

DSHAs:
Domain-Specific Hardware Architectures for
Rapid Prototyping of
Application-Specific Signal Processing Systems
(RASSP)

Doug DeGroot
Texas Instruments

DARPA RASSP Presentation
Washington, D.C.
July 1992

APPLICATION DESIGN SYSTEM

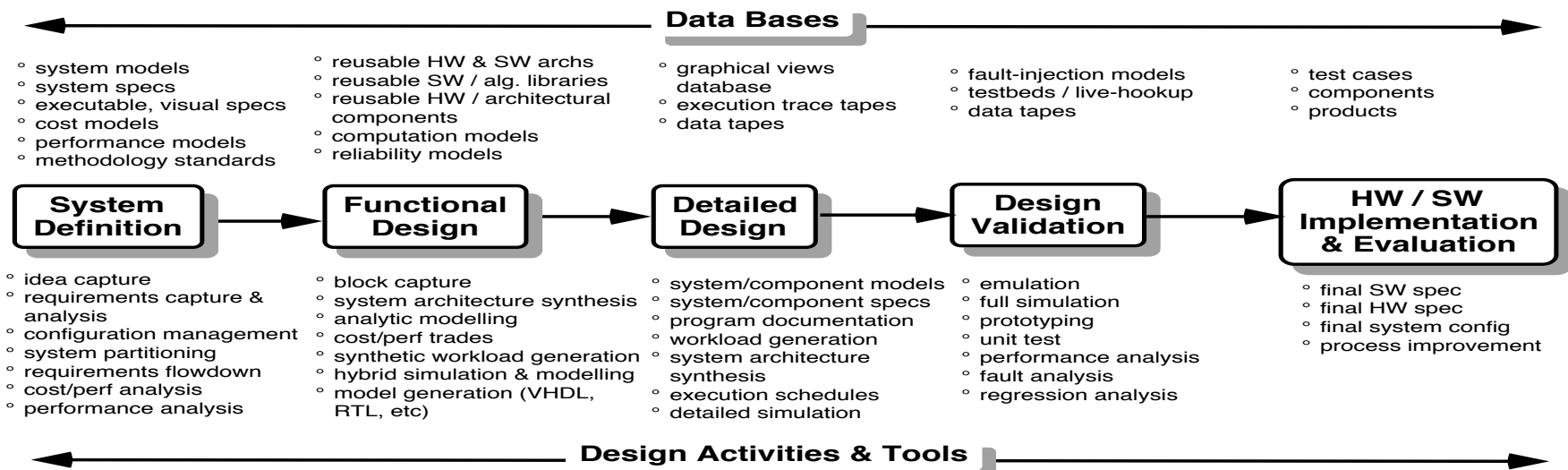
This section of the report focuses on the System Design part of TI's RASSP Vision. It also briefly readdresses the Libraries part of the vision, since the Libraries play such an integral role in the System Design. The Libraries part of the vision is discussed in greater detail in an earlier part of this report.

By System Design, we mean the integrated design and analysis of both the hardware and software components of a complex military system, and by system, we are here restricting our attention to computer-based signal processing systems.

Figure x.1 illustrates a typical breakdown of the major activity stages in a System Design process as it is generally practiced today. Above each stage name are shown some of the major data bases or data elements that are generally employed in that stage. Below each stage name are shown some of the major design activities and design tools generally employed in that stage. These lists are not purported to be complete in any sense, but they do show some of the typical databases required and design activities and tools required for each of the stages in the System Design process, and together they serve to illustrate a common view of the System Design process. Unfortunately, the System Design process is at present a completely informal process, and even when it is consciously followed, it is usually followed in a thoroughly ad hoc manner.

Fortunately, many of these tasks are quite difficult, and many of the required data elements are quite complex, powerful tools and technologies have been brought to bear on the system design process. But unfortunately, these tools and technologies have not yet been brought to any sort of formal integration that allow a systems design engineer to explore multiple facets of a complex system design from a variety of simulation and modeling approaches and to be able to transfer design decisions and insights from one design activity to another. Further, little capability exists for taking a system design process from the point of formal specifications and system requirements through a detailed analysis, modeling, simulation, and design activity and then to a formal set of design specifications for software and hardware implementation.

For RASSP to succeed, such integration is both necessary and desirable. Our approach to this high level of integration is described below. The main theme underlying our approach is the reuse of both formal software reference architectures and hardware reference architectures through Domain-Specific Software Architectures (DSSAs) and Domain-Specific Hardware Architectures (DSHAs) that are integrated through a formal System Design Environment coupled to knowledge-based design advisors and shared Information Object Modelers (IOMs).



Additionally, we believe the System Design process must be formally defined and modeled if the goals and needs of the RASSP methodology are to be met in the system design arena. This formal process must support both in-cycle and out-of-cycle system design activities, where by “in cycle” design activities we mean those design activities initiated as a direct result of a new product or model-year upgrade procurement process, and by “out of cycle” design activities we mean those design activities that lead to the reuse of, modification of, and extensions of both the software and hardware reference architectures and the System Design Environment in preparation for use in “in cycle” design activities. We further believe that our approach will and must support multiple modes of System Design activities, as we recognize that not all design activities of value to RASSP will be carried out as a result of formal product or model-year upgrade procurements. Other modes of system design to be supported include managed experimentation, on-going analysis, design upgrade, and continual "what if" explorations.

In-Cycle Support:

assumes reuse of HW/SW architectures (extended and integrated DSSAs & DSHAs)

Out-of-Cycle Support:

assumes reuse of, modifications of, and extensions of HW/SW architecture as well as introduction of new HW/SW architecture

allows "what if" analysis based on other-than-production goals

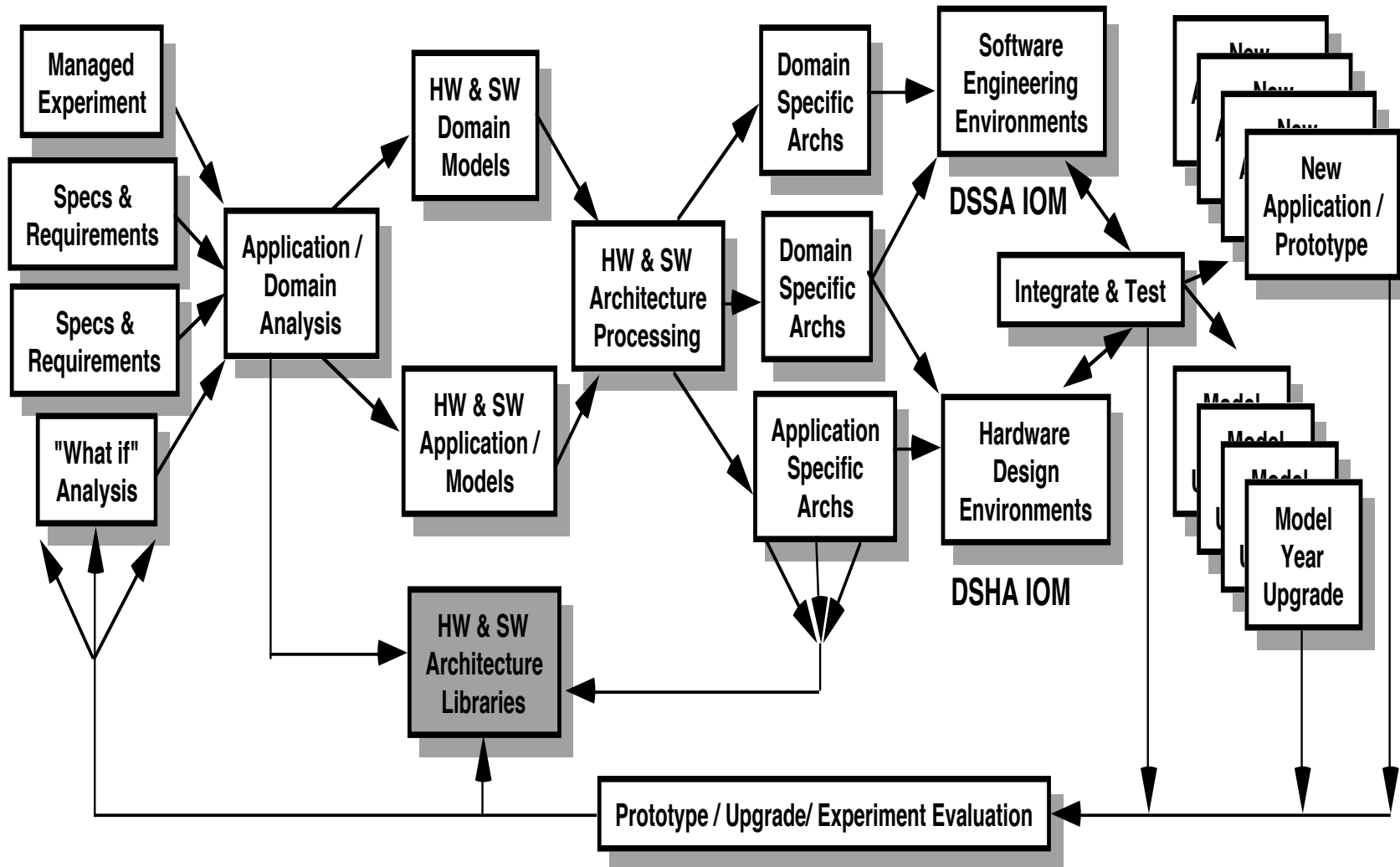
Modes:

Production Design, Model-Year upgrade, experimentation, on-going analysis and design upgrade

Finally, we believe the System Design process must be formally modeled to the extent of supporting multiple, well-defined measurement points and metrics for use in on-going benchmarking activities to support the RASSP methodology.

DSSA SOFTWARE / DSHA HARDWARE DEVELOPMENT

Figure x.2 presents a possible top-level view of a formal System Design process that is capable of supporting both in-cycle and out-of-cycle system design activities as well as supporting multiple modes of system design activities. The process model integrates both a DSSA-based software and a DSHA-based hardware design process. The process can begin from multiple points, depending on the goals of the person engaged in a particular system design effort. Each goal prescribes what in the previous section was called a "mode". Here, the modes shown include Managed Experiments, formal requirements (for both new product and model-year upgrade design), and "what if" analysis. As the process proceeds, both application analysis and domain analysis are required and supported, as we realize the possibility of either software or hardware designs that truly cannot be effectively generalized into a particular domain. Our belief is that the greatest advances in cycle-time reduction will result from reuse of domain-specific designs and knowledge, but even when such domain-specific data is absent, application-specific designs and knowledge can and should be directly supported by the DSSA and DSHA Development Environments. To simplify the remainder of this part of our presentation, we restrict our discussion to domain-specific design issues.



The description of the activities engaged in during each step of the process depends greatly on whether we consider the design and development of a new system or whether we consider the redesign, modification, or extension of an existing system that has already been placed through the process, as well as whether we consider the design activity to be "in-cycle" or "out-of-cycle" activity. In any event, depending on the goals and mode of the design activity, the system requirements are analyzed from a domain or application viewpoint, the hardware and software architectures are formalized and made reusable and extensible, and domain knowledge and design decisions are recorded for future and on-going design activities. These architectures, together with the domain and application related knowledge, data, models, and decisions are then made available through an integrated Information Object Modeler for access by both the software and hardware design environments.

Hardware and software designs and/or implementations can then be iteratively brought together and integrated for formal product design or model year upgrade, including final integration and testing. Additionally, as can be seen in the process flow, if the activation mode were one of managed experiment, prototype analysis, or "what if" analysis, reflections on the results of the experiment or design change can simply be fed back to the Libraries, back to the specifications, or back to the experimental environment.

The process model depends on and benefits from domain-specificity in the system design process in two ways. First, by narrowing our attention to specific domains in which both significant reuse of both a software reference architecture and a hardware reference architecture is exploited, many seemingly unbounded problems become bounded and thus manageable and possibly automatable. The restriction to certain, specific domains benefits the overall system design process, process model, and RASSP goals in another, equally important way, however, as it makes possible the development of intelligent or knowledge-based force-multiplication technology and approaches in the supporting system design tools and system design environment. It is perhaps these domain-specific force multipliers that will most support the short design-time goals of the RASSP program. Thus in our view, the term "domain specific" refers less to a diminution of capabilities than to a significant increase in capabilities that would not otherwise be achievable in a general-purpose approach.

There is currently a DARPA-supported DSSA technology development effort underway. The DSSA program is headed by Lt. Erik Metalla of DARPA SSTO. The current level of funding is approximately \$25M over four years (1991-1994). Six teams are currently involved in the DSSA effort: 1) IBM, 2) GTE/Comtel, 3) Teknowledge, 4) ORA, 5) Honeywell, and 6) TRW. In addition to these teams, a number of other companies, universities, and research centers actively track the DSSA program and attend the progress meetings. While there is no equivalent DSHA program underway, we believe many of the technologies being developed by the DARPA DSSA effort will prove immediately applicable to supporting our vision of a DSHA. Further, many ongoing research and development activities in both industry and academia will admirably fit into and support the goals of a DSHA. The development of a DSHA and its supporting Information Object Modeler should become a formal part of any RASSP effort.

In addition to the DSSA and DSHA development activities, a number of other technologies and programs that will prove beneficial to our integrated DSSA/DSHA System Design vision are being pursued; some of these are listed in Table x.1. Many of these will play a vital role in achieving the TI RASSP vision.

Relevant Technologies / Programs
STARS, Arcadia, Prototech, MARVEL, etc.
OODB (DARPA Open OODB, Falcon)
Knowledge DBs (Stanford, ISI, CMU)
Visual Design Envs. (ONR, FORGE, PieScope)
Architectural Synthesis (Micon, ViParSim, Ptolemy)
Massively Parallel Sim. (HPCC, PADS, VHDL)
 Hierarchical Design Advisors (MCC, CMU, SES)
System-level Simulation (Ptolemy, ADAS, SES)

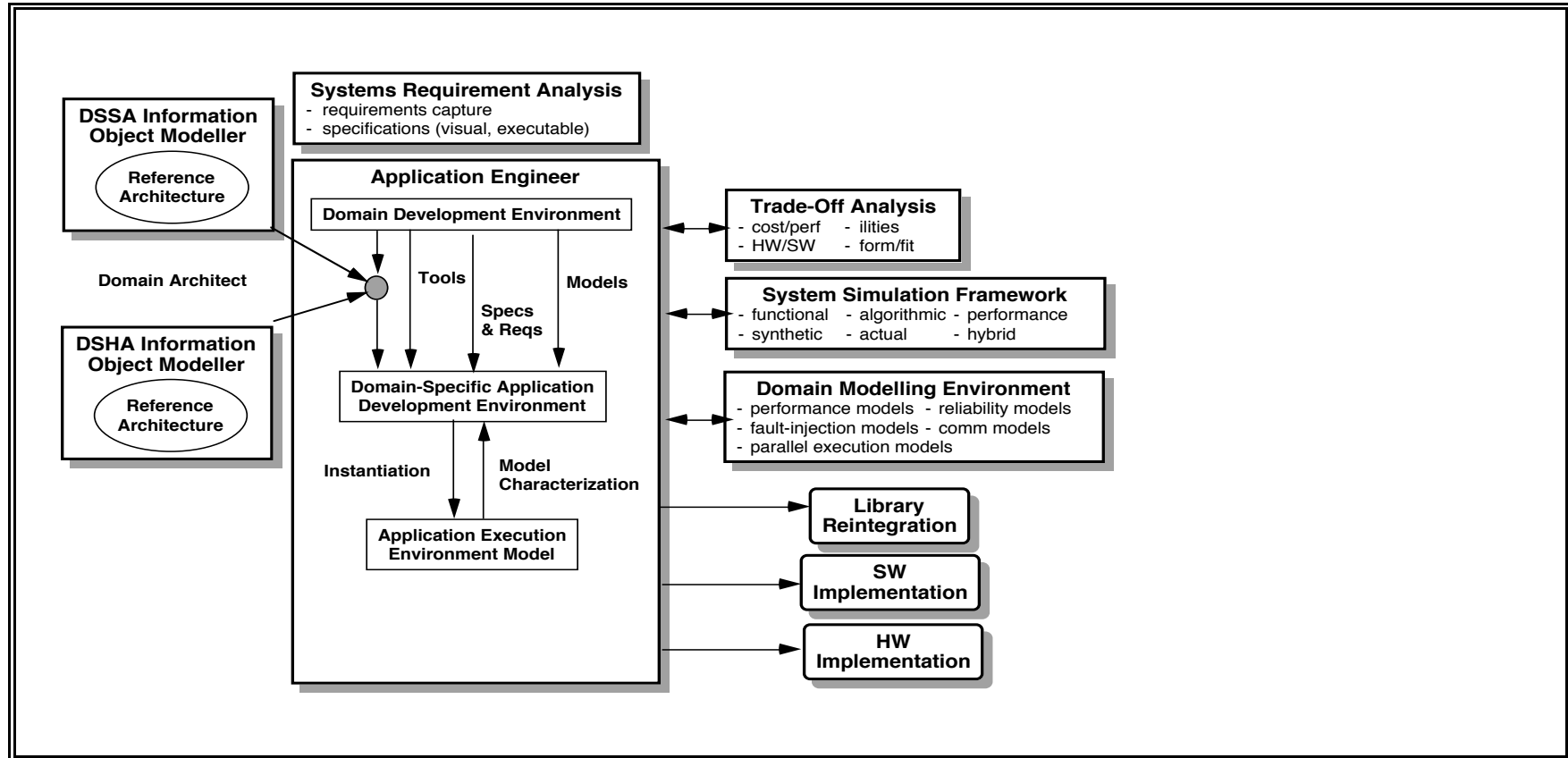
Table x.1 Other Technologies Relevant to DSSAs/DSHAs

RASSP APPLICATION DESIGN SYSTEM

Figure x.3 shows a top-level view of our proposed RASSP Application Design System. The Design System relies upon and heavily exploits the reusable software reference architectures and reusable hardware reference architectures provided through the DSSA and the DSHA Information Object Modelers. The "reference architectures" are those software and hardware architectures that represent the product or system that were previously designed during out-of-cycle design activities and are being upgraded, analyzed, or experimented with, either in a procurement-driven in-cycle design activity or an out-of-cycle design activity; for ease of discussion, the reference architectures are also referred to as the DSSAs or the DSHAs themselves, although such usage of these terms is technically ambiguous.

The responsibility of defining, maintaining, and extending the DSSAs and the DSHAs, along with their supporting Information Object Modelers, falls to the persons called the Domain Architects. The Domain Architects, mostly through out-of-cycle design activities, support the usage of the DSSAs and the DSHAs by the Application System Engineer. It is the Application Engineer who, during an in-cycle design activity, is responsible for taking a set of formal system specifications and requirements and delivering a formal, integrated system design that meets those specifications and requirements. Within the RASSP Methodology, this is done through the modification and extension of reusable hardware and software domain-specific, reference architectures. Although the DSSAs and the DSHAs are maintained and supported separately, they are integrated during application engineering through software-to-hardware mappers and machine organization synthesis, for example. The Domain Development Environment consists of a suite of integrated modeling, simulation, testing, analysis, and synthesis tools that are used both to integrate the DSSAs and the DSHAs and to perform a variety of

managed experiments and benchmarking trials. These exploit multiple formal system domain models such as performance models, cost models, and reliability models.

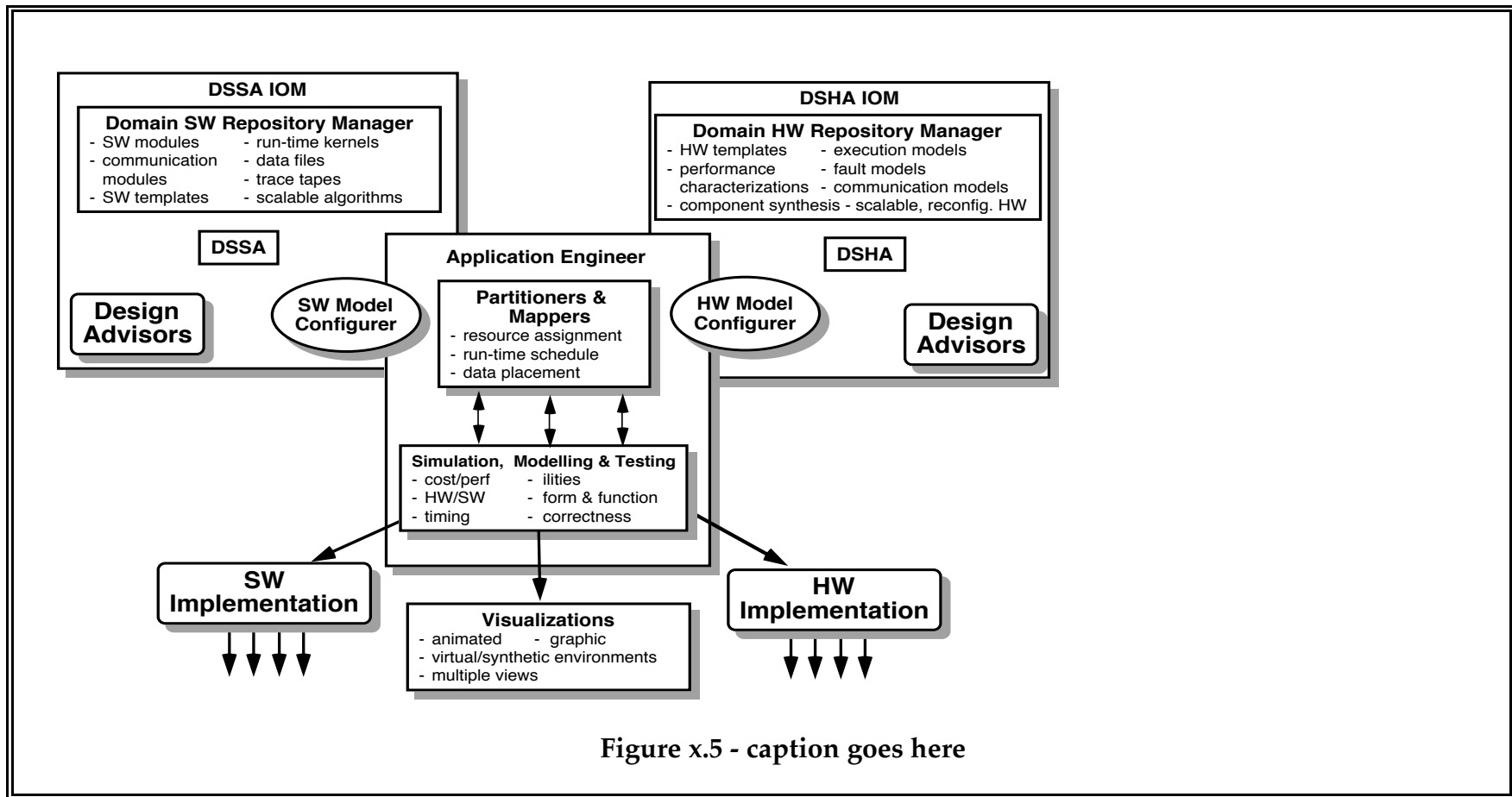


As the domain-specific application is developed, the Application System Engineer also relies upon a model of the Application Execution Environment. Instantiated models of the system design are fed to the application execution environment for validation, testing, and analysis, with formal characterizations of satisfiable application execution environments feeding back to the Application Development Environment. When the Application System Engineer is satisfied with the design, tests, and analyses, the completed design can lead to either implementation followed by test and integration, or simply back to DSSA, DSHA, and library reintegration, depending on the mode of the design activities being pursued.

Note again that the view presented here supports both in-cycle and out-of-cycle system design activities as well as the multiple usage modes.

DESIGN INTEGRATION AND VALIDATION

Given an existing DSSA and DSHA Information Object Modeler, the Application Engineer is responsible for integrating the software architecture and the hardware architecture and validating the resulting integration against the system specifications and requirements. To perform this integration and validation, the Application Engineer must instantiate both the software architecture and the hardware architecture. By suitably parameterizing these two architectures according to the required performance, cost, size, power, and reliability attributes, the Application Engineer initiates a set of bindings of code and hardware modules to the architectural components of the DSSA and the DSHA, respectively. A software model configurer and a hardware model configurer are controlled and guided by the Application Engineer to accomplish a set of bindings that satisfy a set of design constraints and criteria. The constraints and criteria define the set of acceptable software-to-hardware mappings, execution timing schedules, and data access behaviors for the intended system.



Given the two instantiated configurations, the Application Engineer can then simulate and model various aspects of the integrated system design. Through such simulation and modeling, the Application Engineer can perform a variety of hardware/ software tradeoffs, analyze multiple real-time schedules for robustness, assess the suitability of alternative hardware modules, memory configurations, and bus interconnections, explore alternative hardware configurations, and determine the optimal mappings of the instantiated software system to the instantiated hardware system.

Unfortunately, much of this sort of analysis and exploration is today done by hand, with very generic tools that know little or nothing about the functional responsibilities of the system being designed/examined and little or nothing about the environment within which the system will be fielded. Consequently, while the Application System Engineer may

receive productivity support through the use of these tools, little cycle-time reduction results since the tools can offer little if any analysis of the results of a trade-off study or a particular software-to-hardware mapping approach, for example. Consequently, when humans are heavily involved, rapid design frequently does not imply optimal design.

By focusing on specific application domains and relying on the significant reuse of both hardware and software reference architectures, we believe many design support tools can be extended into the domain-analysis arena and provide knowledge-based, application-specific design advice and analysis. Eventually, this extension of design tools into the intelligent, domain-oriented arena will allow more and more of the System Design process to become automated, with the eventual development of an automated design environment in which the Application System Engineer provides minimal correction and advice to an automated design system, rather than as today, where a set of semi-intelligent design advisory tools give minimal advice and correction to the human system designer. We want to reverse the roles!

To assist in the automated design process, the Application Engineer must be presented with a variety of domain-independent as well as domain-specific animated, graphical presentations of the various aspects of the system's design, construction, and behavior. These multiple, animated, graphical views are actually part of the Domain-Specific Application Development Environment's set of predefined graphical support system. The views provide both quantitative and qualitative animated views, with full support for virtual or synthetic reality explorations of complex, time-dependent behaviors and interactions. Because much of the design process will have been abstracted, encapsulated into domain-specific design advisors, and automated, a single Application Engineer can assume responsibility for multiple areas of a system design that are today addressed by multiple design groups, each consisting of multiple people, including hardware fabrication, cost estimation, cost and performance tradeoffs, and production scheduling. This will be made possible through an integrated, design-by-visualization application design environment.

In the process illustrated in Figure x.3, following completion of the simulation, modeling, and testing design activities, the resulting formal designs, specifications, and requirements are used to implement the software and hardware components, or again, simply to feed back into the Libraries or DSSA/DSHA Information Object Modelers.

EXTENDED DSSA SUPPORT ENVIRONMENT

Figure x.4 illustrates some of the fundamental components of and access methods defined for an Extended DSSA Support Environment. The majority of the domain-specific models, knowledge bases, data repositories, design advisors, and architecture components are integrated through the DSSA Information Object Modeler. As described above, the DSSA approach to reusable software benefits from domain specificity in two ways: 1) first, by being able to restrict its attention to specific domains, the problem space becomes manageably small enough that problems that first appear beyond our present capabilities suddenly become approachable, thus leading to early application of leading-edge technologies, albeit in limited (domain-specific) arenas; and 2) second, by allowing currently tractable technologies to be extended in unique and force-multiplying manners that allow significant capabilities for automated design that would be impossible in the domain-independent arena. Such domain-specific extensions and multipliers are encapsulated within the tool set and supporting data domains for each particular DSSA. These include simulation models for the multiple

functions of the DSSA, domain-specific reliability, performance, and execution models, among others, libraries of test data, workload suites, execution traces (both synthetic and real), software synthesis models, and the core of the reusable software libraries themselves.

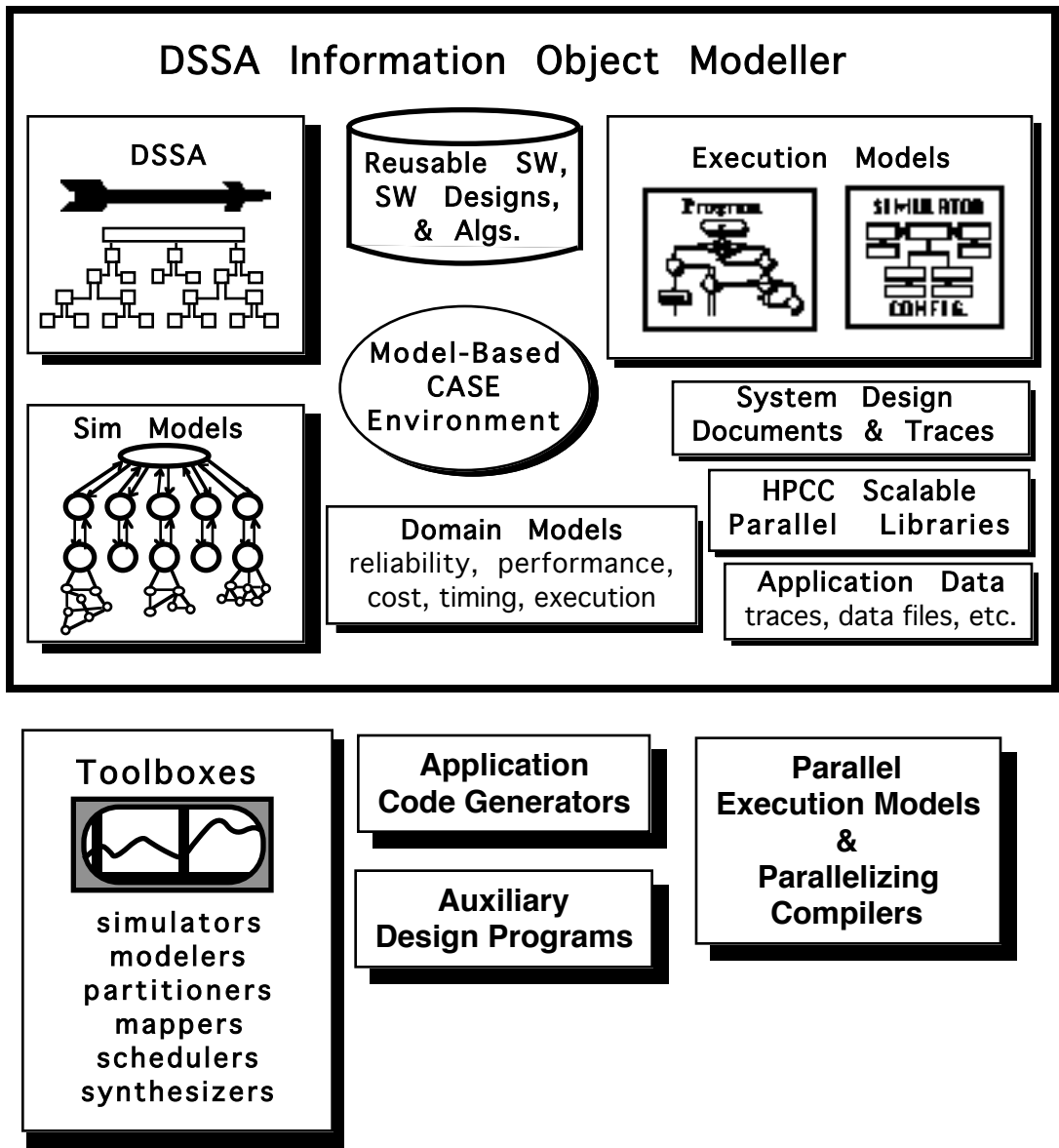


Figure x.4

Associated with the DSSA Information Object Modeler are a number of support tools, including the simulation systems themselves (not the simulation models), parameterizable software partitioning and mapping tools, domain-knowledgeable, off-line, run-time schedule generators, and complete software code generation and synthesis tools capable of extending scalable, parallel algorithms to fit application-specific configurations and performance requirements. We view these tools as being domain-independent, even general purpose, but capable of supporting domain-specific applications and knowledge through parameterized executions, where the parameters are domain-specific and are provided by the DSSA IOM. An analogy can be seen with knowledge-based expert systems; these consist of a domain-independent inferencing mechanism and a set of domain-specific, knowledge encapsulating rules and theories. While the inferencing mechanism can support a possibly large number of rule bases, it is the rule bases themselves that incorporate the domain-specific knowledge and make possible the design, synthesis, and analysis of complex system application. Accordingly, the DSSA IOM can benefit from broad-based, domain-independent technology advances in the general sciences of simulation, modeling, partitioning and mapping, performance prediction, etc.

While the view of an entire extended DSSA Support Environment and DSSA Information Object Modeler presented here may be beyond the scale of achievability within the 4-year RASSP Implementation program, it is important to realize that even without this view, some overall organizational philosophy of software system architecture and software reuse is required to make even the first level of the RASSP methodology realizable. To us, this first level of organization is respectably supplied by a minimal DSSA technology. The good news is that the DSSA technology is currently being developed under DARPA guidance, and it is being developed within the right time-frame. The DSSA technology being developed can easily be extended to support the rapid prototyping and design requirements of RASSP and to support a formal Model-Year upgrade process and process model.

Because the DSSA technology will evolve over time, we have presented here our view of where this evolution will lead. We also believe the DSSA approach should be extended to contemplate an equivalent DSHA technology, and particularly one in which multiple, parallel processors play a leading role in future military system designs. The DSSA and the accompanying DSSA IOM must be integrated into the formal System Design process and process model. Finally, Table x.2 lists some of what we believe to be the most important needs for an effective realization of the RASSP Methodology based on the DSSA approach.

Table x.2 - DSSA Technology Needs

DARPA DSSA methodology

extend to accommodate DSHA extend to accommodate parallel processing

Domain-Specific Design Advisors

system: software partitioning & mapping, performance, real-time schedule, data placement, etc.

application: guidance & control, parallel image processing subsystems, etc.

DARPA open OODB technology

extend to high-performance Knowledge DB

extend to Active DB with "design interest" activations and process enactments

Common SW system simulation model

timing, function, performance, reliability, etc.

analytical modeling, discrete-event, Petri-net, etc.

common interfaces / data exchange

hybrid; hierarchical; animated simulation views

Formal system-level specification langs

SW architecture description languages

performance specification languages

temporal behavior specification languages

fault-behavior specification languages

Parallel execution traces & data models

methodologies for capturing parallel executions

synthetic workload generators / trace tapes

Application and process benchmarks

e.g., DARPA Image Understanding Benchmark

ISPW-6 SW process model; common sensor data files

EXTENDED DSHA SUPPORT ENVIRONMENT

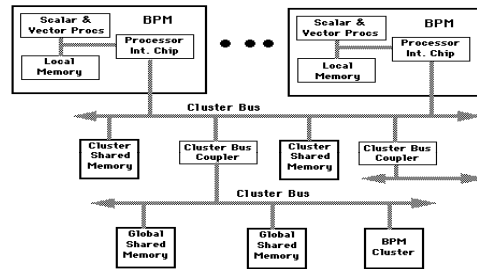
One of our basic beliefs is that the DSSA technology currently being developed to support reuse of software reference architectures can be easily, directly, and naturally extended to support the reuse of hardware reference architectures, and within the RASSP methodology. Reuse of hardware reference architectures is fundamental to the goal of rapid, model year upgrades for complex military systems; this ability would be provided through an integrated Domain-Specific Hardware Architecture (DSHA) System Design Environment. Like the Extended DSSA Support Environment figure before this one, Figure x.5 illustrates some of the fundamental components of and access methods defined for an

Extended DSHA Support Environment. The majority of the domain-specific models, knowledge bases, and architecture components are integrated through the DSHA Information Object Modeler. Just as software architectures can be elevated from the application-specific level to the domain-specific level, we believe hardware architectures can be extracted from existing, fielded designs, that they can also be elevated to the domain level, and that they can thus be reused for new system design as well as for model year upgrade. Texas Instruments, for example, has a well-defined, formal architecture for guidance and control sections of missiles that has seen significant reuse. Further, our own internal research with the DSHA approach has led to significant insights and substantiations of these beliefs.

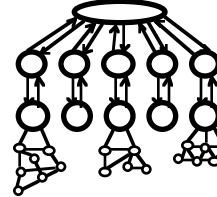
As with the DSSA Information Object Modeler, the DSHA Information Object Modeler contains domain-specific extensions and multipliers encapsulated within the tool set and supporting data domains for each particular DSHA, including, for example, detailed, hierarchical and hybrid simulations for hardware modules at the system, chassis, board, multichip module, and component levels; parallel architectural models represented in multiple performance, behavioral, and costs aspects, and in multiple simulation languages and abstractions; and hardware building block and component libraries, with hardware templates that allow extraction-by-need of performance, execution, or reliability data, models, and/or specifications.

DSHA Information Object Modeller

DSHA



Sim Models



- ### Parallel Architecture Models
- dataflow
 - shared mem
 - systolic
 - msg passing
 - task-structured

HW Component / Building Block Libraries

- functional templates
- performance templates
- fault injection schema
- execution models
- reliability models
- ISA sims/emulators

Table x.3 DSHA Technology Needs

Adapt DARPA DSSA methodology to DSHA

Domain-Specific Design Advisors

packaging, system synthesis, component synthesis, HPCC architectures, reliability

DARPA OODB technology

extend to Knowledge DB

extend with Active DB with "design interest" activations

Architectural description language(s)

domain-specific module interconnect language

HW module/component/system arch descriptions

domain-specific parameters / generators

performance reliability characterization language

Simulation models

model-based analytic models

parallel discrete-event simulation systems

plug-compatible interfaces for hybrid simulation

Architectural synthesis

library of scalable, reconfigurable building blocks

algorithm-to-silicon compilation

application-specific adaptation of reusable HW arch

scalable reliability

Flexible memory architecture

shared memory, cache-coherent, physically distributed; real-time compatible

Experiment management system

stimulus generation, monitoring, data collection and analysis

Software to hardware mapping

simulated annealing / genetic algorithms, dynamic load balancers, data mappers, real-time schedulers (off-line & dynamic)

Visual modeling & simulation tools

qualitative as well as quantitative graphical views

animated virtual world presentations of complex cost/performance and execution data models

As with the Extended DSSA Support Environment, the Extended DSHA Support Environment is associated with a number of support tools, including simulation systems, hardware partitioning and synthesis tools, synthetic workload generators, and fault exercisers.

Table x.3 lists some of what we believe to be the most important needs for an effective realization of the RASSP Methodology based on the DSHA approach; this list certainly does not claim to be exhaustive. While a DSHA will differ significantly from a DSSA, we believe much of the underlying technology development can be shared and mutually exploited by both a DSSA and a DSHA development program. As with the DSSA, the DSHA and the accompanying DSHA IOM must be integrated into the formal System Design process and process model.

DSSAs/DSHAs

Figure x.6 illustrates three example DSSA reference architectures down the left column and three example DSHA reference architectures down the right. The top, left example is from the domain of "ground vehicles"; this example was used in a recent DARPA-sponsored workshop on DSSAs. The second DSSA example is of a missile seeker; it too was presented at the DARPA DSSA workshop. Note that it is written in the Object-Connection-Update formalism. The bottom DSSA example is of a missile guidance system developed at TI as part of the RASSP study phase effort but used in several DoD projects throughout the past several years.

The three DSHA examples on the right illustrate, respectively, a high-level breakdown of the hardware architecture of a signal processing system, a missile signal processing system, and a reconfigurable, scalable hardware architecture capable of supporting extremely high-performance, missile-based, signal processing systems. Similar DSHA illustrations were shown in Figure x.888.

In the middle of this chart are two illustrations that demonstrate the integration of both a DSSA and DSHA Information Object Modeler through an Application Development Environment. Additionally, these illustrations show the ability of the IOMs and the Application Development Environment to support multiple graphical views, each based on the differing information and modeling needs and expertises of the multiple users of the architectures. What is important to us is to provide both an integrated set of views that can be used by the different people involved in a RASSP effort as well as an integrated set of views that can be used to present the different aspects of a design and implementation effort to a single person, letting that person explore the ramifications of his/her design decisions on other parts of the product effort in ways that are not currently possible.

Because the types of views needed to present different information flows, design parameterizations and instantiations, and cost/performance aspects, for example, all require differing models of visualization, it is important to be able to translate information stored in one model into information for another model. We believe it is both necessary and important to develop a formal Application Design System that utilizes the technology of Common Denominator Representation concepts to support multiple, wide-ranging modeling objectives by the multiple users over a variety of

usage modes, from model-year upgrade, to initial system-design, to managed experimentation. These representations must be developed as part of the overall RASSP program effort.