HIGH CONFIDENCE SOFTWARE AND SYSTEMS

RESEARCH NEEDS

High Confidence Software and Systems Coordinating Group

Interagency Working Group on Information Technology Research and Development

January 10, 2001

The HCSS National Research Agenda is dedicated to the memory of Andy Arenth of the National Security Agency, who served as the Working Group's Co-Chair from 1998-1999. Andy skillfully energized and guided the group in establishing a unified vision of the end goal. His contributions and leadership are gratefully acknowledged and will be sorely missed.

HCSS RESEARCH NEEDS

TABLE OF CONTENTS

EXECUTIVE SUMMARY

HCSS RESEARCH NEEDS: EXECUTIVE SUMMARY

Introduction

This White Paper presents a survey of high confidence software and systems research needs. It has been prepared by the High Confidence Software and Systems Coordinating Group (HCSS CG) of the Interagency Working Group in Information Technology Research and Development (IWG/IT R&D). The IWG, which functions under the White House National Science and Technology Council, coordinates Federal multiagency IT R&D efforts. The HCSS CG agencies are: DARPA, NASA, NIH, NIST, NSA, NSF, and OSD/URI.

A high confidence system is one that behaves in a well-understood and predictable fashion. It must withstand malicious attacks as well as naturally occurring hazards, and must not cause or contribute to accidents or unacceptable losses. The HCSS research described in this White Paper focuses on the critical basic science and information technologies necessary to achieve predictably high levels of performance, system safety, security, reliability, and survivability. Security-critical and safety-critical systems (i.e., high-consequence systems) exist in the domains of transportation, health care, electric power generation, manufacturing, oil and gas production, chemical production, and financial services, as well as in law enforcement, emergency services, and national defense.

While high confidence systems have been built in these and other domains for many years, a number of interrelated trends in technology, system requirements, and economics are creating a new environment that challenges the limits of traditional engineering approaches, drawing attention to the need for a radical new approach.

Technology Trends:

- Increasing reliance on software for critical functions
- Increasing reliance on a commodity technology base
- Increasing interconnectivity among disparate systems, products and services

Requirements Trends:

- Increasing complexity of products and services
- Increasing scale with respect to number of concurrent users
- Increasing stress due to higher performance demands
- Increasing exposure to potential compromise due to internal and external access to communications and computing resources, which adds security to other system requirements

Economic Trends:

- Accelerating product and service development cycles due to market pressures
- Expanding developer base of high consequence systems to include non-experts in HCSS techniques

Accompanying these trends are issues and problems associated with both the need and the capability to achieve high

confidence, all of which provide compelling and urgent reasons to pursue this research now. Among them are:

- Achieving high confidence is becoming more difficult as systems become more complex. Today's trends towards widespread use of commercial off-the-shelf (COTS) technology, increased integration, continuous evolution, and larger scale are yielding more complex systems.
- New analysis techniques are urgently needed. Simple forms of analysis previously used to achieve high confidence are at their limits for today's complex and continuously evolving systems. For example, the assumption that concerns are separated by composing components does not account for cross-cutting (e.g. timing) properties in complex systems. These limitations of assurance technology will not allow us to continue to make progress at the levels of integration currently planned. The implications are that complex systems will become less safe and assured without new computer aided analytic tools to assist in addressing the design, analysis and validation of these systems
- The U.S. is not alone in its growing dependence on computing in industries producing safety-critical products/systems. This is especially true in transportation, health care, energy, and manufacturing sectors. Europe and Japan are investing heavily in assurance and dependability technology because they have recognized the need, and the US must also in order not to fall behind.
- Lacking adequate protection, today's information and communications systems will almost certainly become the target of numerous malicious attacks and inappropriate access attempts. New and advanced techniques are required to achieve the necessary system security. Protection against both external and insider threats must be developed, including mechanisms for data protection and for system monitoring, detection, response, and recovery.

The HCSS research described in this White Paper addresses these urgent needs. This research seeks to make the development of high confidence systems less costly while achieving higher quality and providing an enabling capability for unforeseen applications and opportunities. It will narrow the gap between expected and delivered dependability as embedded and information systems become ubiquitous.

The primary outcome of this research will be technologies for building predictably safe, reliable, dependable, secure, and survivable computing and communications systems that can enable a world radically different and much improved from today's reality. These improvements are essential to systems providing for the national security and general welfare of the public, and to meet growing performance and capacity goals of Federal agencies. Systems that use HCSS technologies will be more resistant to failure and malicious manipulation; will respond rapidly to damage or perceived threat by adaptation or reconfiguration; and will be safer, better protected, and more dependable. Such an achievement is necessary if Federal agencies are to accomplish many of the performance and capacity goals contained in their long-range strategic plans and move closer toward their

visions of the future through information technology.

Strategic Goals

This HCSS research embodies four strategic goals that reflect the responsibilities of the Federal government. Pursuing this research will support the Government's role in protecting the welfare of both the public and the individual consumer, and will promote cost and quality improvements in the provision of Government services and the effectiveness of national security.

Protect the Public. This strategic goal focuses on the Federal government's responsibility to protect the public's safety, health, economic welfare, and security. The government must assure the Nation's critical infrastructure services upon which individual citizens depend. To meet this strategic goal, the Federal government must promote technologies that can increase confidence in the safety, reliability, trustworthiness, security, timeliness, and survivability of systems such as transportation systems and communications systems.

Protect the Consumer. This strategic goal focuses on enabling higher reliability, safety, and ease-of-use in commercial products. It envisions cost-effective means to gain assurance that enables commercial products to meet certain minimum quality standards. Results sought include expedited quality certification, validation, and verification; shortened times to market; simplicity of use; plug-and-play interconnection; lower lifecycle costs; and improved customer satisfaction. Confidence is needed in consumer products and services. Such products could include "smart" cars, medical devices, consumer electronics, business systems, smart houses, sensor technologies, Global Positioning System (GPS) receivers, smart cards, educational technologies, electronic commerce software packages, educational technologies, and digital libraries.

Promote Improved Government Services. This strategic goal focuses on the Federal government's continual obligation to provide improved services to the public. Improved government services include ubiquitous, correct, and timely service. These services must become increasingly efficient and use appropriate technologies to help lower the cost of service delivery. HCSS technologies will support the Federal government's modernization efforts that will allow the Government to deliver better service to the public at significantly lower costs.

Promote National Security. This strategic goal focuses on the Federal government's responsibility to protect the security interests of our Nation. The Government must protect the Nation's critical infrastructures upon which the Nation's economic health and security depend. It must also protect military systems that are used to defend our national interests. National security will require defense-in-depth protection services and assurance that those services will perform as required. For example, National air defense systems that protect against incoming missiles and launch defensive missiles are extremely complex and require very high assurance against malfunction and malicious attack.

Technology Goals

Five technology goals must be met through this research to achieve the HCSS strategic goals:

Provide a sound theoretical, scientific, and technological basis for assured

construction of safe, secure systems. To meet this goal, the research must:

- Achieve the capability to specify, design, and integrate components, and assess system behavioral properties including potential modes of failures
- Provide the capability to enforce specific behavioral properties such as performance and throughput
- Be more predictably tolerant of specified behavioral failures including malicious attack mechanisms with graceful degradation and limitation of services

Develop hardware, software, and system engineering tools that incorporate ubiquitous, application-based, domain-based, and risk-based assurance. To meet this goal the HCSS research must:

- Provide methods, tools, and environments for the seamless design, construction, and evaluation of software and systems operational properties
- Provide high confidence systems software and behavioral enforcement mechanisms
- Develop better indicators of overall system reliability that can be achieved through the application of such methods, tools and environments

Reduce the effort, time, and cost of assurance and quality certification processes. To meet this goal, this HCSS research must:

- Improve the productivity of information system design, development, and analysis while simultaneously improving the levels of confidence that can be achieved through such productivity enhancements
- Provide better technology for validation and verification
- Explore information from better-integrated design, development and assurance activity
- Move from dependence on testing to a comprehensive analytic framework

Provide a technology base of public domain, advanced-prototype implementations of high-confidence technologies to enable rapid adoption. To meet this goal, this HCSS research needs to develop reference implementations to illustrate innovative HCSS methods and techniques. This includes system software components and assurance tools, libraries of reusable software implementing verified algorithms and mathematics components, and transition.

Provide measures of results. To meet this goal, HCSS research must develop measures of performance and effectiveness for evaluating confidence improvement achieved through HCSS technologies. Further, it must show that the benefits achieved are cost effective.

Table ES-1 at the end of this Executive Summary provides a strategic overview of this HCSS research.

Research Components

The HCSS research described in this White Paper has four organizational components:

- An *HCSS Foundations* component devoted to developing the supporting

theory and scientific base for high confidence systems.

- An *HCSS Tools and Technologies* component to apply the foundations of HCSS in engineering technology. Tools, techniques, systems software, and associated libraries will provide the engineering framework required to design, build, and certify large-scale systems.
- A research component devoted to *HCSS Engineering and Experimentation* with HCSS tools and technologies that will provide mature reference implementations, scalable proofs-of-concept, reusable tools and techniques, and empirical evidence on HCSS capabilities and limitations.
- A research component devoted to *Demonstrations and Pilot Projects* to apply HCSS technologies to specific user problem domains.

HCSS Foundations (providing the science to build high confidence systems). To achieve high confidence in our future systems, we must first know how, and then provide the technological means to build them. The HCSS foundations component addresses the first of these needs by advancing the scientific basis for designing systems that require critical properties such as security, safety, and reliability.

Research to be pursued under this component includes:

- Theory
- Specification
- Interoperability of formal reasoning techniques and HCSS tools
- Composition/ Decomposition

HCSS Tools and Technologies (providing the means to build HCSS systems). Applying the foundational research to building high confidence in our future systems, we must next provide the means to build such systems. This research component will develop the technology capability (tools, techniques, and supporting component libraries) for sound engineering practices that can apply the theoretical foundations of high confidence to the construction of critical systems. Within this component of the research effort, the following research areas will be pursued:

- Programming Languages, Tools, and Environments
- Modeling and Simulation
- HCSS Building Blocks
- Robust System Design
- Monitoring and Detection
- Validation
- Evidence and Metrics
- Process

These elements must work together collectively (interoperate) in order to support reasoning about high confidence software and systems properties through all phases of development from early design and specification to implementation and validation.

HCSS Engineering & Experimentation (showing that high confidence systems can be built). The tools and techniques that need to be built must be shown to work efficiently and effectively at realistic industrial scales. The benefits to be derived from this research include empirical data that demonstrates increased confidence can be achieved in a cost-effective manner. This component will apply emergent foundational and technology capabilities to challenging areas of system engineering to provide both mature HCSS

reference implementations and the empirical evalutation of HCSS engineering capabilities and limitations. Potential system engineering challenges for pursuit under this component include:

- Software Control of Physical Systems
- High Mobility Systems
- Integrated Hardware Design Environment

HCSS Demonstrations and Pilots (transitioning HCSS technologies to user-agency and commercial domains). Pilot projects will provide a context for all the other components. This component will ensure demonstration and evaluation of HCSS technology research in real systems and at realistic scale. From engineering experience, it is known that techniques that work on small examples will often not work for large, real-world problems. It is critical that high confidence techniques be validated on real systems from the outset to make sure that the research is addressing the right problems.

Table ES-1. HCSS Strategic Overview

Technology Goals	Implementation Strategy	Research Component
Provide a sound, theoretical, and technological basis for assured construction of safe, secure systems	• Develop supporting theory and scientific base for HCSS	HCSS Foundations
Develop hardware, software, and system engineering tools that incorporate ubiquitous, application-based, domain-based, and risk-based assurance	• Develop tools, technologies, and libraries to design and build large-scale systems	HCSS Tools and Techniques
Reduce the effort, time, and cost of assurance and quality certification processes	• Deploy an HCSS engineering technology	HCSS Engineering & Experimentation
Provide a technology base of public domain, advanced-prototype implementations of HCSS technologies to enable rapid adoption	• Develop mature reference implementations, scalable proofs-of-concept, and reusable tools, libraries, and techniques • Conduct experiments in challenging areas of system engineering	HCSS Engineering & Experimentation
Provide measures of progress	• Develop and apply measures of performance and measures of effectiveness	HCSS Demonstrations & Pilots

HCSS RESEARCH NEEDS

PART1: HCSS FOUNDATIONS

1.1 THEORY

Our notions of high confidence systems are being altered and considerably broadened because of the pervasive role of information products and systems in our lives. We rely in unexpected ways upon the correctness and integrity of computing systems, and increasingly our privacy, safety, and well-being depend upon them. Engineering disciplines must have a solid theoretical foundation. Systems that merit high confidence require a theoretical basis for designing and reasoning about them and for evaluating the trust that we place in them.

Status

The risks associated with large, interconnected networks and complex systems are strikingly apparent. Research of the past several decades has not resolved difficult problems such as complexity, scale, and system interaction. Early attempts towards developing theoretical foundations for security and safety have not proved sufficient as a basis for repeatably building safe and secure systems. Current research efforts often pursue a patchwork approach, developing isolated components to address individual concerns; they do not provide fundamental scientific and engineering underpinnings. The consequences can be seen in the area of security, where cryptography, network management, public key infrastructure, intrusion detection and response methods were developed largely independently and fail to provide a systematic basis for engineering secure systems. The case is similar in the area of safety. Isolated techniques exist for computer-aided control design, fault tolerant computation, real-time scheduling, reliability estimation, and criticality analysis for potential failure modes and effects, but these techniques are not integrated, and there are many gaps. For example, they do not adequately address cognitively complex human roles or the adaptive systems increasingly enabled by the power and flexibility of software. The dependability consequences of moving from highly-interactive, human-controlled systems to autonomous or highly-automated systems are not well-understood.

A disturbing lack of theory exists to ground useful assurance technology and practice. We currently depend on naïve metrics derived from hardware wear and fatigue models and on process-based scrutiny of design activities for certification. Useful concepts such as fault tree analysis and failure modes, and effects criticality analysis currently are used, but they are not composable and are easily defeated by the state space explosion that results when subsystems are combined. The consequence is a nearly complete dependence on testing that drives up the cost of software certification and often results in incomplete coverage of system failure conditions. Breakthroughs in scalable model checking have produced dramatic improvements in high-confidence computer chip design, but their application to software, and complex, networked systems remains largely undeveloped. Other promising formal verification approaches suffer from inattention to principles of usability, performance, algorithmic complexity, and scale.

Analysis plays a very minor role and does not provide effective validation and verification of complex embedded and networked systems. The emergence of ubiquitous computation and pervasive embedded device technology will magnify this problem. Enabling advances in device and software technology, operating systems, and networks may be revolutionized in the future by research in areas such as biological and quantum computation. Unless we can provide better foundations for high confidence development and certification, these gaps in safety, security, and assurance will widen rapidly and will increase the likelihood and frequency of both inconvenient and catastrophic system failures.

Needed Research

A new effort to create the science necessary to repeatably construct high confidence systems is needed. Fundamental issues exist in reasoning about concurrency, context-sensitivity, and interference. New research is needed to develop the theories, models, and representations for symbolic and numeric reasoning about system dependability and trust; for analyzing both the structure and behavior of software and systems; for considering the effects in the physical world of system actions controlled by software; for decomposing and allocating requirements and software; and for composing assurance in tandem with subsystems combined into systems. New research is needed to examine these issues, both from the specific perspectives of safety and security, but also in the multidisciplinary integration of these perspectives, where many common issues and strategies can be found.

Modeling and Reasoning. The added time and effort required to model systems as well as build them has proved a strong disincentive to using modeling in software and even system design. Research in design representations should add core support for modeling key system properties in order to move the discovery of design flaws earlier in development, reduce total effort, and enable more effective system acceptability and certification review ("validation and verification") than exists today. Research into abstraction and model reduction must help tame complexity and make analyses tractable for reasoning about high confidence system designs. Abstraction should be exploited to mechanically assist the derivation of models of different properties, and a sound understanding of abstraction must support valid connections between concrete and abstract models. A theory of abstraction should also support reasoning at different, and possibly multiple simultaneous levels of abstraction, and should enable selective focus on different properties.

The mathematical foundations for verification require renewed effort if this technology is to have routine application in future high confidence systems. Past research in formal methods and verification yielded numerous successful applications, especially in hardware and protocol verification. However, much greater capability and flexibility will be required for the verification of complex, software-centric, interconnected systems. New mathematical modeling frameworks and fast decision procedures are needed for reasoning about continuous and discrete time processes. Hybrid system descriptions are needed to relate continuous physical processes to discrete events, transition systems, and software logic. Numerous key research areas remain underdeveloped and unexploited that could improve modeling and analysis for designing and certifying high confidence software and systems. These include: probabilistic methods and stochastic approaches for modeling uncertainty; better-integrated formal and informal methods; domain-specific theories, models, and analysis; and reasoning support for critical operational

systems. Examples of these systems include free flight support in future air traffic management systems or complex medical devices such as anesthesiology systems, medical imaging and radiation treatment systems.

Operating Systems and Middleware. This research will develop foundations for high confidence development of system software. Operating systems and middleware provide the services upon which most high confidence systems will be built. They encapsulate functionality and protection that would be difficult and wasteful to re-implement in repeated, ad hoc development. Their stability and high level of use yields high payoff for any assurance effort. This research seeks predictable, safe, and timely performance for all levels of operating system capability including composable support for pervasively embedded devices. The shift from strict layering in operating systems to tailored composition of distributed systems services reveals an expanded space of fundamental problems such as configuration control, guaranteed timing, and adaptive module reuse, where high confidence methods will be required to achieve predictable qualities of service. The use of distributed object technology and micro-protocols present new opportunities but also new challenges for the development of high confidence systems. Object-oriented design and programming has simplified and improved software development and maintenance, but foundations are needed for verifying and validating object-oriented operating systems and middleware frameworks.

Networking. New paradigms must be developed for operation and management of networked systems to provide transparent protection, better adaptation to changing environments, assured availability of critical services, timing and performance guarantees, and protection against disruption through naturally-occurring events or malicious attack. New foundations must also be developed for networking to support real-time applications and deeply embedded systems of devices.

Central to all network technology is the design of communications and management protocols. Improved theories for the design and analysis of networking protocols are crucial to predictable, robust network performance of control area networks, and predictable services for wireless and mobile communications. Demand for high-performance networking calls for investigation into high-speed technologies, novel encryption device structures, and system architecture. Rapidly emerging optical networking technology raises the priority of research into timing and reliability, vulnerabilities, and protection for optical communications. New approaches must determine how to monitor and assess the health of networks and respond to stress conditions such as overload. New technologies such as active networks potentially can be exploited to achieve high confidence as well as new functionality. Foundations must be established for assuring, evaluating the robustness of, and extending information protection domain techniques such as virtual private networks (VPNs). Many of these issues lie at the intersection of research in scalable information infrastructure and high confidence software and systems and can be addressed jointly.

Security. In the security domain, research in public key cryptography has helped to advance the development of electronic commerce, but there are numerous problems that hinder wider adoption of this cryptographic technology. Problems associated with extremely large-scale key distribution, certificate management, and interoperability each require fundamental advances. The inability of traditional cryptographic systems to perform efficiently or in real-time applications, operate flexibly in diverse environments, and withstand novel attacks all point to

the need for improved mathematical underpinnings. Revolutionary technologies such as quantum cryptography offer the potential to provide "theoretically sound" security protection, but this technology is in its infancy and requires additional work. Even less mature is the mathematics research needed to thwart potentially devastating attacks postulated from quantum computing technology. Other security needs include: secure access control and process separation primitives; control of information flow across system boundaries; code validation techniques including runtime methods to determine code safety and restrict execution; secure implementation of cryptography algorithms and systems to prevent inference of keys and secure group communications protocols; advanced authentication technologies including PKI for attribute-based authentication; and mechanisms for managing and enforcing authorized actions.

Safety. In the safety domain, fundamental research is needed in safety properties of complex systems. Better foundations are needed for routine engineering for safety-critical problems, including: scalable modeling and reasoning for failure modes and effects, allocation of functionality to guarantee resources to critical functions and provide failure isolation, more precise methods for reasoning about the propagation of criticality due to component interaction and interference. Analysis methods need to be developed that can be easily tailored to specific domains for reasoning about safety-critical operational policies and protocols. Research should be expanded in implementation mechanisms such as safety kernels, to monitor, constrain, and enforce sequences of safety-critical actions. New software development methods are needed that support systems engineering goals and assure safety criteria, including technology support for systematic introduction into functional software of code to manage safety-critical properties. Examples include: concurrency, computation order, time, synchronization, fault tolerance. Development is needed of high confidence middleware services that both implement and support safe invocation of computational patterns such as pipelines, time-triggered execution, event-based reactive execution, and publish-subscribe. Better foundations are also needed for certification practice, including approaches for "forensic" analysis of software and systems in validation and verification. Rapid formulation of exploratory analyses might, for example, exploit game-theoretic scenarios for product-based V&V and certification. In cooperation with human computer interaction and information management research, new failure models and analytical techniques must be developed to aid in reasoning about software and human computer interaction. Such foundations would help design interactions that do not induce mode confusion or other sources of human error, and help detect vulnerabilities related to human operation. Better foundations are also needed for managing risk, skill, and command authority that must shared by both human operators and system automation in mixed-initiative systems.

Benefits

This research will provide sound theoretical underpinnings for the development of the technologies required for next-generation high confidence systems. It will provide the means to organize and discipline tool and technology development. It will yield new models for reasoning and critically needed domain theories that can be used to create more rigorous development environments. With such a foundation, we will be increasingly able to design advanced information systems with more assurance that the problems of complexity, scale, and interactions are adequately and appropriately handled.

1.2 SPECIFICATION

Specification is the process of describing a system and its desired properties. The system properties specified might include internal structure, functional behavior, timing behavior, and performance characteristics (e.g., resource consumption). Formal specification is the act of writing things down precisely. Formal specification uses a language with a mathematically defined syntax and semantics. It removes ambiguity associated with other means of expression, and it is specified with sufficient precision to be machine-processed. Thus it can form the basis for automated reasoning (for example, to discover whether it meets system requirements, or whether it contains certain types of errors). With rigorous semantics, a specification may be used to convey a system's requirements for high confidence, support simulator and correct-by-construction code generation, generate criteria to prove that these requirements have been satisfied, and serve as a reference for validating that these requirements are being addressed during system design and implementation.

Specifications describe systems, sub-systems, and components at different levels of abstraction and for different domains. A system's requirements are usually described functionally. As the system design is refined, sub-system and component functions are described in terms of interfaces and services. As the system is further refined, detail is added, possibly of different types. Notations specialized for domains facilitate the communication of essential concepts (e.g., design rules, models, and theorems) of the domain among system designers and engineers. Tools that perform checks on these notations verify that specified requirements are satisfied by the implementation.

Status

Specification has been most successful for functional properties. For software specification, a few areas of industry are now open to trying out generalized formal notations to document a system's properties more rigorously.[1] One current trend is to integrate different specification languages, each able to handle a different aspect of a system. Another is to express non-functional aspects of a system such as its performance, real-time constraints, security policies, and architectural design. These notations, while useful, are not widely applied in industrial settings primarily because they are not easy to use, not scalable, and do not address the domain-oriented specification problems of those industries.

Needed Research

Specifications must convey necessary information to the system builder in a useful and unambiguous manner. Research is needed to investigate how to define, represent, and use logical and verifiable specifications for individual components and their interfaces used to construct

[1] Some formal methods focus on specifying the behavior of sequential systems. States are described in terms of rich mathematical structures like sets, relations, and functions; state transitions are given in terms of pre- and post-conditions. Other methods focus on specifying the behavior of concurrent systems. States typically range over simple domains like integers or are left uninterpreted, and behavior is defined in terms of sequences, trees, or partial orders of events. Still other formalisms wed two different methods, one for handling rich state spaces and another one for behavior, interaction and concurrency. Common to all these methods is the use of mathematical abstraction and composition.

high confidence systems. More needs to be understood about how to modularize and compose component specifications. Efficient and effective methods need to be developed for decomposing a computationally demanding global property into local properties whose verification is computationally simple.

Specification methods and supporting languages and tools should facilitate modeling and abstraction and they should provide means to automate the correlation of the different levels of abstraction within and among systems, sub-systems, and components. Foundations are needed for specification technology that supports reasoning at multiple levels of abstraction and across component boundaries. Means to support analysis of legacy systems are also needed. In limited circumstances, it should be possible to develop abstraction techniques that can construct various types of abstract specifications from existing systems code.

Specification languages that are executable are needed to support tools for system visualization, design, analysis, implementation, and validation. A greater range of domain-specific specification languages is needed to accommodate hardware and software design. Sound abstractions, tailored for various system or problem domains, need to be developed and formally justified. Tailorable specification approaches need to be developed to guide the construction of tools that provide expressive power, problem-based reasoning support, efficiency, and ease of use.

Benefits

This research will result in new approaches to specification. The main benefit of specification is gaining a deeper understanding of the system being specified. It is through this specification process that developers uncover design flaws, inconsistency, ambiguity, and incompleteness. The specification is a useful communication device between customer and designer, between designer and implementor, and between implementor and tester. It serves as a companion document to the system's source code but at a higher level of description.

Formal specifications help to ensure with a high level of confidence that a system behaves as required without unintended or unexpected side effects. Moreover, the tangible product of formal specification is an artifact that can be formally analyzed (e.g., checked to be internally consistent or used to derive other properties of the specified system) and can be used in automated system construction. Systems derived from formal specifications are self-documenting and generate their own validation and verification criteria. When it is necessary to execute mobile or foreign code in networked environments, specification can play a vital role in providing contracts for safe execution.

1.3 INTEROPERABILITY

Different methods are suited for modeling and reasoning about different system properties. For example, a state machine may be best for specifying certain kinds of behavior, while a dependence graph may be best for modeling information flow. Given that no one formal method is likely to be suitable for describing and analyzing every aspect of a complex system, a practical approach is to use different methods in combination. A future issue is that a problem may be too hard to represent in full detail. In this case, a related but more abstract approach is required.

Many difficulties originate in the competing requirements of different properties. For example, the resource requirements of methods for fault tolerance may negatively affect performance. This demands more than a simple combination of different methods; an interacting interpretation of their meanings must be provided. To address competing requirements, various views of the system must be considered where a single change can potentially affect all.

Status

Interoperable approaches for reasoning are needed in order to consider multiple properties and the interaction of these properties and to provide an explanation that spans different methods and supports multiple properties. Some progress has been made on mixing deductive and analytic techniques, and in establishing relationships between logical methods. However, many barriers exist to interoperability. Methods that translate models represented in one logical system to another are not mature. The dependence of a proof on a given logical theory is often deeply embedded. To explain the proof in another theory entails making the rules of inference of the first theory explicit in the second theory.

Needed Research

Research is needed to enable combination of assurance methods that analyze and establish system properties. Interoperability research must provide a key part of the theoretical basis for the tools research described in further detail in Languages, Tools, and Environments (Part 2, Section 1) and in Modeling and Simulation (Part 2, Section 2). Managing human effort in these tools is a key issue. This research should enable clear tradeoffs to be made between the capability of methods used and the effort required to apply them. It can also enable the integration of reasoning into languages and tools that provide information and a usage context. For scalability, this research must develop useful and efficient ways to apply mixed techniques. For example, by first applying analytic (e.g., model checking, decision procedures, simulation) methods one can limit the number of cases that require full search for a proof using deductive methods.

Research is needed into the combination of reasoning methods and of different models to address mutually dependent properties. Dependency analysis that can span different models for different, and sometimes competing, properties is necessary to reveal the pattern and extent of interference. Means must be developed to reason about and explain the consequences of interfering properties and to resolve or mitigate the interference.

Research is needed in open frameworks to organize composite reasoning problems, apply appropriate tools, and present composite results. Research is needed in exploiting failed proofs, which can produce counterexamples to find and explain cases in which a system would fail to perform according to its requirements. In a composite reasoning problem, the explanation must span methods.

Foundations are needed for strategies that combine both approximate and detailed methods and that provide a range of automatic and interactive operation. This research should inform the development of useful tools rather than providing a purely theoretical perspective.

Benefits

By providing a composite analysis framework, this research will enable the use of description and reasoning techniques that are most natural for particular problems. It will simplify and provide general approaches for combining results and for simultaneously achieving desired but competing requirements. This research will enable reasoning and explanation that spans different models and supports multiple properties. An important benefit would be the early production of counter-examples that explain design flaws, bridging different methods to provide combined explanations of complex interactions that could lead to failure.

1.4 COMPOSITION AND DECOMPOSITION

Better scientific understanding is needed on the effects of integrating components and systems, and how to go about this in order to achieve requisite system properties with high confidence. Systems of the future often will be distributed, may have many components, and will be highly dependent on networking and information infrastructure. Components executing in the context of other components may produce unexpected, possibly undocumented, behavior. The computational environments in which they must execute cannot be predicted exactly, and diverse failures must be tolerated. Many systems requiring high confidence involve control of physical systems and require simultaneous control of subsystems through coordinated actions. One component in the computational or physical environment may interact or interfere with another component's operation through (1) physical effects, (2) shared computational resources, or (3) algorithmic or heuristic logic. Such interaction or interference may be through explicit interactions components take (or fail to take), thereby modifying the behavior of other components. Dynamically occurring interference effectively widens "the interface" through which components interact. It can be sources of unexpected failure and failure propagation. So-called "emergent behavior" can also result in systems that are integrated from components. This was dramatically illustrated in the replicated and interacting behavior of recovery software that led to the cascading AT&T long-distance failure in 1990[2].

Status

Progress has been gained in reasoning about logical isolation and encapsulation to prevent interference of operations in computer hardware. However, concurrent operation, dynamic adaptation for logical mismatch, and various types of interference are difficult areas for reasoning about systems and they are key challenges for understanding increasingly software-centered systems.

Modern software and programming language designs have improved modularity and information hiding mechanisms, which limit their potential to interfere with each other in the storage and retrieval of data values. However, this has led to design practices that assume components are "compositional" in all other behaviors: specifically, that they can be integrated and still operate the same way in any environment. For many environments, this assumption does

[2] Philip Elmer-Dewitt, "Ghost in the Machine," *Time*, January 29, 1990, pp. 58-59; (see also "Phone Outages Blamed on Switching Software," *IEEE Software*, p. 100, September 1991).

not hold. In the component marketplace, components are currently provided without explicit descriptions of their behavior. Furthermore, implementation mechanisms intended to achieve specific requirements (e.g., fault tolerance through reconfiguration) are currently designed separately and can inadvertently defeat another property (e.g., security). Techniques are lacking for examining global system behavior that results from composing local, often replicated, behavior.

Needed Research

Research needed to remedy this gap includes theoretical frameworks, engineering methods, and supporting tools that directly enable composition and decomposition (both integration and top-down design methods) with reasoning about consequences for crucial system properties. Sound composition and decomposition methods and sound engineering principles must be developed as a foundation for languages and tools that can be used to engineer robust systems. Balance and synergy must be achieved between the engineering support needed for constructing complex systems and the logical and mathematical support required to extract and combine properties across systems. A scientific basis is needed for detecting and managing interference and interaction in systems. For example, it might be useful to determine which parts of a computation or system are sensitive to context and which are not. Other issues include accommodation of unknown behaviors and environments. Techniques are needed for constructing composite views, for analyzing interaction, and for fast automatic checking of system-level properties.

Foundations are needed in such areas as analyzing software-based concurrency to check isolation and enable partitioning and encapsulation of effects. This can provide a basis to be exploited in languages and engineering tools for building systems that robustly manage interference and limit the propagation of failures. Challenges include the following:

- New approaches to component technology and sound composition of "peer" components coupled with design methods that weigh multiple aspects of a system and compose property-based measures to achieve overall system assurance.
- Support for managing hierarchically designed or layered systems. This includes reasoning and transformation that crosses layer boundaries: hardware, networks, operating systems, and applications. It also includes management across the boundaries of interacting, hierarchically structured subsystems.
- Support for explicit description and analysis of interactions between the system and its operational context. This would further enable reasoning about, and controlling, the interactions of software and systems with the environment or with human operators.
- Representation and management of "meta-descriptions" for recording and managing composite system structure.

Benefit

This research will enable the systematic construction of software-centered systems from components. It will lead to designs that can provide more robust, reliable operation in unpredictable environments. It will provide a new technological basis for adapting components and systems to different contexts. It will improve the predictability of composite system operation and will provide a sound scientific and technological basis for engineering design and construction tools.

PART 2: HCSS TOOLS AND TECHNIQUES

2.1 PROGRAMMING LANGUAGES, TOOLS, AND ENVIRONMENTS

The kinds of systems we want to build are beyond the complexity manageable by sheer human effort. As in other scientific and engineering disciplines, computer aided design and analysis tools serve as enablers and multipliers of human capabilities to design, analyze, and reason about complex systems. Tools can encapsulate sophisticated theory and methods, reduce skill and training requirements, and amplify theoretical and domain-based engineering skills. Tools that assist and enhance the design and analytic capabilities of people and support their reasoning about the properties of complex systems will enhance productivity, enhance the quality and reliability of software-based systems, and reduce costs of production and maintenance of products.

Technology support is needed to embed assurance into the software and system construction process. Languages, tools, and environments must be developed that include support for reasoning about concepts relevant to the domain of application and reasoning about properties demanded of the software. The spectrum of such support includes design tools, programming languages and user support for their correct and efficient use, code generators, and technology to promote verification and testing. These, in turn, involve notations, graphical interfaces, shared databases, theorem provers, modeling methods and model checkers, test-case generators, documentation management, and other logical and developer-oriented mechanisms. All development support software should increasingly foster the production of "correct-by-construction" code and provide analysis capabilities for all stages of development and maintenance.

Status

The software in an application system is inextricably interrelated with the physical or information environment of the domain in which the software is deployed; however, general-purpose programming languages and verification systems do not naturally incorporate the concepts of the application domains (operational or structural). Constraints and relationships that are natural to the application domain must be tediously and repeatedly built into the system, slowing and complicating development as well as breeding possibilities for error. Similarly, general-purpose languages and verification systems are not expressive enough to enable or enforce system properties such as fault tolerance. The same weakness is found in design tools. As a consequence, the software development process and the software assurance process are often conducted as separate activities, with the result that properties such as performance and fault tolerance may be ignored at the outset and then patched in (often unsuccessfully) once the development is well along.

Programming languages hold a unique position in the software development process, being the facilitator and gatekeeper between concept and implementation. The source code produced is

the basis of both the product implementation and the ongoing maintenance activity thereafter. Source code also bears the human-accessible semantic content of the system: Faults as well as upgrades are, with current technology, understood and dealt with in terms of the programming language. In addition, compiler technology is often cited as one of the great success of computer science. The field has advanced tremendously in both theory and technology, with a current ability to routinely produce optimizing compilers for a bewildering array of source and target languages and computer architectures. In contrast to the more traditional development methods, the newer intentional and aspect-based programming approaches promise implementation support that avoids explicitly programmed imperative actions that allow the implementation to be optimized with respect to performance and other non-functional properties such as fault tolerance. Although this would appear to overcome many of the problems mentioned earlier (separate development of functional and non-functional aspects), the research is very recent and results are immature.

Another promising direction is that of design patterns where recent work has shown how such patterns can be established, tested, and reused with a concomitant increase in assurance for the systems constructed. Such high-level reuse holds potential for more widespread adoption and greater effect than reuse of code, but much more remains to be studied. Little work has been done in pattern verification methods or in ways to provide specialization or modification of patterns for use in circumstances that demand high confidence. Tools for verification would appear to hold the greatest promise for providing high confidence because of the unequivocal, logical guarantees they can offer. Verification methods, and model checking in particular, have enjoyed a sudden and dramatic increase in acceptance and use for computer hardware verification. Widely publicized failures of commodity processors and the subsequent detection via model checking of the source of the failures have increased industry awareness of the risks of unverified hardware designs. Similar progress has not been achieved for software systems, which have far less regularity and much greater complexity than hardware systems. Of the various directions being explored to increase the use of verification techniques, two appear to hold the greatest promise: "lightweight" verification methods and extended type systems.

Lightweight methods are those that sacrifice some aspect of completeness of verification in order to provide more "effortless" application of the tools. While complete verification of a system is the ideal, it is seldom practical for two reasons: The computation time required is exorbitant and the sophistication required of the developer is too high. In these cases, some combination of incomplete and even informal methods may be useful to automatically eliminate some classes of errors. Where full verification is seldom modular or compositional, techniques based on abstraction and partial correctness often can be so. A number of recent techniques such as proof-carrying code (PCC) and code certification depend on lightweight proof checkers rather than heavy-duty theorem provers. Of course, proofs must still be generated, but this more difficult activity is done at the time of code generation while the checking is done at run time. Good engineering approaches are lacking for the management of effort in a coordinated verification and development activity. Proof-carrying code illustrates the strategic combination of two tools: one a theorem prover (heavyweight) that exploits the information-rich, compile-time environment, the other a proof checker (lightweight) for runtime checking.

Finally, the techniques used by compilers and those of verifiers have not yet come together in extended type checking. Traditional compilers provide type checking, which is a limited form of

verification, assuring that programs make consistent use of data and functions. Checking of extended types can be done by model checkers, verified decision procedures, or theorem provers. While there are a number of encouraging theoretical results, this line of research is in its infancy.

Needed Research

The overall HCSS languages, tools, and environments strategy is to eliminate error sources through a combination of logically precise automated mechanisms and automation-supported human activity. Domain-specific models and languages are needed that are readable and reviewable by application and systems engineers, and are formal and rigorous enough to be interpreted and analyzed for specific properties, for example system safety properties. The environment in which the development takes place should assist the developer to detect and correct mistakes as early as possible. With automated assistance, the code that is eventually generated should be guaranteed to be free of as many types of errors as possible. Research is needed in analysis that can be made on the code incrementally as it is developed. Ideally the developer will not have to learn an entire new method, but will view the assurance step as an enhancement of his/her current method. Consequently, the methods can be made not only lightweight, but relatively invisible and nonintrusive or helpful, with simple, small extensions to what developers already do and to the tools and languages they find useful for development. This goal is far easier to achieve in domain-specific contexts than at the most general level, since the semantics of the domain can be incorporated to guide, restrict, and reason about the development process. New technologies are needed to integrate domain-specific reasoning capability into languages and design tools. Such technologies would include domain-specific and application-specific extended type systems as well as techniques for specializing more general languages and tools with domain theories and inference mechanisms. Domain-specific techniques need to be incorporated in languages and tools, for example hazard analysis in the safety domain and vulnerability analysis in the security domain.

Advances are needed to make model checking fast enough to be practical for irregular computational structure, such as is exhibited by software. Promising directions for expanded applicability are abstraction and hierarchy, set- and graph-theoretic methods, and modular model checking. Research is needed in combining approaches, including: program slicers, model checking, theorem proving, semantic analyses, relational analysis over finite domains, type inference and analysis, and testing. In this area, approaches should be explored to support risk- and effort-based use of these methods. Research is needed in good engineering approaches for managing verification effort. This includes both new technologies that can reduce or allocate effort and approaches for making tradeoffs of verification effort and risk. Furthermore increasing use of graphics, non-standard input modes, and voice entail careful application of these media in language definition and for increased productivity and program quality. Formal methods and language research can take advantage of these new technologies.

Other important research should address expression of and reasoning about additional attributes such as real-time constraints, performance requirements, fault-tolerance, reliability, security demands, and failure detection and recovery. Lightweight methods and tools should be extended to help establish these properties as well. Intentional and aspect-based programming needs to be developed further in this same regard.

Benefit

Tools that assist and enhance human design and analytic capabilities, and support human reasoning about the properties of complex systems will enhance productivity, "raise the bar" for quality and reliability of software-based systems, and reduce costs of production and maintenance of products. Programming language research can support the creation of high confidence systems by providing expressive, semantically sound methods for implementing problem solutions. Compilers can provide stringent tests and useful feedback, filtering out a variety of errors and supporting the validation process. Good language design can significantly improve the maintainability and evolvability of a system. The use of domain-specific languages and tools provides even tighter assurance that the semantics of the application are respected.

Assurance technologies that can be embedded in languages and design and analysis tools will simplify the construction of systems with requirements for highly assured non-functional properties such as real-time response. For such systems, the traditional approach has called for separate measures to achieve the assurance with a resultant much higher cost or the abandonment of the high assurance goals. The new assurance technology would reduce the tendency to defer or omit assurance measures due to time and cost pressures. Safety, security, and fault-tolerance would be considered in design from the outset rather than being patched onto existing designs. With combined assurance and tool strategies, the bar can be raised for the lowest level of assurance that can be widely demanded. The mechanisms to assure desired properties would be simplified for many cases, and additional verification effort could be selectively applied according to risk.

2.2 MODELING AND SIMULATION

The importance of models is well understood in the design of complex mechanical systems where modeling and simulation are used to explore possible behaviors of the modeled system, narrow the design space, and reduce expensive construction and physical testing steps through early illustration of design hazards. Similar approaches are needed to enable development of high confidence, software-centric systems and to reduce test and certification costs. However, as useful as simulation can be, it is often incomplete, missing important inherent behavior. Formal model checking and verification may be needed to provide more rigorous and comprehensive evaluation.

Model construction is prerequisite to both formal and informal reasoning about systems. Even though techniques such as model checking are automated, they may depend upon manual construction of appropriate models of software and systems. It is difficult to develop models, and most currently are "crafted by hand" by a team of simulation experts or logicians separate from the engineers who design and build the system. This doubles the effort by requiring separate development activity and assurance activity. It also de-couples development from assurance, with the result that assurance plays a lesser, or often even neglected, role, or that the system and the model (which is the basis for reasoning about the system's properties) diverge.

Models may contain information about the domain, the application, and the implementation. Domain analysis can provide an explicit description of the principles of an application domain such as aerodynamics, kinematics, or economics. Ideally, descriptions would be constructed that

allow rigorous reasoning from these principles. These descriptions would apply to various kinds of software and system applications in the corresponding domains. Furthermore, where software controls a mechanical or information system, the software (as well as the controlled system) is an active system element and its behavior may need to be described in the model.

In some cases, software and engineering models might be used directly for reasoning about systems. Often, however, a fully detailed description may not be needed or may be too large for feasible analysis. The needed reasoning may concern only specific aspects of the system. In this case, simplified or partial models are needed to manage the complexity of analysis. For tractable simulation reduced models that preserve essential properties of interest are often required. Consequently, methods and tools are needed to obtain models at different levels of fidelity or abstraction. Methods are also needed for understanding and managing the relationships between models at different levels.

Status

It is difficult to develop models. Most are developed by hand. This doubles the effort by requiring separate development activity and assurance activity. It also can de-couple development from assurance activities, with the result that assurance plays a lesser, often neglected, role. In general, commercial components are not accompanied by either behavioral descriptions or other models.

Some analytic techniques now exist for deriving models of software behavior, but these are limited. Software models often have large state spaces and can lead to infeasible formal analyses.

Domain engineering is a largely informal, often* ad hoc*, process. Furthermore, little direct use or reuse is made of domain information in downstream tools. Potentially useful engineering models are often put on the shelf after design and are not employed during software and systems engineering. Models that could support both informative simulation and proof are inadequately explored. Because of these difficulties, modeling and formal analysis of domains, software, and system implementations are currently in limited use. As a result, designers and operators of complex systems often do not have accurate abstract views of system and software behavior.

Needed Research

Research is needed to develop rigorous descriptions of application domains and to use such descriptions in languages and tools. Domain models, captured as formal theories, should provide a basis for rigorous reasoning about software and systems. In combination with descriptions of specific application requirements, domain theories and their associated reasoning tools might be used to specialize tool and language technology into design environments that provide the support necessary for building high-confidence applications in the domain. Such tools would provide greater assurance about domain-specific aspects of system behavior.

Research in tools for describing and analyzing system and software behavior is needed. One issue is how to automate the creation of models from software, including software model

derivation from widely used software distribution formats such as Java byte code. Automated abstraction is needed to generate models having feasible analyses while preserving validity and correspondence to the real system. Another research issue is how methods currently used for verification and analysis (e.g., theorem proving, model checking) may supplement informal simulation techniques, and how common models may be generated that could be used for both verification and simulation.

Benefit

The benefit of this research is that the behavior and other properties of systems will become explicit and more available for analysis and prediction. Simplifying the task of modeling would make assurance technology more accessible and usable for developing high-quality software and systems. Automation will reduce human errors and improve the accuracy of models. Successful abstraction and other scaling techniques will enable sound and automated analysis of large systems and will improve human understanding of system design and operation.

This research can also enable the generation ("autocoding") of correct software from system specifications and domain-specific models for certain classes of applications. Furthermore, it can enable semantic visualization that exploits model information to improve designer, evaluator, and operator understanding of systems and software.

2.3 HCSS BUILDING BLOCKS

Rapid engineering of high confidence systems requires a technology base of components with understood high confidence properties and a methodology for assembling those components in such a way that the properties of the resulting system can be understood, reasoned about, and validated. Such components, or building blocks, include hardware, operating systems, middleware, communications, and other run-time services.

Status

Current high confidence systems are largely constructed from scratch, treating each new system as a bottom-up design effort. While off-the-shelf processors may be used, they typically come from product lines targeting niche markets. These products lack significant software, programming, and system development support. They are often programmed by hand in assembly language, making analysis of the correctness and properties of the resulting functional unit and system difficult. Operating systems for high confidence systems are similarly characterized as targeting niche markets, causing them to lag broad-based commercial offerings in functionality and performance. The last decade, for example, has seen the emergence of a robust market for real-time operating systems targeting embedded systems, but these products lack the critical networking support essential to emerging system architectures. Middleware technologies, such as CORBA and Java Virtual Machine, that raise the level of abstraction presented to the programmer, are attracting increasing attention from the HCSS community, but these technologies currently fall short of the mark in performance, overhead, and support for high confidence properties. Where extensions to these technologies for high confidence are being

developed, the existing development processes emphasize stove-piped methods for each property rather than integrated solutions.

Needed Research

A two-pronged approach is needed to make COTS technologies viable in the construction of affordable high confidence systems.

First, reference implementations demonstrating the incorporation of HCSS properties into a new generation of commercially viable components, such as processors, operating systems, schedulers, error handlers, protocols, probes, monitoring modules, policy modules for various security policies, scheduling policies, concurrency policies, and supporting middleware environments must be developed. Successful transitioning of these reference implementations to mainstream product lines will require that such extensions impose minimal interference with existing functionality and performance, and minimally impact cost and time-to-market. Second, middleware technologies that permit the construction of high confidence systems out of components that are not necessarily themselves high confidence must be developed. In spite of our best efforts at improving the viability of COTS components in HCSS, transition, deployment, and adoption is likely to be slow. Even as such products become available, system developers will need to rely on a heterogeneous technology base of robust and untrusted components. Furthermore, as "embedded" systems become networked, developers will find themselves increasingly relying on major systems or subsystems over which they have no control or information. Middleware technologies that can rely on the HCSS properties of lower system layers (where those properties are known and validated) and detect and compensate for the absence of such properties where necessary are central to the construction of next-generation high confidence systems.

Benefits

These technologies will provide the capability of rapidly assembling high confidence systems from commodity components while concurrently demonstrating reference implementations of commercially viable next generation components with inherently stronger high confidence properties. The availability of such a technology base will allow the rapid, affordable construction of high confidence systems.

2.4 ROBUST SYSTEM DESIGN

System design often focuses primarily on the required behavior of the system under nominal (normal) operating conditions. High confidence systems, however, must continue to operate correctly and safely under a variety of adverse operating conditions that can interfere with or compromise functional integrity. Sources of adversity include the physical environment, unanticipated use of the system, and anomalies. Examples of physical phenomena that affect system operation include vibration, temperature, humidity, lightning, electromagnetic disturbances, and (for air vehicles) atmospheric disturbances such as wake vortices, and weather

related adversities such as wind shear, turbulence, and icing. Examples of unanticipated use include human operator error, unexpected inputs, and use of a system or component outside its design envelope. Increasing emphasis on reusable components and interconnectivity can also lead to a system or component being used for a purpose other than that for which it was designed or being required to interact with other systems or components not envisioned at design time. Anomalous conditions range from failure modes such as component, data and timing faults to unusual workload demand, to malicious activity. High confidence systems must often exhibit robustness against a large subset of this broad range of adverse conditions.

As an example, consider a surgeon at an urban medical center remotely guiding a telesurgical procedure at a rural or combat field hospital. In such a case, a high-confidence system is needed to help ensure the functional integrity of the system operations even if a storm causes the hospital to suddenly switch to auxiliary power, or if the network is attacked by a malicious intruder. Similarly, advanced commercial or military aircraft that depend on high-confidence systems must not lose control if a wind shear is encountered or if components such as sensors, actuators, or processors are impaired or damaged.

Status

New performance and survivability demands are arising for many systems. For example, for future airspace systems there is a need to increase airspace capacity, or to increase the number of passengers carried by a single vehicle. Within today's systems and controls discipline, general methods and techniques exist for modeling and analyzing function, performance, and robustness of most dynamical processes. However, current design methods for assuring integrity and performance will not be sufficient to assure future high confidence systems that demand improved performance and survivability under a broad range of operating conditions.

Needed Research

The design fidelity of systems, as well as functional integrity and performance in the specific application, must be guaranteed for both nominal and adverse environments and conditions. Designs should facilitate inclusion of new requirements that impact robustness. In addition to showing that the high confidence system has no design errors and is fault tolerant to specific classes of faults and component failures, it must be shown that these systems perform the intended function and will not malfunction under a variety of conditions. New system mechanisms and abstractions to simplify the designer's task in constructing robust systems are needed. Architectures for these systems must be developed and demonstrated that can reduce vulnerability to adverse environments and conditions. Analysis and validation methods must be developed that enable a system to be assessed in the context of the function it is to perform. Research into a variety of related tools and technologies is needed to address this topic.

System-Level Modeling and Analysis. Currently, practitioners with different specialties separately, and independently, address different types of design issues. Development of the needed modeling methods will require a multidisciplinary approach that includes the merging or correlation of models and techniques for problems such as system identification, state estimation and prediction, uncertainty modeling, parameter dependence, and discrete event transitions.

Software tools are needed that will facilitate the development of these models, which can then form the foundation for design, simulation, analysis, and validation processes. Modeling and analysis techniques and tools developed under other components of this research must be scaled up to capture system level behavior and expanded in scope to accurately capture the operational environment, including failure modes, operational uncertainties, and unanticipated inputs. Techniques are also needed to analyze fault propagation between interconnected systems.

Abstractions and Mechanisms. New abstractions are needed to simplify the incorporation of robustness into high confidence systems. New mechanisms that allow systems to tolerate various types of faults and anomalies while continuing to satisfy other high confidence properties are needed. Existing fault tolerance methods need to be rethought to accommodate operation under real-time and security constraints and encapsulated as reusable (but customizable) services and middleware. Innovative new methods of robustly interfacing software components are also needed to automatically "impedance match" components by ascribing explicit semantics and constraint to interface definitions. Active interfaces, for example, could not only handle mismatches in data types and formats, but could further include mechanisms to address timing and synchronization issues, incomplete data, and propagation of information on the uncertainty or "quality" of the data.

Architecture. Techniques are needed to enable the systematic design of failure or degraded modes of operation that can limit some aspects of function or performance in order to assure system survival. Architectural constructs that allow isolation of faults or malicious behavior and limit propagation of errors or uncertainties are needed.

Human System Interaction. High confidence systems must be developed at every stage with consideration of human-machine interface issues, because the human operator eventually becomes a critical part of the operational system. Methods must also be developed for designing tolerance to human errors or malicious actions into the system. Formal methods could be explored for precisely representing to operators the behavior of systems with complex functions and modes and in supporting automated processes that assist in their training.

Benefit

The main benefit of this research will be reusable design knowledge encapsulated in the form of reusable abstractions and enforcement mechanisms. This HCSS research will lead to robustly designed systems that achieve greater resilience under adverse or hostile operating conditions and will provide improved performance and function for normal operations.

2.5 MONITORING, DETECTION, AND ADAPTIVE RESPONSE

Once a system has been designed, built, tested, and deployed, there is a need to ensure that it continues to operate as a high-confidence system. Employing the technologies and practices developed under the Robust System Design topic will provide a crucial enabling foundation for this, but will be insufficient in ensuring that the global system behavior is maintained. For real-time safety-critical systems, secure systems, and other critical systems, one way to provide this additional level of confidence is to identify measurable system parameters and then develop methods of sensing these parameters to monitor the status of the system. When these parameters

fall outside established bounds, due to inadvertent or deliberate actions, it may be an indication that the system has reached the point where high confidence can no longer be guaranteed. Then, a response may be needed to restore the system to a high confidence state, gracefully degrade it, or halt it.

Real-time monitoring methods must therefore be developed for detection, diagnosis, and prognosis of malfunctions and failures in adverse environments and operating conditions. Monitoring must account for unanticipated events as well as those that can be predicted. Real-time accommodation includes the capability for system reconfiguration to recover from system failures and errors, the capability to adapt to and mitigate adverse environmental conditions, and the ability to provide operator warnings and cues.

Status

Most research in fault tolerance for safety-critical systems has focused on conditions such as failure of a processor or sensor. It also has focused on the assurance provided by fixed levels of component redundancy. For example, identical computations might be carried out on three processors, and the results compared. Often, the comparison is carried out by a voter that determines, for example, that the same result is given by only two of the three processors and that result is used. This entails dedicated processors, which require additional space, power, and weight. Furthermore, if the voter is not also redundant, it becomes the single point of failure. Lacking are fault tolerance methods that can dynamically assess the location of vulnerabilities, and invoke reconfiguration and adaptation to prevent or accommodate faults.[3] A system may not only need to detect and react to each parameter. It may need to monitor a collection of parameters, as well. Consider, for example, the problem of intrusion detection in a secure system.

A great deal of investment in secure systems research has focused on traditional information security mechanisms for confidentiality, integrity, and availability. Simple sets of parameters for security do not adequately describe a highly secure, highly available, and high-integrity system. Security sensors and probes generally work independently of each other, with no communication from one sensor to another. Most sensors and probes are designed to watch for patterns (signatures) in the network, and produce a warning when they are detected. Others utilize statistical analysis or a combination of the two methods. Still others use new, experimental methods. Since there is little or no communication among the sensors and probes, an event detected by one device cannot necessarily be correlated with a related detection by another device. Consequently, while none of a system's parameters individually may exceed threshold, considered collectively they may indicate an out-of-bounds condition. It is also necessary to ensure the integrity of the information reported by a sensor or probe.

[3] National Research Council, *Trust in Cyberspace*, Fred B. Schneider, Editor, National Academy Press, 1999. The full text of the book is available on-line at http://www.nap.edu/readingroom/ records/0309065585.html

Needed Research

Research is needed in the correlation of sensor data for detection and analysis of non-local failures or attacks. For safe control of physical systems, sensor and actuator calibration against predictive models may be needed to detect variations that could arise from physical obstruction or interference in the operating environment. Fusion with other data sets (e.g., patterns of attack) may be required, as well. Visualization techniques are needed to aid in the analysis of data.

Approaches are needed for ongoing evaluation of sensor and actuator effectiveness during system operation. Filtering of data is necessary to limit what and how much data should be collected to maximize the possibility of detecting anomalies. Probes will need to be provided with some degree of real-time data analysis capability.

One profitable area of research might be in how sensors and probes collaborate and share information with each other. This would allow for specialized sensors or probes that would not have to look for all events. Each could have a specific tailored focus. For example, for detecting intruders in a secure system, static sensing may not detect distributed attack strategies that take advantage of the limitations of static sensors. By allowing specialized probes to move freely through the network, the part of a distributed attack that one sensor cannot detect may be detected by another.

Issues include the performance impact of augmenting (e.g., "global") sensors, which should be minimal so they do not significantly degrade performance for legitimate users of the system, and they should be designed to be unobtrusive. These sensors should be easily adaptable to support the level of confidence needed in the system. Techniques are needed that will allow them to protect their own integrity and the integrity of their data.

After data has been collected, processed and analyzed, appropriate response techniques must be employed. Research is needed in toleration and adaptation strategies and techniques. For example, new fault tolerance techniques for high confidence systems might be developed and demonstrated that can evaluate failures or vulnerabilities and provide adaptation, reconfiguration, restructuring, and function migration. Analytical methods for determining and quantifying fault coverage and the resulting performance of such techniques are also needed.

Benefits

The availability of this technology would provide a capability that is lacking in current systems built without monitoring, detection, and accommodation capability. It will enable us to shift our focus away from total reliance on *a priori*, "bullet-proof," system security or safety, which is often unsuccessful. Understanding how to monitor, detect, and respond dynamically to inadvertent or deliberate actions will allow systems to deal with disruptions and intruders while still meeting mission requirements.

2.6 VALIDATION

The term validation, as used here, incorporates the traditional definitions of both "validation and verification" ("V&V"). It encompasses all activities contributing to an assessment of the confidence one can have in a system.

Software and system validation activities may occur sporadically throughout the development process, using methods that vary in completeness. For example, consider an aircraft flight control system (FCS). Once a system design is verified mathematically, a prototype FCS can be built. This prototype must be analyzed and evaluated for correct function in its specific application and for functional integrity under nominal and adverse environments and conditions. To evaluate the FCS at this stage, it might be interconnected to a simulation of the aircraft, engines, sensors, actuators, and atmosphere, as if it were in actual operation. In this case, the FCS issues commands to the simulated control surfaces. The simulation determines the impact on the simulated aircraft and feeds back to the FCS computed input values that would have been provided by actual sensors on the aircraft.

It may be possible to validate some attributes during the early phases of design if appropriate specification methods and tools are used (e.g., live-lock, deadlock, error recovery). Others may not be assessed until a complete implementation is created.

Status

Validating properties of software currently is done by hand, and typically entails developing a test suite that exercises certain properties of the software. Exhaustive testing of all cases is impossible for non-trivial software; hence testing cannot prove the absence of design flaws or execution time faults and failures. Similarly, simulation of system operations under nominal and adverse environments and conditions may encounter an infinite number of conditions that can occur. Simulation and testing may be useful design tools, but exhaustive testing and simulation alone are impractical validation methods for complex high-confidence systems.

The dominant software testing theories and methods are based upon "white box" testing that assumes the program code or other detailed representation of the software module to be available. (This is generally untrue of commercial, off-the-shelf (COTS) software and much legacy software.) White box testing tools typically analyze the control and data flows in the program code to craft a test suite that covers certain paths. Paths are tested by selecting input data that forces their evaluation and checking for the corresponding expected output data. When necessary, "black box" testing must be used. It presumes that only the interfaces are available, and the tests must be conducted by observing input/output exchanges at the interfaces. Systems and integration testing in industry is based primarily on black box testing. Black box and formal methods together have been applied (specifically, composing components using process algebra and state space exploration). Some techniques exist to limit the number of tests to a manageable set. Initial research in using formal methods to limit testing requirements is promising, but not mature. Formal descriptions (e.g., using finite state automata or process algebra models) and verification or model checking are becoming widely-accepted tools for specifying and validating communications protocols. Model checkers are also increasingly used for hardware verification.

Some tools that, e.g., combine testing, analysis and formal verification have been studied[4]. However, most software is validated without the consideration or benefit of formal models and reasoning techniques. Most of these theories and best practices remain beyond the reach of the majority of validation practitioners, typically programmers with ordinary training and skills.

Needed Research

New validation methods, employing HCSS technologies are needed to supplement and reduce the need for costly and time-consuming post hoc testing- and simulation-based processes. Research is also needed that leads to validation methods that systematically combine formal methods and language-based assurance, analysis, testing, and simulation. At the system level, innovative, model-based mathematical methods that evaluate and quantify system properties such as stability, robustness, and performance over a wide range of uncertainties (including the most severe and most probable adverse conditions) could provide much needed contributions to validation.

For example, analysis of extended programming languages (e.g., domain-specific, languages, extended type systems) might generate specifications that can be analyzed for different attributes. Theoretical work in Interoperability (Part 1, Section 3) and Composition (Part 1, Section 4) should lead to research in the exchange, translation, and combination of language and tool capabilities to validate properties. Research recommended in Programming Languages, Tools, and Environments (Part 2, Section 1) should lead to a means of generating code that can be guaranteed to be a correct implementation of algorithms that were designed and analyzed in mathematical design tools.

In this context, the research and development focus needed is on those attributes that may be assessed in the development phases and assisted by automation of design and development tools. For example, automated validation technology is needed that can consider "correct-by-construction" techniques that may have been used in system development, or mechanisms that have been included for fault tolerance and recovery. As one example, methods might be explored that are analogous to proof-carrying code for certifying software that was developed using these techniques and mechanisms.

Benefits

Substantial improvements in validation can be gained by including a wider range of validation methods, in particular formal verification and analysis. Integration of HCSS techniques and tools together with automation over the development cycle will provide a more comprehensive end-to-end validation process, and can help mitigate the level of training required of validators. Reduced cost and time of evaluation is a big benefit. Representatives of the aerospace and aviation industry reported at planning workshops that industry spends 50% to 60% of its budget on V&V.

---

[4] Proceedings of both the Protocol Specification, Testing and Verification (PSTV) (1980, et seq.) and Formal Description Techniques (FDT) (1987, et seq.) workshop series were published by Elsevier Science, North Holland, Amsterdam and New York.

2.7 EVIDENCE AND METRICS

Decisions about the acceptability of systems and about comparisons of the quality of one system with another require supporting evidence. For systems with high confidence requirements, the soundness of the evidence and its evaluation carries critical safety, security, and economic consequences.

Status

Satisfactory quantitative measures for certifying software and systems reliability remain elusive. Historically, statistical reasoning provided the underpinnings for estimating reliability of composite systems. Reliability requirements are formulated statistically, (e.g. average number of accidents per flight hour), then formally decomposed and allocated to different components of the system. Models of the reliability of physical system components are generally based on estimates of component wear and breakage after extensive testing. Some reliability metrics also incorporate statistical properties of replication-based fault tolerance methods. Software reliability modeling has largely attempted to mimic physical statistical reliability models, but with variants such as "reliability growth models" that treat trends in the interval of operation without failure as a measure of software reliability.

Testing typically has been the principal means to establish that software or hardware modules satisfy their requrements. The "level" of required reliability that is allocated to a component (e.g., 10^9 vs. 10^4 failure-free hours of operation) determines the testing effort that must be put into gathering evidence. Substantial research investment has been dedicated to finding methods for generating tests, measuring test coverage, and building test-based evidence for dependability. Certain properties of a single execution, such as throughput, can indeed be measured by a test execution and the figure of merit can be used for comparison with respect to a standard set of benchmarks. However, extrapolating from the set of executions explored by a finite test suite to properties of all executions is unsound: combinations of execution conditions that may remain unexplored could cause failure. Although unit and integration testing are essential activities, and must be performed, certification based primarily on testing is extremely costly. Furthermore, testing that would be sufficient to actually achieve prescribed levels of reliability is generally infeasible.[5] Complex attributes pose the core problem, since functional correctness and crosscutting properties such as safety and security defy valid evaluation using only simple numeric measures of test outcomes. Rudimentary reliability-based testing approaches do not systematically account adequately for coupled failures, where failure of one subsystem may trigger failure of another.

Evaluating reliability is a fundamentally difficult problem. It is impossible to provide general automatic methods to guarantee that programs will not exhibit arbitrary undesirable behaviors. However, some properties can be checked. Correct-by-construction technology is being developed to refine functional specifications into code in a sequence of provably correct steps. Methods and tools that support the automated assessment and integration of desired properties

[5] Ricky W. Butler and George B. Finelli, "The Infeasibilty of Experimental Qualification of Life Critical Software Reliability", ACM SIGSOFT '91, Conference on Software for Critical Systems, New Orleans, Louisiana, December 4-6, 1991, pp. 66-76. The paper may be accessed from the Web at http://techreports.larc.nasa.gov/ltrs/PDF/NASA-91-acm-rwb.pdf.

during the development cycle contribute to dependability. However, current measurement practice that depends entirely on post-development testing does not consider evidence resulting from such development activities.

Needed Research

Research is needed to establish foundations for evidence that can yield consistent, reproducible evaluation. Tools and technology should better integrate engineering and certification processes. Certifiers should exploit evidence produced during system engineering design and development in order to reduce effort, identify errors and omissions, and increase confidence added by the certification activity.

New approaches must be developed that provide evidence based on HCSS technologies. In this context "measurement" does not reduce to a simple metric whose values are expressed in the numerical units traditional for measurement of physical phenomena. More structured types of representation are required to capture the evidence from different assurance activities. The means used to assess a property and establish evidence of its presence within information technologies should consider results from mathematical analyses, verification, model checking, and type checking. Research is needed in how different evaluation methods can be combined efficiently to build support for confidence. For example, verification or model checking can help limit testing effort. Particularly lacking is research that contributes to reasoning about specific complex behavioral attributes and about the composition of systems and evidence for such attributes.

Benefits

This research will yield new methods and technology for validating systems. Integrated development and certification practice with rigorous evaluation standards will pave the way for routine, safe exploitation of information technology in future embedded and complex systems. By using evidence from the full range of HCSS technologies, costly test-based evaluation can be limited, thereby significantly reducing cost in the development lifecycle. This will spur the creation of innovative technologies for next-generation software-centered consumer products and systems.

2.8 PROCESS

Developers, in producing a system, follow either a formal or informal process that dictates the stages in the development, the methods and tools used within each stage, and the human and environment management issues. Confidence in the system produced begins with confidence in the process used to develop it.

Status

At the same time that hardware has gone through constant, rapid change and improvement, software development process has evolved slowly in keeping with the increasing size and complexity of the demands. This same rapid hardware change means that both the hardware and

software components of the system being built are changing independently over the course of system development.

There is today little agreement about process issues because of this rapid change, the wide variety of demands, and the lack of sufficient study of actual practice to determine what causes the successes and the failures. The principles governing the construction of large information systems are as yet not well understood because these systems lack the solid scientific foundation that informs the construction process for physical systems; e.g., buildings and bridges. The development processes used today for IT-based systems are insufficiently predictable and lack the measurements and standards needed to provide high confidence guarantees. While there have been many impressive successes, the failure rate remains astronomically high. To build high-confidence distributed systems that are interconnected with other networked systems remains a challenge. While building today's systems is already challenging, tomorrow's systems will be even more dynamic (both at the hardware and software levels), even more interconnected, and even more exposed to an unknown environment.

Needed Research

Information technology urgently needs a more solid scientific base for the system development process. There is a great need for models of distributed environments and of the process that will develop high-confidence systems in those environments. Empirical study of what works and does not work must be done. With these three in place—a solid scientific base, appropriate models, and empirical studies—the development and implementation of new processes can be done in an informed way, meaningful measures and metrics can be established, and real progress can be made to keep pace with the changing environment. The new process models will incorporate building manageable information systems from components while forging high confidence through engineering discipline and integrated analysis, test, validation and verification methods. The results of all the other research called for, particularly that on composition, decomposition, formal methods, and validation, must be integrated into a number of appropriate process models that produce, along with a product, a confidence rating that is meaningful and acceptable to the targeted user community.

Benefits

The best tools and techniques cannot guarantee high confidence without a development process that assures use of the tools, adherence to standards, appropriate measurement, and validation. Confidence in the tools and techniques will not be achieved without improvement in the scientific basis for information-system construction. The benefit of this research is, therefore, increased understanding of the construction methods that we manipulate and the ability to integrate the results of other research into a working whole, involving people, platforms, tools, environments, and management. Without successful research in this area we cannot expect consistent, repeatable development of high-confidence systems over a broad spectrum of applications.

In order to fully realize these benefits significant changes in engineering practice will be necessary, for example development teams may need to be restructured to include expertise in

formal methods in addition to designers, coders, testers, and analysts. Furthermore, process steps must be adapted to take advantage of complementary technologies such as formal methods and testing.

PART 3: HCSS ENGINEERING AND EXPERIMENTATION

In order to successfully implement this research, it is necessary that the resulting technology be hardened for practical use and that it be evaluated experimentally. The purpose of the HCSS Engineering and Experimentation component is to develop robust public domain reference implementations and to conduct demonstrations of High Confidence Software and Systems technologies for significant applications. These efforts would aid in technology transition by illustrating both high-confidence systems solutions and the assurance evidence produced during development. The following sections outline areas in which reference implementations and application experiments would be particularly useful. These should be understood to be examples of work that would be suitable for this component rather than a precise list of topics to be undertaken.

3.1 SOFTWARE CONTROL OF PHYSICAL SYSTEMS

The flexibility of software has resulted in both greatly enhanced system capability and greatly increased dependence on software correctness. Technological advance depends on information technology and increasingly complex control systems. For example, highly coordinated operation and mode transition are required for physically coupled subsystems such as fuel delivery, steering, and braking in an automobile. This entails coordination of operating region feasibility constraints and cooperative management of dynamically changing control authority. The feasibility of advanced national capabilities such as intelligent vehicle highway systems or the high-speed civilian transport and the safety, quality, and competitiveness of electronically controlled U.S. consumer products, medical devices, etc., depends critically on high-confidence control systems that can exploit software for more robust and capable systems.

Status

However, an engineering gap (and consequently, an assurance gap) exists between the physical design of an aircraft, train or automobile and the design of the software that controls the system. For example, a controlled system may be required to perform in several modes of operation. During engineering development the set of control laws that manage system operation within a single mode are systematically designed. Computer-assisted design environments are able to generate software that correctly implements the mathematics of these laws. However, the software logic to manage the transition of the system among modes, the coordination of interacting controllers, and the interaction with operators, lacks this sound engineering basis. Control systems typically are implemented in an inflexible hierarchy of control loops that make adaptation and reconfiguration difficult. Controller code is notorious for being poorly structured and difficult to understand.

Needed Research

Reference systems building blocks are needed to enable routine, high-confidence implementation of control systems. The systems technologies should have a rigorous basis and robust design. For example, recent research in hybrid systems provides a theoretical framework that combines formal descriptions of the continuous and discrete (logical) properties of a system, and enables reasoning using rigorous design tools such as model-checkers, verifiers, and analysis tools. Implementation technology is needed for hybrid systems. Approaches are also needed that systematically transfer information and models used in designing physical systems into rigorous system implementations.

Benefit

This research will provide reference implementations of software system building blocks that can be used in high confidence control systems. It will also integrate technologies for reasoning about and building digital control systems, providing software control assurance that is grounded in the physical design of the system to be controlled. This research will illustrate high-confidence implementation technology for a ubiquitous and technologically critical class of software. The resulting technology will enable a wide range of complex, high-confidence, future systems.

3.2 HARDWARE AND SOFTWARE PLATFORMS

The research challenges being addressed by this HCSS engineering experiment are in the area of high-assurance hardware and software platforms. The goal is to develop an HCSS reference implementation for a hardware verification environment based upon a loosely integrated collection of existing tools such as cycle simulation, model checking, theorem proving and other automated analysis techniques that accept designs specified in a hardware design language (e.g., VHDL) as their input. This body of tools will provide a practical proof-of-concept for hardware verification. The result will unify ongoing work at a number of institutions and will provide a common environment that is crucially needed in the development of a successful methodology for formally verified design.

Status

The cost of hardware design errors becomes greater as more functionality is integrated onto a single chip. This cost is measured in terms of increasing costs for engineering changes, and in the increasing proportion of engineering resources devoted to simulation in order to find and correct design errors before fabrication. This cost is now becoming acute, as design teams often have several verification engineers for each design engineer. Most important is the cost of delayed production due to increased engineering effort required for design verification.

The problem in developing high-assurance hardware designs rests in the increasing complexity of hardware implementations. While the conceptual intent of the hardware component is still relatively clear (e.g., encryption or signal processing), the implementation of these components in current design technologies is becoming increasingly difficult.

Work in the hardware verification area is ongoing. Industry is focusing on equivalence and model-checking methods that are able to provide high-assurance at a conceptually low level of design. Academia is developing general theorem proving technology for hardware design that may be transferable to industrial practice in the near future.

- Model checking is very effective when the design is represented at the 'bit' level, or when a related bit-level design is available. It has the advantage that the check is automatic, and useful properties of the design can be checked.
- Theorem proving methods have the advantage that they provide a detailed check on the designer's reasoning, and that these checks can be carried out at any level of abstraction. So representations of the "same" design can be verified correct from top-level down to a detailed gate level with assurance that integrity is maintained between levels.

Formal verification tools allow a designer to prove that a circuit design meets a given specification for all possible input patterns. In many cases they also provide a faster "debugging" capability. The difficulty is that formal verification algorithms and techniques are becoming increasingly domain specific, and there are strong mathematical reasons for believing that this must be so. Given that verification tools are becoming increasingly specialized toward one aspect of design or another, the engineer wanting to benefit from formal verification is faced with a significant problem. That is, although the engineer has many tools available that can solve specific aspects of the overall verification problem (e.g., arithmetic circuits or bus protocols); these tools cannot be used effectively in concert. This is because each tool requires the design to be represented in a specialized form that makes explicit the information the tool needs in order to do its work.

The effort proposed herein is not being performed by industry, as they do not address the verification of hardware designs in a standard design language, such as VHDL, from a high-level Instruction Set Architecture (ISA) down to the Register Transfer Level (RTL). Furthermore, current industrial research in formal methods applied to hardware design is centered on techniques that are not as thorough or rigorous as those envisioned to be developed, under HCSS.

Needed Research

This effort will focus on the integration of formally based tools into a unified hardware verification environment. Such an environment will make available to the hardware designer three types of tools that support formal specification and design verification: design-rule checkers, model checkers, and theorem provers. Perhaps more important than the tools is the construction of enhanced design libraries that include for each reference component, a specification, one or more implementations, and assurance that design rules have been met, and that the designs satisfy the specification.

The effort will demonstrate that HCSS technologies can efficiently be made part of the hardware design process by continuing to develop techniques based upon an IEEE standard design language, VHDL. Such a common hardware-description language for verification, synthesis, and simulation tools will allow existing verification and synthesis algorithms, and

future techniques, to be applied to the verification of a single design. Using formal verification technology allows support for any number of VHDL-based design tools and methods. Adding support for formal reasoning to practical digital system design and unifying the addition with existing CAD tools requires that flexible interfaces be provided so that design data can flow between tools easily. This must be done without requiring the designer learn about the particular data format and support standard hardware description languages such as VHDL and Verilog, both IEEE standard languages. In the future, additional verification or automated design tools, particularly those supporting verified hardware-software interaction (*co-design for high assurance*), should be investigated and treated in the same manner.

This effort will not be performing basic research in formal methods. Formal verification techniques have been successfully applied in a variety of example verification efforts and will serve as the basis for this effort. The measure of success for integrating formal verification with CAD tools will be the ease with which hardware designers familiar with current hardware technologies can incorporate formal methods in their work.

Benefits

Completion of this effort will provide a method and a reference implementation for formally based design verification and implementation of hardware components. It will enable future systems that exploit reconfigurable hardware, and will enable better hardware-software co-design. The methodology embodied in this reference implementation would provide increased assurance in hardware design that could be used by commercial hardware producers to achieve:

- Verification at all levels of design abstraction
- Decreased engineering cost and time-to-market
- Faster adoption of new tools & techniques
- Ability to leverage tool efforts of many researchers
- Greater cooperation among researchers on advanced techniques

3.3 HIGH MOBILITY SYSTEMS

High mobility systems are characterized by large numbers of nodes (ranging from small devices to large systems) communicating over heterogeneous networks (ranging from high-speed optical fibers to low-bandwidth wireless and satellite links). Such systems are becoming increasingly important to high confidence applications such as transportation (intelligent highways, train control), medicine (remote patient monitoring, remote surgery), robotics, and national security (small unit operations, mine countermeasures).

Status

Mobile networked devices are becoming ubiquitous in society and the field of mobile computing has spawned considerable research in recent years. While this research has led to

technology offering improvements in bandwidth and reliability as well as support for quality-of-service and disconnected operation, attention has been focused primarily on exchange of information between mobile devices. As the mobile computing landscape expands to include sensors, actuators, and control systems that must operate under stringent real-time, security, and reliability constraints, the existing technology base will prove to be inadequate to support the requirements of high-confidence mobile systems. Aside from the low bandwidth and intermittent connectivity issues, the highly dynamic nature of such systems (e.g., ad hoc networking, sensitivity to environmental disturbances) poses special challenges for high confidence.

Needed Research

Critical issues include the support for real-time control over potentially interruptible connections; rapid, secure reconfiguration through ad hoc networking techniques; and dealing with unreliability due to power constraints and environmental disturbances. Such issues must be addressed through: adaptation strategies that allow reconfiguration in the mobile environment; innovative uses of redundancy and work partitioning; and application-level solutions that allow continued, autonomous, real-time operation during periods of disconnection (e.g., through extrapolation of previous sensor data streams).

Benefits

This task will demonstrate that high confidence design principles and methods can be effectively employed in the challenging environment posed by highly mobile systems, admitting a range of new applications such as remote surgery and cooperating mobile robots that were not previously conceivable.

PART 4: HCSS DEMONSTRATIONS AND PILOTS

Government services and systems are obliged to protect the privacy and safety of citizens. Federal agencies might engage in a Pilots and Demonstrations category of research in order to further adoption of high confidence technologies in Federal systems. The purpose of this component of the research would be to spur application of high confidence technologies to critical systems in the Federal government. ITR&D research agencies and other Federal agencies could partner to fund pilot projects critical to digital government or to an agency's mission.