criteria. We also discuss the role of DARTS in megaprogramming, part of the ARPA STARS initiative, and suggest that DARTS is a suitable architecture for achieving the STARS vision of process-driven reuse.

ABOUT THE AUTHORS

Robert G. Crispen is a Systems Analyst with the Missiles & Space Division of the Boeing Defense & Space Group. He worked on the Modular Simulator Program, the Ada Simulation Validation Program, and is currently a researcher in an R&D program at Boeing. Before joining Boeing, he was a Senior Systems Design Engineer on commercial flight simulators at GMI in Tulsa, OK. He holds a Bachelor of Arts degree from the Johns Hopkins University in Liberal Arts/Psychology.

Brett W. Freemon is a Senior Software Engineer with the Missiles & Space Division of the Boeing Defense & Space Group. He has worked on all areas of the life cycle of simulation and training systems from proposal through delivery and installation. He is currently working on a research and development program centering on software reuse technology. He holds a Bachelor of Science degree from the Georgia Institute of Technology in Applied Physics.

K. C. King is manager of the ARPA STARS demonstration with the Boeing Defense & Space Group. This demonstration entails using STARS and megaprogramming technologies to develop an operational flight instrument trainer for the Navy T-34C aircraft. Prior to joining the STARS program, he developed architectures for a number of large-scale DoD information systems. He holds a Bachelor of Arts degree from the University of Michigan in Political Science.

William V. Tucker is the R&D Manager for Boeing Defense & Space Group, Simulation and Training Systems organization. He has managed various trainer programs, including US, UK, and RSAF E-3, KC-135 and the Modular Simulator Design Program. He is a member of a functional oversight committee for the ARPA/Navy T-34 FIT STARS demonstration project. He holds a Bachelor of Science degree from Wichita State University in Electrical Engineering.

DARTS: A DOMAIN ARCHITECTURE FOR REUSE IN TRAINING SYSTEMS

Robert G. Crispin, Brett W. Freemon, K. C. King, and William V. Tucker
Boeing Defense & Space Group
Huntsville, Alabama

WHAT IS AN ARCHITECTURE?

An architecture, as we intend to use the term, consists of (a) a partitioning strategy and (b) a coordination strategy. The partitioning strategy leads to dividing the entire system into discrete, non-overlapping parts or components. The coordination strategy leads to explicitly defined interfaces between those parts.

These two strategies provide an engineering approach to bridging the gap between the system as a whole (as represented by its specification) and the design (the plan to build the product from primitive parts, such as computer instructions, metal struts, and switches).

The reach of an architecture can extend from a single system (an architecture that solves a unique product problem) to an entire family or product line of systems. In the latter case, once the partitioning methods and coordination rules are determined, multiple products can be generated using the same methods and rules.

By the definition we are offering, the Software Engineering Institute's (SEI's) Air Vehicle Structural Model (AVSM)^{1,2} and the HAVE MODULE Modular Simulator (Mod Sim)³ fit into our discussion of architectures.

Because the architecture will determine both the list of parts for a particular product and the coordination between those parts (their size and shape, among other things) the architecture chosen for a particular training system has a decisive impact on reuse. We have observed⁴ that software parts which were developed under one architecture were adapted only with great difficulty to serve in another architecture. When the architectures are significantly different, we have concluded that redevelopment is more cost-effective than re-coding.⁵

WHAT IS A "GOOD" ARCHITECTURE

If architectures are different from one another, it ought to be possible to say that one architecture is better or worse than another in some meaningful sense. Nevertheless, we have seen very little in either the training systems literature or the software engineering literature on what makes one architecture better than another. There are assertions that one architecture or another is a "good thing," but there is little public scrutiny of criteria.

We will offer the following characteristics against which software architectures can be measured. Since we are about to describe a specific software architecture, the Domain Architecture for Reuse in Training Systems (DARTS), these criteria may be unconsciously biased toward DARTS. However, we have attempted to establish a widely acceptable set of criteria.

A good architecture can be leveraged. It must show promise of lasting beyond present programs, rather than being a quick fix of specific current problems. It must be adaptable to easily fit many development methods. It must promote the highest levels of reuse maturity. It must hold up to changing requirements. And it must be scalable across a significant portion of the training systems domain.

A good architecture promotes system understanding. It must "look like" the problem space in some significant sense. It must be clear, and it must clearly meet both user and end customer requirements. Its quality and style should match what are considered sound systems and software engineering principles.

A good architecture is rational. It should promote and support a repeatable and improvable process for building out a specific member of the family.

A good architecture is affordable. It must be "efficient enough" in both time and memory. It must support large-scale cost and schedule improvements in both the short term and the long term. And it must have been defined, published, and demonstrated to work in order to reduce risk.

A good architecture is a good citizen. It should not violate company or customer standards. It should be broadly accepted or acceptable in the training systems and customer community. It should be available in the public domain rather than being bound to a proprietary hardware or software system. It should meet the emerging framework criteria articulated by the ARPA DSSA project. And it should take advantage of military and international standards like the Ada programming language and ISO communications protocols.

HISTORY OF DARTS

Boeing Defense & Space Group had participated in the Ada Simulation Validation Program (ASVP) where the term "structural model" was first introduced to industry. We had also participated in Mod Sim³ from the beginning of the program through our role as the prime contractor for the demonstration/validation phase. When we had a preview of structural modeling at the SEI, we were anxious to reconcile the two, if that were possible. Fortunately, ASVP provided a common starting point.

With help from the SEI, we were able to realize what it was that we needed to develop: an architecture which captured both the reusable form of a structural model and the reusable content of Mod Sim.

The resulting software architecture, DARTS, was developed as the domain-specific software architecture (DSSA) for the Air Vehicle Training System (AVTS) used by the Navy/STARS 1993 demonstration of the benefits of megaprogramming.

Characteristics Adapted From Mod Sim:

Several features of the Mod Sim architecture are incorporated in DARTS:

- DARTS is based on the notion of a generic flight simulator that is capable of being adapted into any present or foreseeable training simulator. The generic simulator is partitioned into approximately 125 air vehicle Functions or areas of capability.
- Each of these Functions is assigned to one of twelve Segments (see Figure 1).
- A Segment is characterized as being coherent internally and loosely coupled externally. That is, the Functions assigned to a Segment "go together" in the sense that there are data flow, execution order, or other dependencies between them. Func-

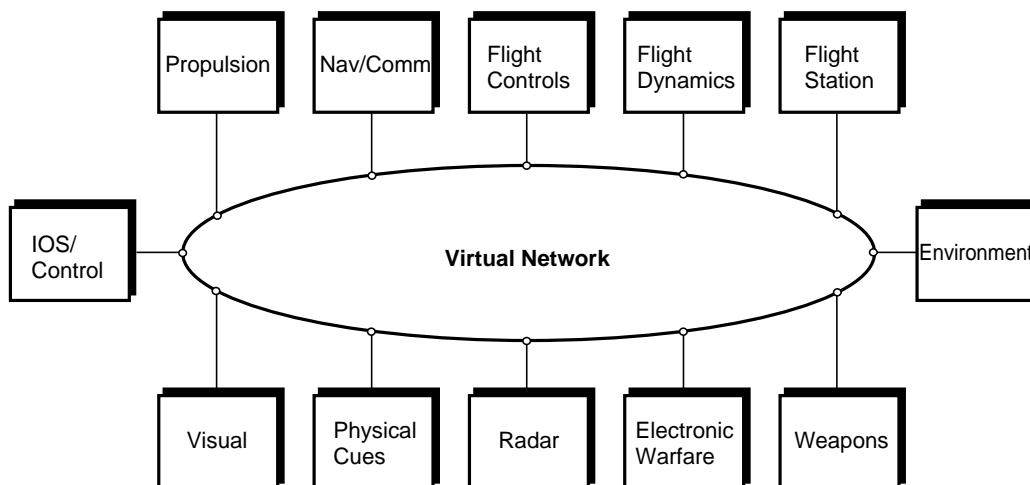


Figure 1 DARTS Segments

tions assigned to different Segments, on the other hand, do not "go together" in this sense and may execute independently of Functions in other Segments, except that they may produce data for or consume data from those Functions.

► Segments have a clearly defined set of interfaces with one another. The generic interface definitions, which are maintained as compilable Ada code and which are adaptable to the requirements of a specific training system, define the only means by which Segments may communicate with one another.

To summarize the first four points, a sizable block of systems engineering work was done on Mod Sim and its follow-ons. This work was subjected to an industry-wide review process and evaluated via a demonstration project. There is a defined process for adapting this work to virtually all kinds of simulators.

In our conversations with the Software Productivity Consortium (SPC) and STARS, it became clear that this reusable systems engineering and its work products are very similar to the processes we now call Domain Engineering⁶.

► DARTS retains the message-based communication method between Segments pioneered on Mod Sim. Message-based communications have a certain safety factor which is absent in shared-memory architectures. Imagine the following scenario: variable *x* is computed by a Segment running in another CPU. Your Segment executes the following code:

```
Y := Shared_Memory.X;  
Do_Something_Else;  
Z := Shared_Memory.X;  
if (Y /= Z) then  
    Strike_Pilot_Repeatedly_  
    With_Control_Column;  
end if;
```

In a shared-memory architecture, if the other Segment changes variable *x* while this Segment is `Do[ing]_Something_Else`, the pilot might become unhappy! In a message-based architecture like DARTS, on the other hand, variables are only updated when the application program requests that

they be updated.

Message-based communication is not the only safe mechanism for avoiding the situation above. The SEI's AVSM, for example, has a data synchronization mechanism at the start of each frame which accomplishes the same thing. Nevertheless, shared memory often ties the builder of a training system to one or a handful of vendors of shared memory hardware. We believe message-based communication is a more general solution. We also understand that achieving universal agreement on this point is unlikely.

Mod Sim Characteristics Discarded in DARTS

A few features of the original Mod Sim architecture imposed unnecessary restrictions on implementations, and were replaced in DARTS:

► Basic Mod Sim divided a trainer into twelve separate boxes, called Modules. Because DARTS ought to work on *any* computer system or combination of computer systems, we discarded the notion of one Segment per box (or Module). Instead, we distinguished between Segments (closely coupled software systems) and Modules (computational systems) so that any number of Segments could reside in a Module. The current version of the Mod Sim specifications have adopted this convention as well.

► Mod Sim Segments communicated with one another over a fiber-optic network. This capability is still available for communication between Segments which reside in different Modules, but for Segments in the same Module, it makes sense to communicate through shared memory.

The wrong way to do this is to require individual Segments to know where they are and where other Segments are, so that they can use shared memory for some communications and fiber optics for others. The right way, in our opinion, is to create the concept of a Virtual Network (VNET). A Segment simply calls "Put" or "Get" indicating that it wants to give data to other Segments or get data from other Segments. It is up to the VNET software to determine how to transfer the data.

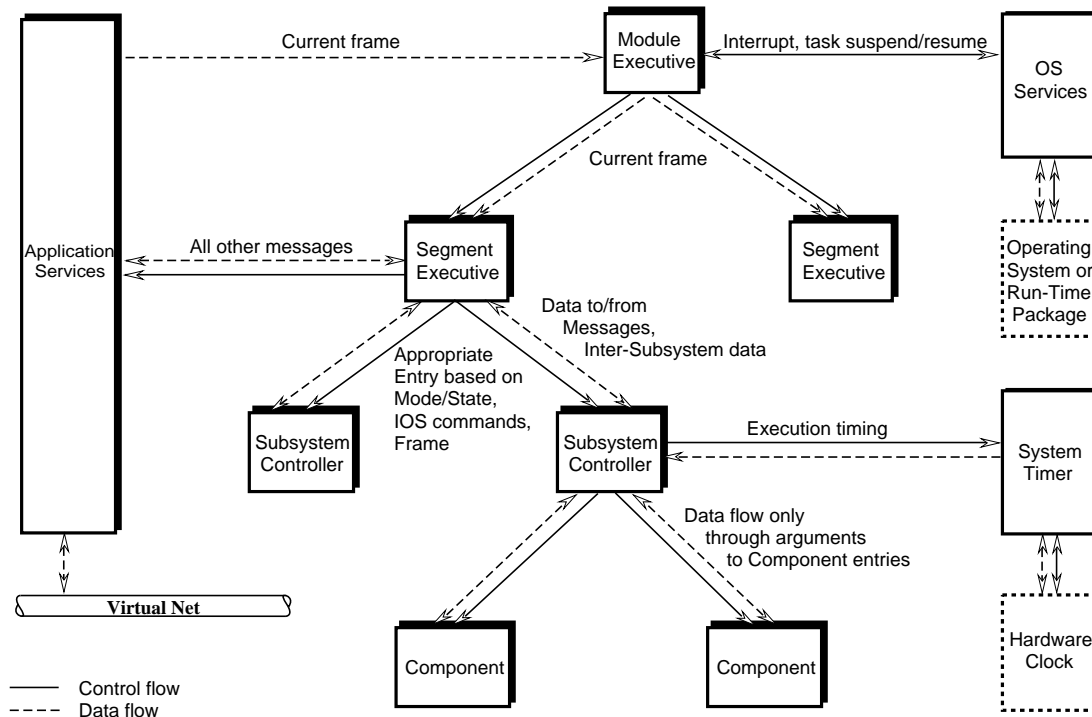


Figure 2. Domain Architecture for Reuse in Training Systems Overview

Characteristics Adapted From Structural Model

The internal workings of a given Segment are irrelevant to the operation of a "classic" Mod Sim. So long as the Segments meet their interface requirements, any internal architecture can be used.

Note that requiring standard interfaces is not the issue. Having standard interfaces makes it quite simple, for example, to subcontract and subsequently accept one or more Segments on the simulator. In the Mod Sim demonstration, we produced only two of the eleven Segments, subcontracting the other nine. This capability was retained in DARTS.

But to assert that the internal workings of a Segment don't matter is to assert that any architecture is as good as any other, which is contrary to our thesis. Accordingly, with only a few modifications that we describe below, we incorporated the SEI's AVSM into our Modules and Segments.

THE DARTS ARCHITECTURE

An overview of DARTS is shown in Figure 2. A

training system is divided into Segments; Segments are divided into Subsystems; and Subsystems are divided into Components. Segments are grouped together into Modules. Note that the analysis that produces the final architecture begins with functional decomposition and ends with what can sensibly be described as objects.

Is the DARTS analysis methodology "real" Object Oriented Design (OOD)? In one sense it certainly is, since the leaf nodes are what anyone would describe as "objects". On the other hand, since DARTS begins with a functional decomposition, it is occasionally necessary to divide the function of a Component into several Segments. For example, a hydraulic pump Component may exist in the Flight Station Segment to produce the simulation of hydraulic fluid flows, while a hydraulic pump Component may also be required in the Physical Cues Segment whose only function is to simulate the sound of the pump's operation.

We have preferred to use the term "Object Abstracted Design," and we follow those who view the applicability of pure OOD to training systems design with some skepticism.⁷ Rich McCabe of the Software Productivity Consortium gave an insight

which may temper some of the passions aroused by this issue: as a rule, functions in the real world are accomplished, not by spirits or demons, but by objects.⁸ To found a systems engineering practice on this commonsense notion seems at least defensible.

Module Executive

There is one Module Executive for every Module (computational system). All operating system and hardware dependent functions such as interrupt, task suspend and resume, and so on, are located here.

The Module Executive "causes" the Segment Executives to execute. This is kept deliberately ambiguous, since the right way of doing this on a given program may be to call the Segment Executives as subprograms, or it may be to schedule their execution as independent tasks. Because data flow between the Module Executive and Segment Executives is one-way and small (the clock tick message passes from the Module Executive to its Segment Executives), it does not stand in the way of implementing the right choice for a program.

Segment Executive

The Segment Executives are responsible for all communications over the VNET apart from the clock tick message. By isolating the VNET communications functions in the Segment Executives, the lower-level elements (Subsystem Controller and Component) may be reused from similar software for other architectures. All data contained in messages (that is, all data defined in the adapted DARTS Interface Specifications) flows through the Segment Executives between the VNET and lower level elements.

The Segment Executives are also responsible for mode and state control logic (total freeze, reposition, run mode, and so on).

The Segment Executives schedule the execution of their Subsystem Controllers by using a scheduling table mechanism similar to that used in the AVSM. A difference between DARTS and the AVSM is that functions such as malfunction insertion and mode/state change which the AVSM handles

through a separate aperiodic scheduling thread are handled in the main execution thread in DARTS: a mode or state change message in DARTS is a message like any other, though it is processed by the Segment Executives. Because of the way the AVSM does aperiodic execution, this is not a large or significant change.

The Segment Executives in DARTS call upon the appropriate aperiodic entries in each of the Subsystem Controllers, based on the receipt of the appropriate control messages through the VNET.

Subsystems and Subsystem Controllers

Subsystems correspond to the Functions allocated to Segments in the Mod Sim architecture. Though the analysis has been completed on only half the Segments so far, it appears that there will be little difficulty in accomplishing this for other Segments. Nevertheless, the possibility must be raised that more than one Subsystem will be required to implement a given Function. There is no structural impediment to doing this in DARTS.

Subsystem Controllers are implemented as in the AVSM. In the AVSM data flows out of Subsystems through a shared memory based Export Area, while in DARTS (largely because of the correspondence between Subsystems and interface messages) the Segment Executive provides the Subsystem Controller with data from messages and builds messages to send to the VNET.

All data flow between Subsystems takes place through buffers maintained in the Segment Executives.

We believe that it is the responsibility of the individual Components to provide "safe" input values for themselves. Thus, the Initialize entry for each Component provides both input and output data. Once the Component has executed its Initialize function, it may execute without error even though no input data has yet arrived over the VNET. This eliminates all worries about which Components or Subsystems need to execute before others in order to avoid erroneous data being processed.

The VNET provides information to the Import function as to whether or not new data has been re-

ceived since the last iteration. DARTS takes advantage of this by only copying data from message buffers when new data has been received. This new-data information is not available in the AVSM.

Components

As in the AVSM, the lowest level element is called the Component. Each of the Components corresponds to an Object in the OOD sense. Thus, a Hydraulics Subsystem may consist of Pump, Valve and Reservoir Components. However, the Hydraulics Subsystem may contain Components such as Flows, Bleeds and Pressures which are less clearly objects. Nevertheless, these "function objects" are assigned to Components so that the Subsystem Controller only needs to contain "glue" logic between the Components.

In DARTS, as in the AVSM, all knowledge about the operation and state of Components is contained within the Components. And no knowledge about the external environment (simulation control commands, presence or absence of other Components, computational environment) is contained within the Components. Components compute the state of the objects they simulate in a purely abstract, and therefore reusable, manner. These rules, which are among the most attractive features of the Structural Model, comprise "knowledge firewalls".

Just as in the AVSM, all data flow between Components takes place through the subprogram calls for each of the entries in the Components. As the SEI points out, this set of entries is both necessary and sufficient to permit the knowledge firewalls described above to operate.

MEGAPROGRAMMING AND DARTS

A team has been assembled consisting of ARPA, NAVAIR, NTSC, Boeing, DUAL Inc., and the SPC to demonstrate the applicability of the concept of megaprogramming to the training system domain, and, as part of the process, to evaluate DARTS as a domain architecture for achieving reuse in the context of megaprogramming.

As defined by STARS, megaprogramming is "the practice of building and evolving computer software

component by component. Megaprogramming builds on the processes and technologies of software reuse, software engineering environments, software architecture engineering, and application generation in order to provide a component-oriented product line"⁹

To realize a quantum improvement in the way software-intensive systems are developed, megaprogramming envisions two distinct but cooperating lifecycles, corresponding to the family of systems (product line, domain) and to the specific system (product) respectively.

Architecture becomes a key unifying feature of the product line lifecycle, while processes for its use are the driver for the product or project lifecycle.

The processes which drive the product line lifecycle are collectively known as domain engineering and include not only the familiar notion of domain analysis, but extend to managing the product line investment, creating reusable assets (processes and components) and supporting multiple projects that use those domain assets.

Under megaprogramming, the process of building a specific system is referred to as application engineering. Achieving the quantum improvement expected by megaprogramming comes primarily through leveraging the processes, components, and technology assets developed under the domain engineering investment effort to produce individual products very uniformly, quickly, and at the lowest cost per product.

The heart of this investment in domain assets is to pre-position all of the commonality among members of the family along with processes for adding values for the defined variability among all possible members of the family. For example, all Operational Flight Trainers simulate aircraft engines, while the number of engines varies from aircraft to aircraft.

Domain engineering work products are being developed for the Air Vehicle Training System (AVTS) domain based on the DARTS architecture. The work products follow the SPC Synthesis guidelines and are derived from the DARTS architecture.

<p>Leverage Major elements proven on multiple programs (F-16 Mod Sim, USAF Structural Model) Not tied to any CASE tool or computer vendor Subsystem specs and bodies and Component specs may be automatically generated Reuse of Components across multiple architectures Scalability by plug-replacement of Components, Subsystems, Segments Segments may be eliminated or combined for product-line variations Based on industry-wide Domain Engineering effort</p> <p>Simplifies System Understanding Small number of well-defined elements (12 Segments, Subsystems, Components) Structure maps to requirements analysis (early management visibility into software) Software engineering principles from SEI, SPC and STARS</p> <p>Rational Subsystems and Components are a toolset, not a straitjacket Results of early systems engineering activities flow into design Interface specifications clarify requirements, guide design</p> <p>Affordable Much systems engineering work is already done, simply by selecting the architecture Parallel development, testing improve schedule performance Lower integration time proven in F-16 simulator program Architecture proven in F-16 program to be fast, cheap enough for 50 Hz WST Exact specification of computer power, best computer architecture for segment</p> <p>Good Citizen Based on standards in public domain (FDDI, XTP, Ada) Mod Sim and Structural Model government-sponsored for simulation industry ISWG got wide consensus from government, simulation industry Contact with ARPA DSSA program through STARS</p>

Figure 3. Summary of DARTS Performance Issues

Each of the DARTS Segments has been defined as a domain and specified with: (a) a decision model for capturing the variability of the domain; (b) product requirements for representing the adaptable requirements; (c) product design for representing the tailorable design data; and (d) process specification that guides the application engineer through the instantiation of an instance of the domain.

The final step of the domain engineering process is to implement the domain (i.e., adaptable code and documents and information for their generation) so that the application engineer can generate the products for a given program.

Within the domain of each Segment, DARTS guided the domain analysis and each of the work products. Generally, Functions re-used from Mod Sim became DARTS Subsystems, and Components were derived for each of the Subsystems.

These work products are being incorporated into a Software Engineering Environment that will be used by application engineers to construct a T-34C Flight Instrument Trainer (FIT). The primary purpose of this demonstration effort is to show the benefits of megaprogramming on a real-world air

vehicle training device.

DARTS support of and conformance to megaprogramming has been recognized in its adoption by the Navy/STARS demonstration project. DARTS is specific to a domain; in this case, the product line of air vehicle training systems. With sponsorship from ARPA, the engineering data foundation of DARTS is being validated by subjecting it to a formal, defined domain engineering process authored by the SPC.⁶ Using the SPC's domain engineering process, DARTS is being configured to support high leverage reuse in the form of domain commonality and variability. Again, using the SPC's domain engineering methodology, DARTS is being extended to include defined processes for building out any member of the air vehicle training system product line.

ADVANTAGES OF DARTS

The performance of DARTS, as it appears to us at the present time, against the criteria we discussed at the start of this paper is summarized in Figure 3. Some advantages of DARTS which were captured from its progenitors deserve special mention.

Advantages Captured From the AVSM

The first set of advantages of the DARTS follows the advantages given for the AVSM, because DARTS incorporates such large parts of the Structural Model.

- The Subsystem Controllers and Components are based on reusable templates. Every Subsystem looks like every other Subsystem and every Component looks like every other Component, in that they have the same subprogram entries and the same package structure.
- Components are so structured as to be widely reusable. Since Components have no knowledge of their environments and little dependence on the architecture, they should be reusable in the widest possible context.
- Reuse becomes a matter of selecting and adapting from this set of identical parts, and it is entirely possible to automate this selection and adaptation based on a decision model captured in a SEE.
- DARTS provides an integration harness for each of the Components and Subsystems that can permit early, structured testing. Integration of Components into working Subsystems, integration of Subsystems into working Segments, and integration of working Segments into a working training system can be done in a structured manner, and early prototyping steps of design can be done with actual components.

Advantages Captured From Mod Sim

The second set of advantages of DARTS is derived from the advantages in the Mod Sim architecture, because many of its strongest features are also incorporated in DARTS.

- Since DARTS begins with a widely accepted decomposition based on functional requirements, requirements traceability is illuminated rather than obscured by the architecture.
- The division of Functions into Segments facilitates scalability. Quite often, the functionality of entire Segments is not required for a given training system. For example, Electronic Warfare is not

commonly found on transport aircraft simulators.

- Delivery is more predictable, since the components are all nameable and locatable very early in the program. Each of the Subsystems and Components can be tracked from a very early date in the program, so that reaction to delays and data voids have lower impact.
- DARTS is designed to permit Segments to be easily subcontractable. Companies with expertise in visual systems, electronic warfare, or weapons but which have little or no training system experience can compete to build the appropriate Segments. Software development and testing can take place in parallel with many workers and organizations until the very latest point in the schedule, thus greatly reducing time to delivery. Further, the ability of Segments to be tested as stand-alone components lowers both prime and subcontractor risk at acceptance.
- As requirements change, within a range of simulators, or as follow-ons require more or less CPU power, DARTS permits near-zero-effort addition or deletion of Segments and of computational power allocated to a Segment. Segments may be moved from one Module to another, again with near-zero effort. When this change is anticipated, a hardware architecture can be chosen for the affected Segments that permits the simple plug-replacement of CPUs with less powerful or more powerful CPUs.
- Interfaces between Segments are strictly specified in compilable Ada. Adaptation of these reusable interfaces to the requirements of a specific program is accomplished by the decision model for the domain, and we have demonstrated that it is easy to automate this process.

DISADVANTAGES OF DARTS

- Communications between Segments using messages and a VNET may take a larger amount of execution time than communications using shared memory, even when data synchronization (as in the AVSM) is taken into account. We believe that this price is small on today's CPUs, and will be smaller in tomorrow's CPUs.

Further, because messages permit a wider variety

of computational hardware to be used on a simulator, the dollars-and-cents cost of computers may not be very different between the two methods. Nevertheless, we see the possibility that another architecture or a modified DARTS will be more appropriate for some programs.

► DARTS, like the AVSM, absolutely requires data flow control, which takes engineering hours. Our slogan has been, "If you want to control data flow, you've got to control data flow." The generic, adaptable interface specification provides a great deal of help in this control process, and utilities associated with DARTS automate much of the tedious coding work like message setup and connection.

► Organizations and companies in the simulation industry have historically seen Mod Sim as a hardware architecture, of importance only in a niche. This Mod Sim ancestry may be a political disadvantage when dealing with those organizations.

► As we indicated earlier, since DARTS begins with functional decomposition, the functionality of some components may be spread across several Segments. We have observed that cases like this are the exception, not the rule. Still, a certain amount of object elegance is lost in DARTS, and effort is required to maintain concurrency among these objects. We have tentatively opted for giving identical names to Components whose "pure object" functionality is divided across Segments (i.e., a Component named Hydraulic_Pump in both Flight Station and Physical Cues Segments), since most configuration management systems can be made to indicate the connection.

CONCLUSION

DARTS has been the result of an evolutionary process which has incorporated the results of current research in software engineering and principles from Boeing Defense & Space Group's experience in the simulation industry.

Research at Boeing and at the SEI has proven that the two major elements of DARTS (Mod Sim and the Structural Model) are effective, low-risk architectures which can have significant impact on program cost and schedule. Research under STARS has confirmed this.

The overwhelming advantage of DARTS becomes apparent when it is used as a foundation for megaprogramming. Given the requirements for a specific program, the decision model for the domain and the adaptable, reusable components and interface specifications of DARTS, one can indeed "turn the crank" of a definable, automatable process to produce code and documents. This vision of process-driven reuse has been realized in our work for STARS, and is so much more powerful than other reuse models, that we believe it does represent a quantum improvement in the way we develop and reuse software.

REFERENCES

1. Software Engineering Institute. "An Introduction to Structural Models", 14th I/ITSEC, Nov. 1992.
2. Software Engineering Institute. *Structural Modeling Guidebook* (Draft), January 1993.
3. The Boeing Company. *System Segment Specification for the Generic Modular Simulator System*, Volumes I-XIV, S495-10400, June 1991.
4. Freemon, Brett W. and Crispen, R. G. "Testing A Technology For Reuse", 14th I/ITSEC *Proceedings*, Nov. 1992.
5. The Boeing Company. *Ada Simulator Validation Program Final Report*, D495-49506-1, Sept. 1988.
6. Software Productivity Consortium. *Synthesis Guidebook Volume I Methodology Definition*, SPC-91122-MC, Dec. 1991.
7. Gross, David C. and Stuckey, Lynn D. "Is Object-Oriented Design Sound Simulator Software Engineering?" 14th I/ITSEC *Proceedings*, Nov. 1992.
8. Rich McCabe, personal communication, April 1993.
9. Boehm, Barry W. and Scherlis, William L. "Megaprogramming (Preliminary Version)". STARS '92 *Proceedings*, Dec. 1992,