N95- 31249

GENERALIZED IMPLEMENTATION OF SOFTWARE SAFETY POLICIES[†]

John C. Knight
knight@virginia.edu

Kevin G. Wika
wika@virginia.edu

Department of Computer Science
University of Virginia
Charlottesville, VA 22903

(804) 924-7605

An Abstract Submitted to:

Nineteenth Annual Software Engineering Workshop
Software Engineering Laboratory
Goddard Space Flight Center
Greenbelt, MD

†. Supported in part by the National Science Foundation under grant number CCR-9213427 and in part by NASA under grant number NAG1-1123-FDP.

Introduction

As part of a research program in the engineering of software for safety-critical systems, we are performing two case studies. The first case study, which is well underway, is a safety-critical medical application. The second, which is just starting, is a digital control system for a nuclear research reactor. Our goal is to use these case studies to permit us to obtain a better understanding of the issues facing developers of safety-critical systems, and to provide a vehicle for the assessment of research ideas.

The case studies are not based on the analysis of existing software development by others. Instead, we are attempting to create software for new and novel systems in a process that ultimately will involve all phases of the software lifecycle. In this abstract, we summarize our results to date in a small part of this project, namely the determination and classification of policies related to software safety that must be enforced to ensure safe operation. We hypothesize that this classification will permit a general approach to the implementation of a policy enforcement mechanism.

The Problem

The functionality demanded by modern applications, including safety-critical applications, frequently leads to software that is very large and complex. Functionality requirements have increased because of the many benefits of computer-based control and the availability of inexpensive yet powerful computing hardware. Hardware performance limits that formerly restricted software complexity are rarely reached because of the remarkable hardware performance now available.

Unfortunately, significant software defects tend to remain in such systems after deployment despite extensive effort on the part of the developers [2,6]. Building these systems to perform as desired is very difficult for a number of reasons. Even the best software development processes cannot ensure that faults are avoided completely during development. Similarly, fault detection techniques are imperfect. Research has shown, for example, that testing as an approach to verification cannot demonstrate sufficient levels of dependability because of the sheer number of tests that are required [1].

Even building very small, simple software systems that achieve the extreme dependability necessary for safety-critical applications has proven to be very challenging. Formal techniques have made substantial progress and have been applied to real systems in a number of cases, but their application to large, complex systems remains mostly impractical. The complexity of large systems involving characteristics such as real-time operation and distributed processing is likely to preclude any significant assurance that the systems meet desired dependability goals if traditional techniques are used in traditional ways.

A central question that arises is how to deal with a software system that is on the one hand safety critical and on the other hand large and complex, i.e., so large and complex as to preclude a complete attack on the problem of showing dependability using even the best available techniques. We outline an approach to this problem that we are pursuing in the next section.

Technical Approach

An approach that has been tried in many safety-critical systems is to isolate the problem of ensuring safe operation so that a small part of the software, often termed a *kernel*, is responsible. This is the approach we are following but we are attempting to develop a general, comprehensive approach to the problem by exploiting an analogy with security kernels.

Security kernels are used to enforce access-control policies in classified information systems. The idea of trying to exploit this technique to implement safety rather than security, i.e., the concept of a more general *safety kernel*, was proposed by Rushby [5,7], among others. The idea that Rushby suggested is different from other architectures described as safety kernels because certain essential safety policies are enforced regardless of the actions of the application software. This is in direct analogy with security kernels that enforce access control with a similar degree of generality. Other safety-kernel architectures that have been developed tend to provide a set of services that enforce required safety policies, *if used appropriately by the application*. This is a critical distinction.

The safety-kernel idea is of value if it is able to enforce a suitably large subset of the required safety policies. A major benefit would be gained if this safety-kernel approach could be implemented in a reusable manner, i.e., in such a way that the same safety kernel implementation could be used in a variety of applications. To evaluate the safety-kernel idea, assess its utility, and try to get some insight into generality that might be possible in an implementation we have analyzed the two case studies at our disposal. We report our results in the next section.

Empirical Results

We began this study by identifying the safety policies required by each application. We then examined the two sets to ascertain whether general classes of policies existed and whether the policies were similar in the two cases after application-dependent parameters were removed. We begin this section by summarizing the important details of the two applications and list examples of the safety policies they require. We then discuss the resulting structure of the policies and its implication on implementation and generality.

Magnetic Stereotaxis System

The first case study that we are engaged in is the *Magnetic Stereotaxis System* (MSS). This is an investigational device for performing human neurosurgery being developed in a joint effort between the Department of Physics at the University of Virginia and the Department of Neurosurgery at the University of Iowa [3,4]. The device operates by manipulating a small permanent magnet (known as a "seed") within the brain using an externally applied magnetic field. The patient is positioned at the center of six superconducting electromagnets. Under the direction of the computer, power supplies and current controllers regulate the electric current in the electromagnets thereby producing the magnetic field that acts on the seed. Along each axis perpendicular to the patient's body, an X-Ray source and camera produce fluoroscopic images for tracking the seed. By varying the magnitude and gradient of the external field, the seed can be moved along a non-linear path and positioned at a site requiring therapy, e.g., a tumor.

When the MSS is in operation, there are a large number of events that could lead to patient injury. The complete set is determined by a hazard analysis including the use of techniques such as system fault-tree analysis. Events that could lead to patient injury include failure of current controllers, X-Ray overdose, incorrect calculation of currents for a seed movement, and failure to respond promptly to an increase in seed velocity. Each of these could be the result of numerous different faults, and, in fact, the software could either initiate or prevent many of these failures. Such failures can be prevented irrespective of their cause and irrespective of the state of the equipment if safety policies such as the following (stated here informally) are enforced:

- If the seed moves faster than 2.0 mm/sec, the coil currents must be set to zero.
- If the vision system cannot locate the seed while it is being moved, the coil currents must be set to zero.
- The currents must be within 5.0 A of the value predicted by the coil control model.
- The current requested of a controller must be in the range -100 A to +100 A.
- Before moving the seed, a reversal check must be executed to ensure that the requested currents provide the desired direction within 5 degrees.
- An X-Ray device must be "off" for 0.2 sec before an "on" command is executed.
- The total X-Ray dose during an operation must be less than 100 millirem.

In the MSS system, a total of 42 safety policies have been identified. They are all similar in complexity and breadth to these examples.

University of Virginia Reactor

The target of the second case study is the nuclear research reactor currently operated by the University of Virginia. It is a 2 MW thermal, concrete-walled pool reactor. It was originally constructed in 1959 as a 1 MW system, and it was upgraded to 2 MW in 1973. Though only a research reactor rather than a power reactor, the issues raised are significant and can be related easily to the problems faced by full-scale reactor systems.

The system operates using 20 to 25 plate-type fuel assemblies placed on a rectangular grid plate. There are three scramable control rods, and one non-scramable regulating rod that can be put in automatic mode. The primary process variables that are measured are: 1) Gross output, by movable fission chamber; 2) Neutron flux, by ion chamber; 3) Start-up neutron flux and period, by BF_3 counter; 4) Core inlet and outlet temperatures, by thermocouples; 5) Primary system flow, by pressure gauge; 6) Control and regulating rod positions, by potentiometer; 7) Gross gamma-ray dose, by ion chamber; 8) Various limit set switches to monitor pool level, etc.

As with the MSS, there are a large number of events that could lead to a reactor accident with the potential to cause extensive damage. Some examples of events that could result in hazards include uncontrolled withdrawal of the reactor control rods, loss of water in the reactor pool, failure of a coolant pump, and high radiation levels outside of the reactor pool. Again as with the MSS, such failures can be prevented irrespective of their cause if safety policies such as the following (again stated informally) are enforced:

- The control rods must not be withdrawn at a rate faster than 1.5 mm/sec.
- When control parameters are adjusted, the state of the reactor must respond to reflect the control settings.
- The position of the regulating rod must be adjusted at least once per second based on the power output of the reactor.

If any of the following conditions is true, the control rods must be scrammed:

- A safety channel indicates a power greater than 125% of maximum power.
- The flow in the primary cooling system is below 3,400 liters/min (900 gals/min).
- The reactor inlet water temperature exceeds 105° F.
- The pool level falls below 19 ft. 3 1/4 in.
- The radiation at the reactor face exceeds 2 mR/hr.

A preliminary identification of the safety policies in this application revealed a total of 43 safety policies. As detailed requirements analysis proceeds, this number is likely to rise.

Once the initial sets of safety policies had been identified for the two applications, we focused on identifying common characteristics both within and between the two applications that might permit a logical organization of the two sets of safety policies. We were seeking insight into what might be a general case in order to permit us to begin consideration of a general, reusable, safety kernel. After examining a variety of possibilities, the characteristic that permitted the most complete and systematic classification of the policies was based on the origin and derivation of the safety policies.

Safety policies such as the examples above result from the system safety analysis, and specify safety requirements that must be met by the various system components. In a system safety analysis, a set of mishaps are identified along with hazards that could cause the particular mishap. Each hazard is in turn placed at the root of a system fault tree and the failure conditions that could result in the hazard are analyzed. The exact form of a fault tree depends on the hazard being considered and the details of the particular application. However, we have identified a canonical fault-tree pattern for computer-controlled devices, and we have been able to classify failure conditions according to their location and purpose with respect to the canonical fault tree. We are thus able to classify safety policies according to which type of condition the policy addresses, and this has yielded the following *general* categories of policies:

System operation	Device operation
Device failure	Device input from software
Software error	Failure response
Software input	Operator input to the software
Sensor input	Configuration or application data
Operator error	Operator information

Subsequent re-analysis of the two complete sets of safety policies from our two case studies has shown that the various policies fit into the taxonomy very well. Thus, although the applications are very different, their requirements for safe operation are remarkably similar in basic form and differ to a large extent only in application-specific detail. Though important, these details can be viewed as parameters that can be used to tailor a general implementation strategy, i.e., a general-purpose safety kernel operating in a manner analogous to a security kernel.

A safety kernel prototype is being developed that will enforce policies from the first six categories of policies identified above. These are policies that originate near the top of the canonical fault tree discussed above. They have been selected for enforcement because they are most closely associated with the operation of the application devices. It is the devices that actually cause a mishap, so it makes sense to enforce safety policies that are directly related to devices. Policies from the other classes were omitted because the benefits were not as great and the pragmatic issues of quality assurance, cost, and functional performance would be adversely affected by enforcing policies from these classes.

Conclusions

Based on the two systems we have been studying, it appears to be the case that a great deal of structure exists in the safety policies that have to be enforced. Given this situation, there seems to be a strong possibility that a reusable safety kernel operating independently of the application in a manner analogous to the operation of a security kernel can be built. Such a kernel would permit execution-time enforcement of selected safety policies for systems too complex to verify by traditional means.

Acknowledgments

This work was supported in part by the National Science Foundation under grant number CCR-9213427, and in part by NASA under grant number NAG1-1123-FDP.

References

1. Butler, R. W. and G. B. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Transactions on Software Engineering*, Vol. 19-1, pp. 3-12, Jan. 1993.

2. Garman, J. R., "The Bug Heard 'Round the World," ACM Software Engineering Notes Vol. 6-5, pp. 3-10, October 1981.

3. Gillies, G. T. et al, "Magnetic Manipulation Instrumentation for Medical Physics Research," *Review of Scientific Instruments*, Vol. 65-3, pp. 533 - 562, March 1994.

4. Grady, M. S. et al, "Preliminary Experimental Investigation of *in vivo* Magnetic Manipulation: Results and Potential Application in Hyperthermia," *Medical Physics* Vol. 16-2, pp. 263 - 272, Mar/Apr. 1989.

5. Leveson, N. G., T. J. Shimeall, J. L. Stolzy and J. C. Thomas, "Design for Safe Software," in *Proceedings AIAA Space Sciences Meeting*, Reno, Nevada, 1983.

6. Neumann, P.G., Editor, "Risks to the Public". Software Engineering Notes.

7. Rushby J., "Kernels for Safety?," in *Safe and Secure Computing Systems*, T. Anderson Ed., Blackwell Scientific Publications, 1989, pp. 210-220.

GENERALIZED IMPLEMENTATION OF SOFTWARE SAFETY POLICIES*

John C. Knight Kevin G. Wika

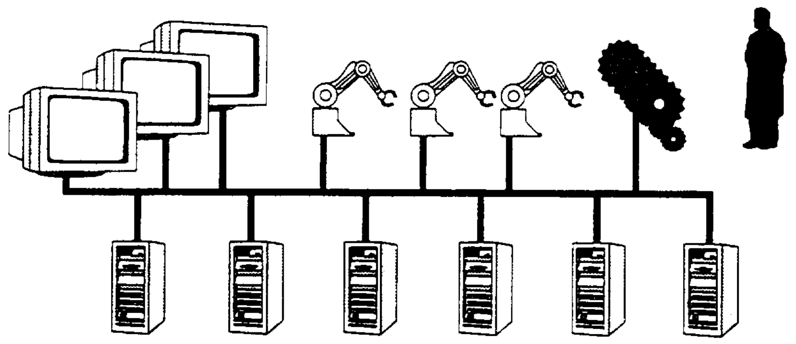
Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903

* Work sponsored in part by NASA, the NSF, the NRC & Motorola Inc

UVA
Department of Computer Science

THE PROBLEM WE FACE

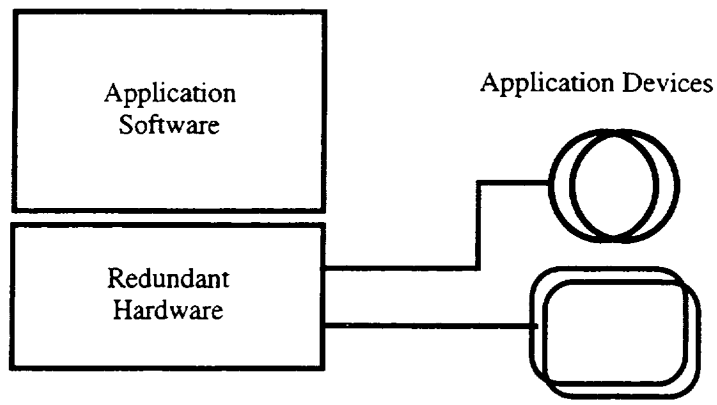
- Software Is *Large And Complex* In Many Safety-critical Systems:



- Huge Subsystems, E.g. System Services, Windowing, The Application, Etc.
- How Do We Build Safety-critical Software That Is:
  - Dependable?
  - Cost-effective?

UVA
Department of Computer Science

NAIVE SYSTEM ARCHITECTURE

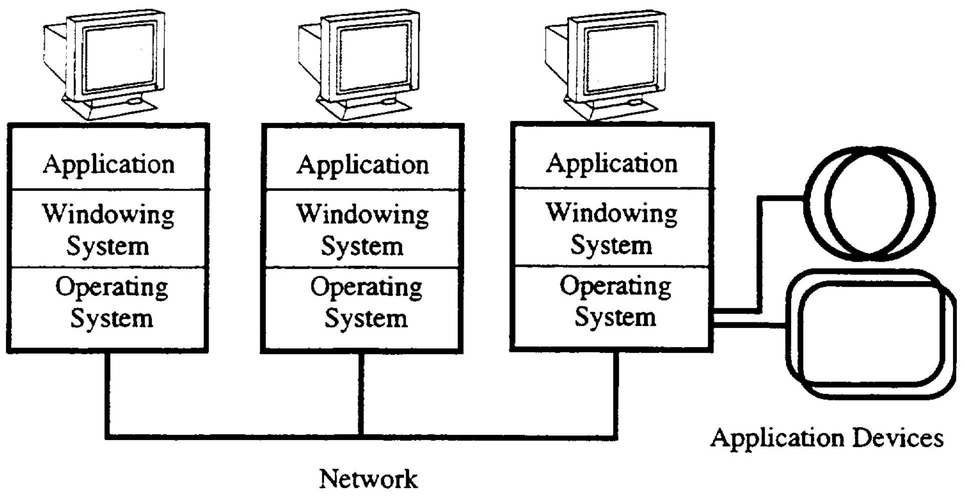


- Keep It Simple, S*****

NASA GSFC/SEL - 94 - Slide 3 (© John C. Knight 1994)

UVA
Department of Computer Science

REALISTIC APPLICATION ARCHITECTURE

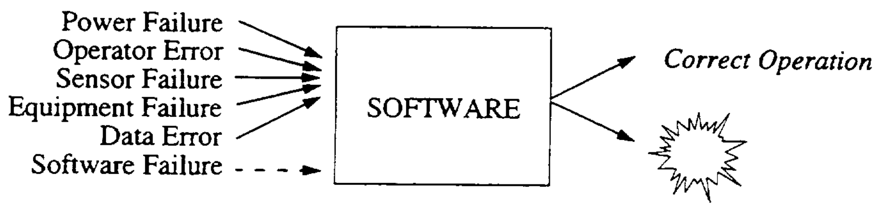


- Diverse Hardware, Network, High-performance Displays
- Extensive, Diverse And Unreliable Software, Perhaps Off-the-shelf

NASA GSFC/SEL - 94 - Slide 4 (© John C. Knight 1994)

UVA
Department of Computer Science

SAFETY REQUIREMENTS AND SAFETY POLICIES



- Safety Requirements Can Often Be Expressed As *Safety Policies*
- Safety Policies — Policies That "Software" Must Enforce To Avoid Hazard
- Policies Such As The Following (From A Nuclear Reactor):

  If the flow in the primary cooling system is below 3,400 liters/ minute, a scram must occur.

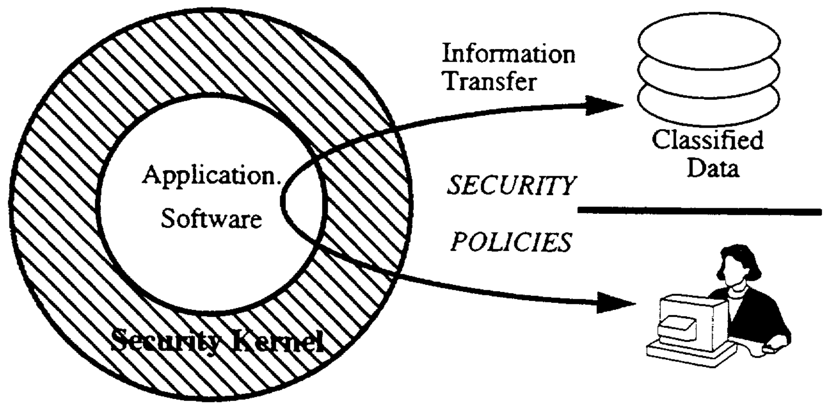
  The source range must be indicating at least 2 counts/second before a safety rod can be withdrawn.

How Do We Ensure Enforcement Of Safety Policies?

UVA
Department of Computer Science

SECURITY KERNEL CONCEPT

- Concept Is That Security Kernel Controls Access To All Information
- Kernel Enforces A Set Of *Security* Policies Irrespective Of Application Software's Actions:

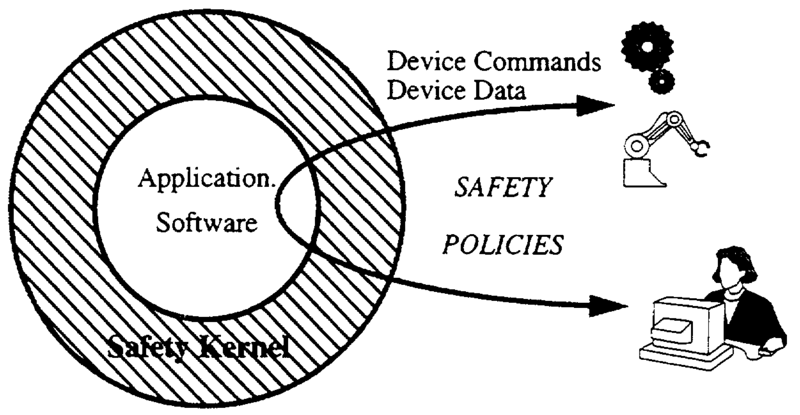


- Might A Similar Approach Work For Safety (Rushby, 1989)?

UVA
Department of Computer Science

SAFETY KERNEL CONCEPT

- Concept Is That Safety Kernel Controls Access To All Devices
- Kernel Enforces A Set Of *Safety* Policies Irrespective Of Application Software's Actions:

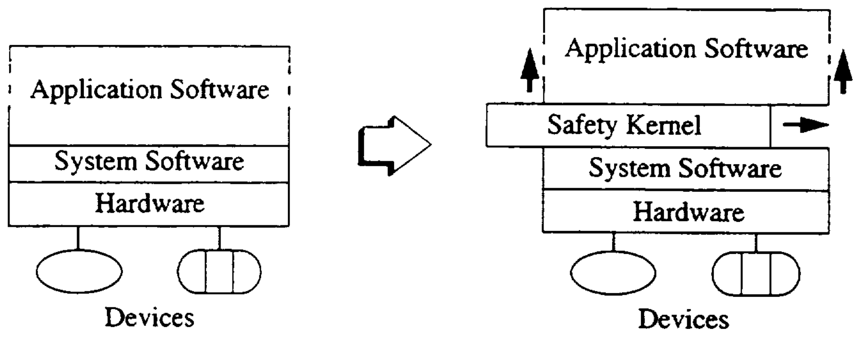


- A Similar Approach Appears To Work For Safety

UVA
Department of Computer Science

NASA GSFC/SEL - 94 - Slide 7 (© John C. Knight 1994)

SAFETY KERNEL

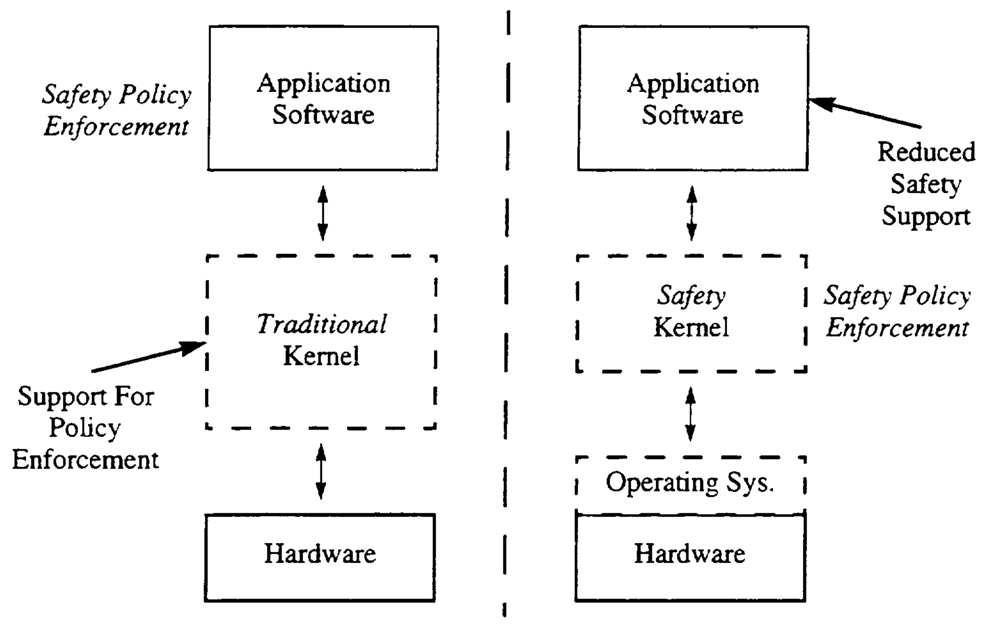


- Policy Enforcement Given To Smallest, Simplest Kernel Possible
- Kernel Controls Access To All Devices Thereby Controlling Effect Of Software
- Policy Enforcement:
  - Certain Important Policies Entirely Enforced By Kernel
  - Enforcement Support For Other Policies

UVA
Department of Computer Science

NASA GSFC/SEL - 94 - Slide 8 (© John C. Knight 1994)

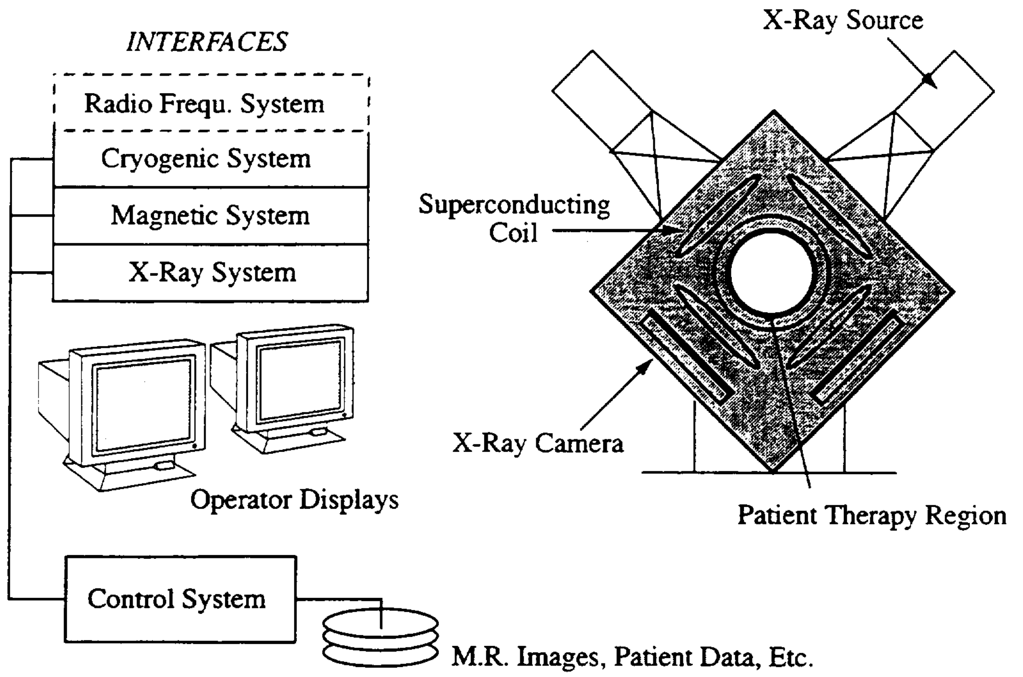
"TRADITIONAL" KERNEL vs. SAFETY KERNEL



UVA
Department of Computer Science

NASA GSFC/SEL - 94 - Slide 9 (© John C. Knight 1994)

CASE STUDY - MAGNETIC STEREOTAXIS SYSTEM



UVA
Department of Computer Science

NASA GSFC/SEL - 94 - Slide 10 (© John C. Knight 1994)

SOME OF THE MSS SAFETY POLICIES

If the seed moves faster than 2.0 mm/sec., the coil currents must be set to zero.

The coil currents must be within 5.0 amps of the predicted value.

The coil current requested by the application must be within the range -100 amps to 100 amps.

An X-ray source must be "off" for 0.2 seconds before an "on" command is executed.

The total X-ray dose during an operation must be less than 100 millirem.

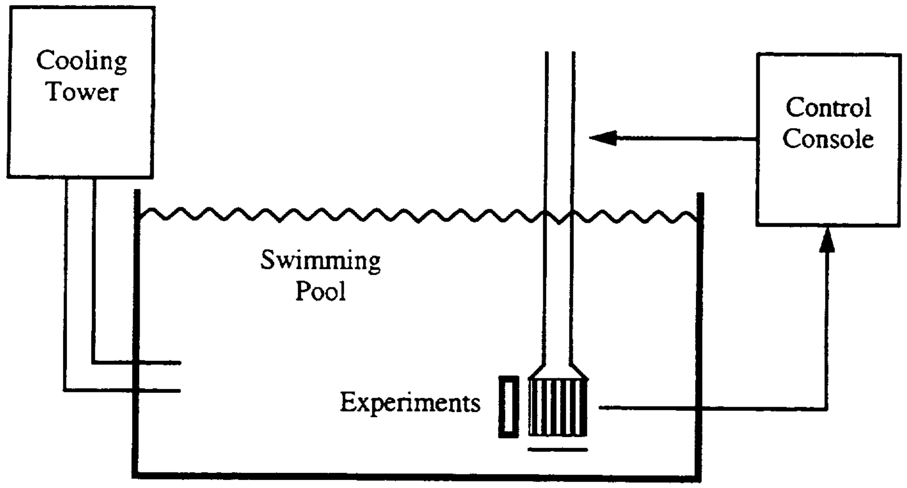
Before moving the seed, a reversal check must be executed on the requested currents to compare the predicted force with the desired force.

And so on.....

NASA GSFC/SEL - 94 - Slide 11 (© John C. Knight 1994)

UVA
Department of Computer Science

CASE STUDY - UVA RESEARCH REACTOR



NASA GSFC/SEL - 94 - Slide 12 (© John C. Knight 1994)

UVA
Department of Computer Science

SOME OF THE REACTOR SAFETY POLICIES

The control rods must not be withdrawn at a rate faster than 1.5 mm/sec.

The position of the regulating control rod must be adjusted at least once per second based on the power of the reactor.

The control rods must be scrammed if a safety channel indicates a power level greater than 125% of the authorized maximum.

The control rods must be scrammed if the pool water level falls below 19' 3.25".

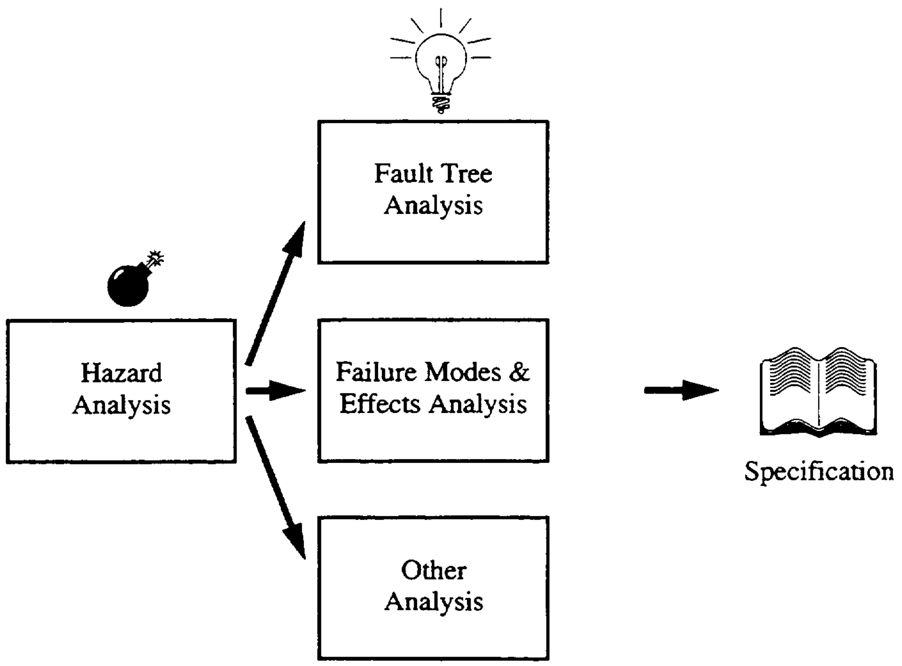
The control rods must be scrammed if the inlet water temperature exceeds 105^{o} F

And so on.....

NASA GSFC/SEL - 94 - Slide 13 (© John C. Knight 1994)

UVA
Department of Computer Science

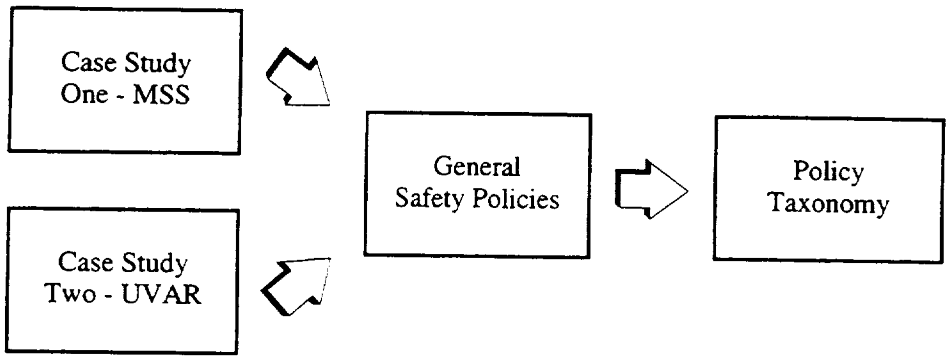
SAFETY POLICY DEVELOPMENT



NASA GSFC/SEL - 94 - Slide 14 (© John C. Knight 1994)

UVA
Department of Computer Science

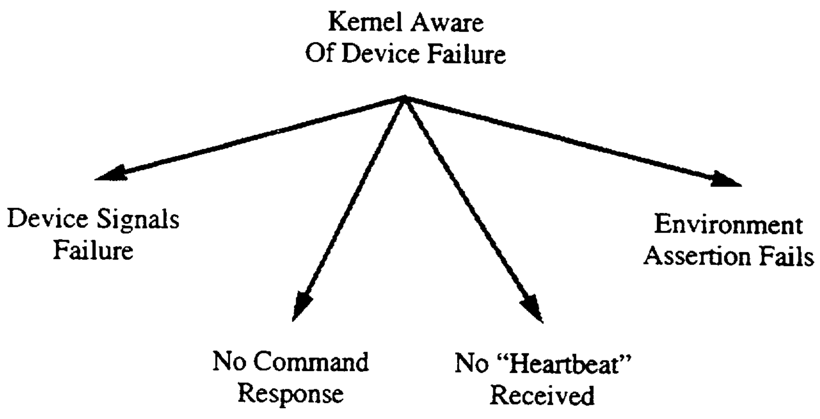
TAXONOMY - GENERAL SAFETY POLICIES



NASA GSFC/SEL - 94 - Slide 15 (© John C. Knight 1994)

UVA
Department of Computer Science

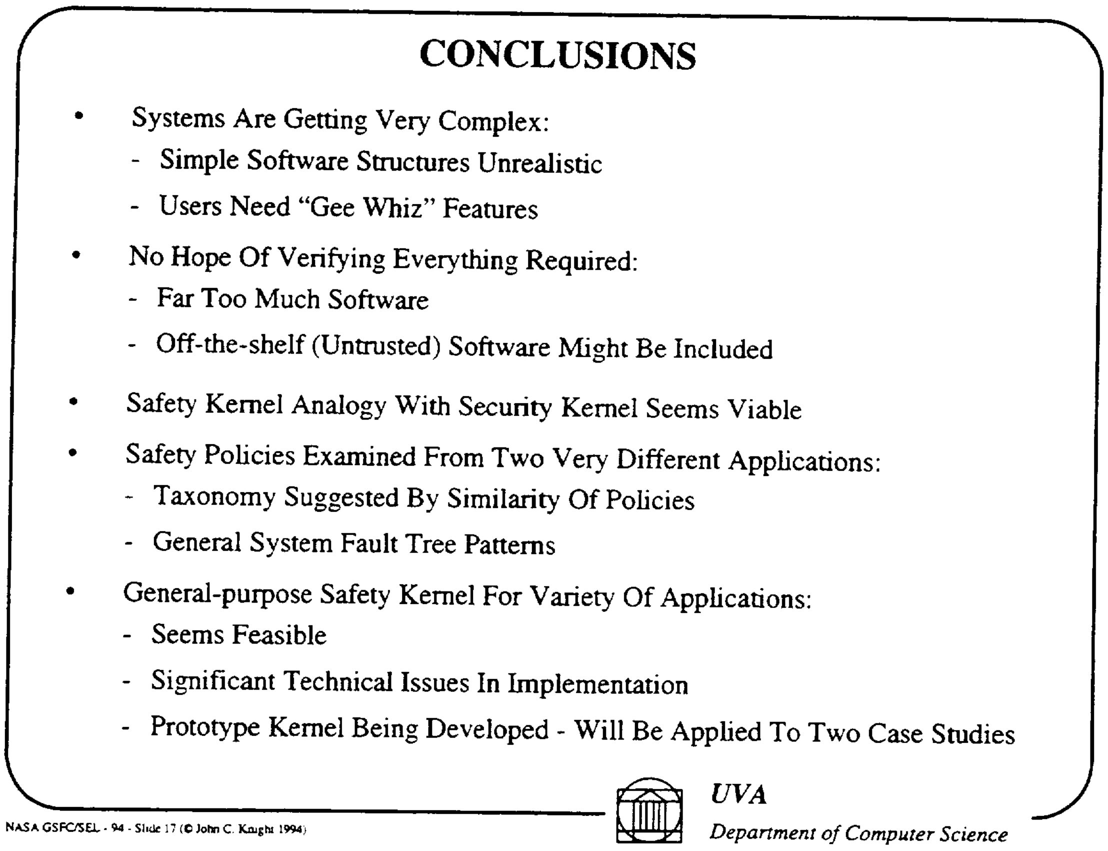
EXAMPLE - DEVICE FAILURE DETECTION



- Captures Essential Content Of "All" Device Failure Detection Policies
- Parameterized Implementation In Reusable Safety Kernel
- Follows From Generalized System Fault Trees

NASA GSFC/SEL - 94 - Slide 16 (© John C. Knight 1994)

UVA
Department of Computer Science



CONCLUSIONS

- Systems Are Getting Very Complex:
  - Simple Software Structures Unrealistic
  - Users Need "Gee Whiz" Features
- No Hope Of Verifying Everything Required:
  - Far Too Much Software
  - Off-the-shelf (Untrusted) Software Might Be Included
- Safety Kernel Analogy With Security Kernel Seems Viable
- Safety Policies Examined From Two Very Different Applications:
  - Taxonomy Suggested By Similarity Of Policies
  - General System Fault Tree Patterns
- General-purpose Safety Kernel For Variety Of Applications:
  - Seems Feasible
  - Significant Technical Issues In Implementation
  - Prototype Kernel Being Developed - Will Be Applied To Two Case Studies

NASA GSFC/SEL - 94 - Slide 17 (© John C. Knight 1994)

UVA
Department of Computer Science

C-4.

Session 6: Measurement

A Quantitative Comparison of Corrective and Perfective Maintenance
Joel Henry, East Tennessee State University

Does Software Design Complexity Affect Maintenance Effort?
Christopher Lott, University of Kaiserslautern

Profile of Software Engineering Within NASA
Craig Sinclair, Science Applications International Corporation

PRECEDING PAGE BLANK NOT FILMED