Software Engineering Institute

Results of SEI Independent Research and Development Projects

Len Bass, Paul Clements, Dionisio de Niz, Peter Feiler, Matthew Geiger, Jeffrey Hansen, Jörgen Hansson, Scott Hissam, James Ivers, Mark Klein, Karthik Lakshmanan, Gabriel Moreno, Daniel Plakosh, Raj Rajkumar, Kristopher Rush, Cal Waits, Kurt Wallnau, & Lutz Wrage

December 2009

TECHNICAL REPORT
CMU/SEI-2009-TR-025
ESC-TR-2009-025

http://www.sei.cmu.edu

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

Table of Contents

List of Figures

List of Tables

Abstract

The Software Engineering Institute (SEI) annually undertakes several independent research and development (IRAD) projects. These projects serve to (1) support feasibility studies investigating whether further work by the SEI would be of potential benefit and (2) support further exploratory work to determine whether there is sufficient value in eventually funding the feasibility study work as an SEI initiative. Projects are chosen based on their potential to mature and/or transition software engineering practices, develop information that will help in deciding whether further work is worth funding, and set new directions for SEI work. This report describes the IRAD projects that were conducted during fiscal year 2009 (October 2008 through September 2009).

1 Introduction

1.1 Purpose of the SEI Independent Research and Development Program

Software Engineering Institute (SEI) independent research and development (IRAD) funds are used in two ways: (1) to support feasibility studies investigating whether further work by the SEI would be of potential benefit and (2) to support further exploratory work to determine whether there is sufficient value in eventually funding the feasibility study work as an SEI initiative. It is anticipated that each year there will be three or four feasibility studies and that one or two of these studies will be further funded to lay the foundation for the work possibly becoming an initiative.

Feasibility studies are evaluated against the following criteria:

- Mission criticality: To what extent is there a potentially dramatic increase in maturing and/or transitioning software engineering practices if work on the proposed topic yields positive results? What will the impact be on the Department of Defense (DoD)?
- Sufficiency of study results: To what extent will information developed by the study help in deciding whether further work is worth funding?
- New directions: To what extent does the work set new directions as contrasted with building on current work? Ideally, the SEI seeks a mix of studies that build on current work and studies that set new directions.

1.2 Overview of IRAD Projects

The following research projects were undertaken in FY 2009:

- Modeling and Validating MILS Security Architecture
  Jorgen Hansson, Lutz Wrage, and Peter Feiler
- Relating Business Goals to Architecturally Significant Requirements for Software Systems
  Paul Clements and Len Bass
- Achieving Predictable Performance in Multicore Embedded Real-Time Systems
  Dionisio de Niz, Jeffrey Hansen, Gabriel Moreno, Daniel Plakosh, Jorgen Hanson, Mark Klein, Karthik Lakshmanan, and Raj Rajkumar
- Parallel Distributed Acquisition System
  Kristopher Rush, Matthew Geiger, and Cal Waits
- Programming Models for the Multicore Era
  James Ivers and Kurt Wallnau

These projects are summarized in this technical report.

2 Modeling and Validating MILS Security Architecture

Jorgen Hansson, Lutz Wrage, and Peter Feiler

2.1 Purpose

The Department of Defense (DoD) policy of multi-level security (MLS) has long employed the Bell-LaPadula and Biba approaches for confidentiality and integrity; more recently, the multiple independent levels of security/safety (MILS) approach has been proposed. These approaches allow designers of software-intensive systems to specify security levels and requirements for access to protected data, but they do not enable them to predict runtime behavior. In this article, model-based engineering (MBE) and architectural modeling are shown to be a platform for multi-dimensional, multi-fidelity analysis that is conducive for use with Bell-LaPadula, Biba, and MILS approaches and enables a system designer to exercise various architectural design options for confidentiality and data integrity prior to system realization. In that way, MBE and architectural modeling can efficiently be used to validate security of system architectures and thus gain confidence in the system design.

Initial studies have shown the feasibility of using model-based engineering as a proactive measure to determine the viability of a system architecture to enforce security. In this project we propose a model-based engineering (MBE) framework containing techniques for validating MILS architectures. In epitome, this means validating structural rigidity of MILS decomposition and enforcement of assumptions and constraints required by MILS.

2.2 Background

Security and confidentiality[1] are becoming increasingly important in embedded and real-time systems, which are characterized to operate under significant resource constraints while ensuring high levels of dependability (e.g., reliability, availability, safety) as well as security. Encryption, authentication, security and protection mechanisms add increased bandwidth (CPU, network, memory), affect temporal behavior of the system as well as power consumption (especially important in battery-driven, or limited-life time devices (e.g., sensor networks or cellular phones). Model-based engineering has shown promise in validating non-functional behavior of a system architecture, and by annotating a model with multiple dimensions of attributes regarding performance, power usage, security, etc., it becomes possible to conduct in-depth validation of system behavior, and also reason about the effects of architectural changes, or changes in, for example, security requirements.

Security as an architectural concern crosscuts all levels of the system (application, middleware, operating systems, and hardware). Security requires intra- and inter-level verification and has immediate effects on the runtime behavior of the system, specifically on other dependability attributes.

[1] Confidentiality addresses concerns that sensitive data should be disclosed to or accessed only by authorized users (i.e., enforcing prevention of unauthorized disclosure of information). Data integrity is closely related, as it concerns prevention of unauthorized modifications of data.

The designer must enforce inter-level and intra-level security through the architecture. Figure 2-1 depicts various system levels involved in the verification of security privileges against confidentiality requirements. The designer seeks to ensure that the software applications do not compromise the confidentiality of the secure information they are exchanging. Consequentially, software applications must execute on top of a secure operating system, be mapped to a protected and secured hardware memory space, and communicate over a secure communication channel. If the data is labeled confidential, then, every architectural layer must have a clearance of at least that level.

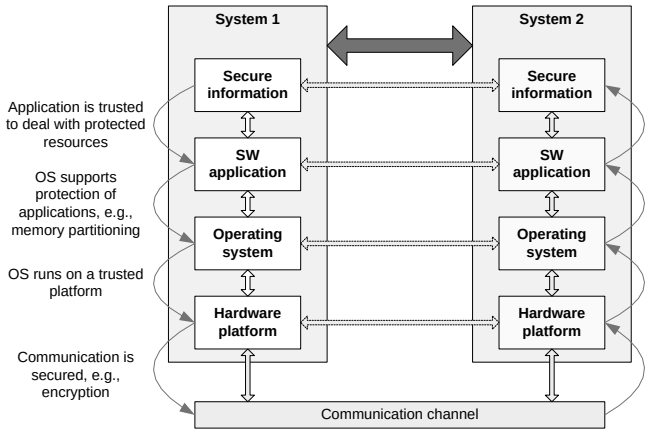


Figure 2-1: System Perspective on Security

As mentioned earlier, the designer needs to acknowledge that security comes with a cost. Encryption, authentication, security, and protection mechanisms increase bandwidth demand in terms of CPU, network, and memory. These increases affect temporal behavior of the system (worst-case execution time, response time, schedulability, and end-to-end latency) as well as power consumption (especially important in battery-driven or limited-life time devices). Security is interlinked with the other non-functional behaviors, such as predictability/timeliness and resource consumption, and it inadvertently affects reliability and availability. As a result, security cannot be considered in isolation. The system designer makes choices to trade these quality attributes against each other. Figure 2-2 illustrates some of those dependencies on the single model–multiple analyses view.

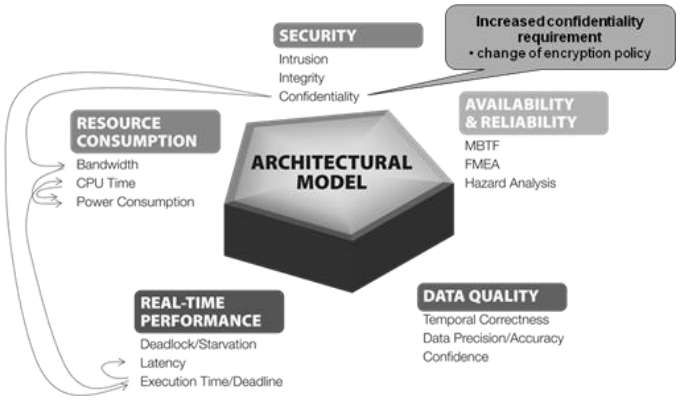


Figure 2-2: An Example of Impact on and Interaction of Non-Functional Behavior Due to a Change in Security

MILS has been proposed as an approach to building secure systems, and it provides a reusable framework for high assurance system specification and verification. It is led by the United States Air Force Research Laboratory, and represents a joint effort across the Air Force, Army, Navy, National Security Agency, Boeing, Lockheed Martin, and others. The core idea of MILS is to reduce the amount of safety/security critical code; reducing the size allows scrutiny and analysis to be conducted that otherwise would be deemed intractable. To achieve this, MILS adopts the concept of a separation kernel. The identified benefits of using a MILS architecture are

- reduction in physical hardware
- easier control and management of information among various communities of interest
- cheaper development of highly secure systems, as well as a faster time-to-market
- overall increase in safety
- less need for re-architecting systems to meet security standards

2.3 Approach

Modeling of system quality attributes, including security, is often done—when it is done—with low-fidelity software models and disjointed architectural specifications by various engineers using their own specialized notations. These models are typically not maintained or documented throughout the life cycle and make it difficult to obtain a system view. However, a single-source architecture model of the system that is annotated with analysis-specific information allows changes to the architecture to be reflected in the various analysis models with little effort; those models can easily be regenerated from the architecture model. This approach also allows the designer to conduct adequate tradeoff analysis and evaluate architectural variations prior to system realization, gaining confidence in the architectural design. Models also can be used to evaluate effects of reconfiguration and system revisions in post-development phases.

To model and validate the confidentiality of a system, we distinguish between general and application-dependent validation. General validation of confidentiality is the process of ensuring that a modeled system conforms to a set of common conditions that support system confidentiality independent of a specific reasoning framework for security. MBE takes advantage of the versatile concept of subjects operating on objects by permissible access (read, execute, append, and write), a notion introduced by Bell and LaPadula, enabling us to model and validate security at both the software and hardware levels. This form of validation assumes that subjects and objects are assigned a security level that is the minimum representation to enforce basic confidentiality and need-to-know principles. By contrast, application-specific validation relies on detailed confidentiality requirements and a specific reasoning-based security framework.

2.3.1 Validating MILS Architectures with the MBE Approach

MILS uses two mechanisms to modularize—or to "divide and conquer"—in architecting secure systems: partitions and separation into layers. The MILS architecture isolates processes in partitions that define a collection of data objects, code, and system resources and can be evaluated separately. Each partition is divided into three layers, each of which is responsible for its own security domain and nothing else: separation kernel, responsible for enforcing data isolation, control of information flow, periods processing, and damage limitation; middleware service layer; and application layer.

Thus, MILS separates security mechanisms and concerns into the following components types, classified by how they process data:

- SLS—single-level secure component: processes data at one security level
- MSLS—multiple single-level secure component: processes data at multiple levels, but maintains separations between classes of data
- MLS—multi-level secure component: processes data at multiple levels simultaneously

The MILS architecture approach builds on partitioning as one key concept to enforce damage limitation.

At an abstract level, a MILS architecture is a collection of data processing components (processes) that can communicate with each other. These processes are isolated, such that they can communication over known communication paths only—that is, there are no covert channels between any two processes. Isolation of components can be achieved by placing each process on a different processor and connecting them with one or more networks As shown in Figure 2-3, each process is labeled with its MILS classification (SLS, MSLS, MLS) and the security levels of the data it is permitted to process.

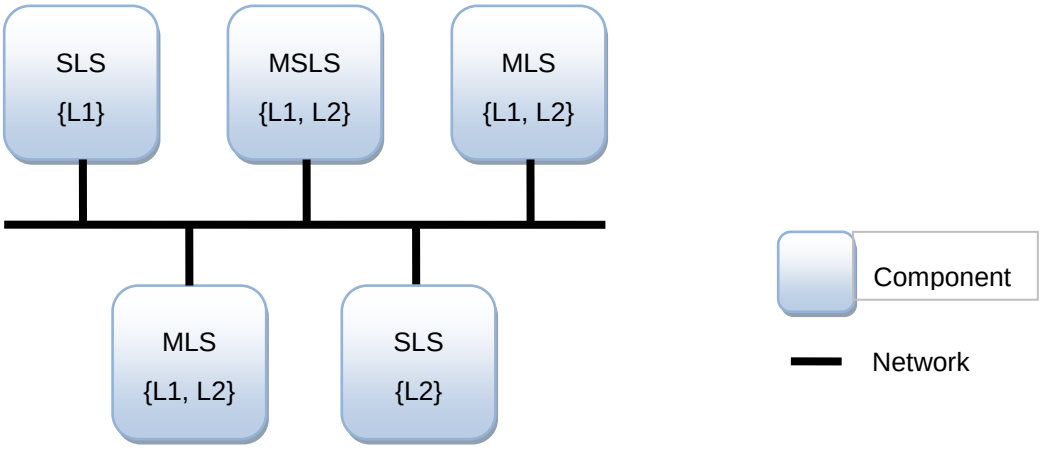


Figure 2-3: Conceptual View of a MILS System

Using one processor per MILS process is expensive and adds weight to the system. Also, the network connection adds latency to communication and may limit the available bandwidth. For these reasons it is preferable to be able to place multiple processes on the same processor. This can be achieved by a trusted separation kernel (SK) that partitions a single physical computer into a number of virtual machines or *partitions* that can each provide an execution environment for a process. The concept of partitions is also used in avionics systems, where the main motivation is fault containment. All communication between partitions on the same processor is controlled by the SK. Many real-world systems contain multiple processing nodes that are connected via one or more networks. Each processing node can run an SK to manage local partitions, whereas trusted middleware is needed to control information flow between partitions on different nodes.

A system is secure if all information flows through and between processing components adhere to the system's security policy. Information flows inside a process depend on the implementation

and must be evaluated for each individual process. MILS focuses on enforcing the security policy for information flows between processes. The SK and middleware are trusted to check if data exchanged between processes adheres to the policy. In particular, for the Bell-LaPadula security, the MILS approach guarantees that a system is secure if it is composed of secure components.

There are two possible approaches to enforcing the security policy for inter-partition policy enforcement: dynamic and static. Dynamic policy enforcement is needed if a system has a variable number of processes where a process can communicate with any other process, or the security level of processes can change at runtime. A static environment has a fixed number of processes communicating using fixed channels, and the security level of a process remains constant. If a process exchanges information at different security levels, this must happen through communication ports that have different but fixed security levels.

In the dynamic case, messages exchanged between components must be labeled with the security level of the contained data. This security level can be the (current) security level of the sender processes or the security level of a communication port of the process. The trusted intermediary (SK and/or middleware) verifies at runtime—based on the message security level and receiver's (current) security level—that the security policy allows the delivery of the message to the receiving process.

In the static case, all communication channels and the security level of data exchanged over them are known a priori. The conformance of communication paths to the security policy can be verified statically at design time. It is now sufficient to configure the trusted intermediary to allow only communication along these pre-determined channels. There is no need to evaluate security levels at runtime.

The Architecture Analysis & Design Language (AADL) can be used to model a MILS system with static policy enforcement as outlined in the previous section. For example, an AADL process can model a MILS partition, and one or more AADL threads can model the execution of a MILS process inside a partition. Ports and their connections represent a static set of communication channels along which processes exchange information. We indicate the security level of the information at the AADL port using an AADL property.

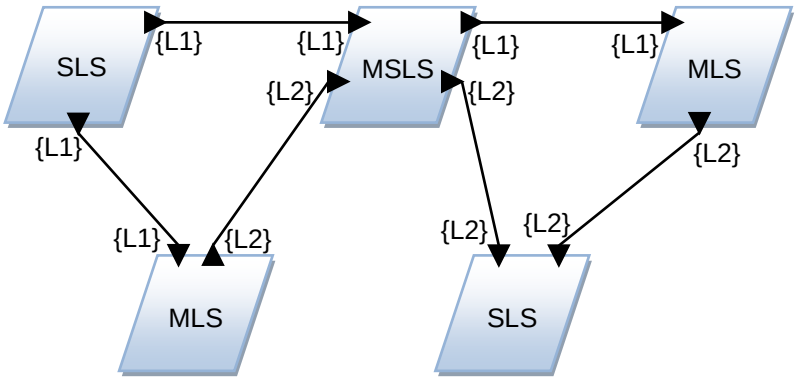


Figure 2-4: Application Layer View of a MILS System in AADL

A MILS SK can be modeled as the software part of an AADL processor. Such a processor also represents hardware components that execute threads. To indicate the presence of an SK in a processor we label the processor as trusted. In AADL we model the fact that a partition executes on a

processor by binding the corresponding AADL process to the AADL processor. The network that connects a set of processors is modeled as one or more AADL bus components. Connections between AADL processes that reside on different AADL processors are bound to these bus components to model that a communication channel is routed over a certain network. An AADL bus component represents the physical network and the protocols used in the communication. This means that trusted middleware is considered to part of the bus. If communication over a network uses trusted middleware we label the corresponding bus component accordingly. Figure 2-5 shows AADL processes bound to trusted processors and connections bound to a trusted bus. To simplify the picture, we have grouped processes that run on the same processor in an AADL system component. The processor binding is shown at the system level.

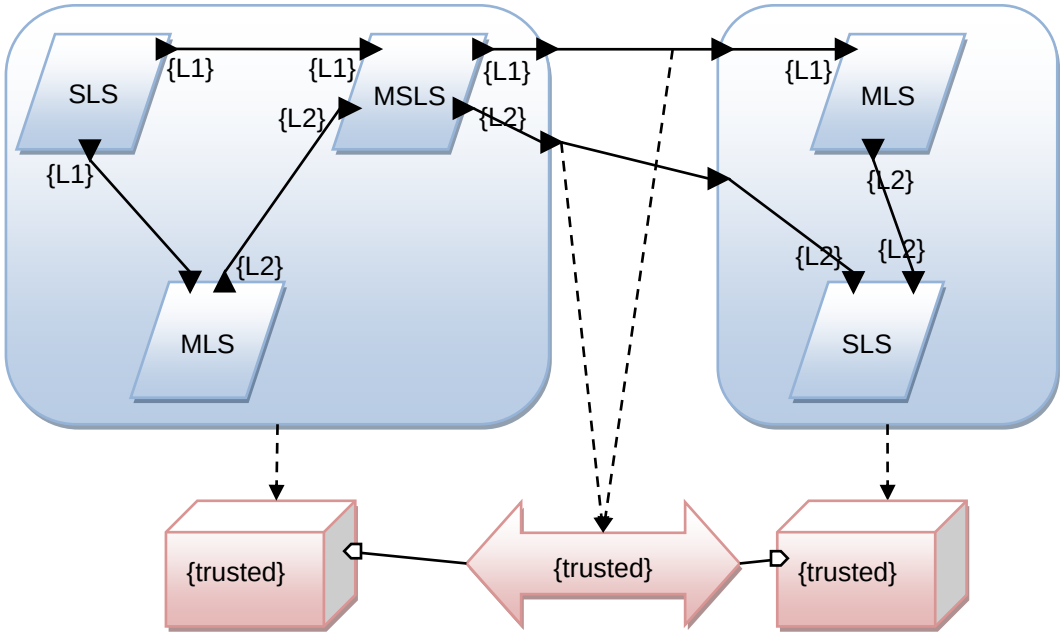


Figure 2-5: MILS Application Layer Components Bound to Execution Platform

2.3.2 MILS Classification in AADL

The MILS classification of a subject is a label that expresses if a subject may handle objects of only a single security level or objects of multiple security levels. In addition, if a subject can handle multiple security levels, the label must capture if a subject that handles objects of multiple security levels keeps these security levels separate from each other. We use the labels SLS, MSLS, and MLS. In AADL, we define a property MILS::Classification in a new property set named MILS. The type of this property, MILS::Classes, is defined as an enumeration containing the three MILS category labels.

Table 2-1: The MILS Property Set

```
property set MILS is
 Classes: type enumeration (SLS, MSLS, MLS);

 Classification: MILS::Classes
 applies to (system, process, device);

 Partition: aadlboolean => false
 applies to (process);

 Trusted: aadlboolean => false
 applies to (processor, bus);
end MILS;
```

We assign no default value to the MILS::Classification property, such that the classification of an AADL component is initially undefined. Assuming that all system, process, and device components should be explicitly assigned a MILS classification, we can develop a simple checking procedure to identify components that still need a classification in a partially refined model. Based on the component's features and flows through the component, the analysis can also propose an appropriate label. The MILS classification of a component does not propagate down the containment hierarchy to subcomponents because the MILS classification does not generally constrain valid classifications of its subcomponents; it depends only on the component's features and flows.

The standard AADL semantics of process components are not quite strong enough to model MILS partitions. AADL processes have their own protected address spaces (space partitioning), but they may interfere with each other's timing (i.e., no time partitioning). This opens up the possibility of covert channels between two processes based on timing variations. We add this requirement to processes through the property MILS::Partition.

The third property, MILS::Trusted, can be applied to execution platform components. It indicates if a component guarantees the enforcement of constraints on the allowed communication between application software components that are bound to it.

2.3.3 MILS Classification of AADL Components

A MILS classification can be assigned to AADL process, device, and system components. External enforcement of security policy is possible for interactions between components of these categories.

An AADL process component represents a protected address space that contains executable code and data. The code inside a process is executed by one or more threads, where each thread may potentially access all memory locations that belong to its containing process and execute all code loaded into that process. The execution platform cannot enforce any constraints on these intra-process interactions. In contrast, interactions between threads that execute in different processes can be controlled by the underlying execution platform. Any communication that leaves the process's address space can only be performed indirectly by invoking services of the execution platform.

An AADL device component typically represents a physical device together with associated device driver software. The device driver runs in kernel space and is considered part of an AADL

processor component. User-space device drivers are modeled as explicit threads. Any communication between a device and a process can be controlled by the execution platform because the communication must either use platform services for communication or use shared data under the control of the platform's MMU.

AADL system components can represent a complete system or a subsystem composed of execution platform and application software components. We include system components as MILS components to support hierarchical and incremental modeling.

System, process, and device components can be treated as isolated entities, where the isolation is enforced by the AADL semantics. Other application software components are not isolated, that is, there is no a priori guarantee that other application software components do not interact in a way that violates MILS constraints. As an example, assume that an AADL model contains a process that has two thread subcomponents. If the model does not indicate any interactions between these two threads, if must be verified, by inspecting the actual thread implementations as used in the deployed system, that there are indeed no interactions between them. In contrast, the execution platform can prevent threads from exchanging data across process boundaries if such an interaction is not specified in the model.

Execution platform components are not assigned a MILS classification because all application-specific manipulation of data happens in application software components. Application software components are bound to execution platform components, such that we need to capture if an execution platform component can support only application components that have a single security level or if application components with multiple security levels can be bound to it. The property MILS::Trusted indicates if an AADL processor component provides verified enforcement mechanisms to limit communication between processes and devices to only those that are explicitly allowed, that is, are declared as connections in the AADL model. Similarly, a bus is trusted if it only allows information flow along communication paths that are specified in the model.

Memory components are passive and cannot cause unintended information flow, such that they are implicitly trusted.

2.3.4 Information Flows in AADL Models

In AADL we model explicit information flow into and out of application software components with features (ports, parameters, and shared data access). These features are connected to model the path along which information flows either between features of components on the same level of the containment hierarchy or between subcomponent features and features of their enclosing component. The directions of features and connections determine the direction of information flow. Explicit information flows through a component can be represented in an AADL component type by a flow specification that starts at an incoming feature and ends at an outgoing feature. Such a flow specification is an abstraction of the path along which information flows from the incoming feature through contained connections and subcomponents to the outgoing feature.

In addition to explicitly connected features, remote subprogram calls implicitly introduce information flows via subprogram parameters (and outgoing subprogram ports) whose values are transferred between the calling and the called thread. The direction of subprogram parameters and ports determines the direction of information flows. If a remotely called subprogram has access to

shared data, its features may be the source or destination of a flow through the component containing the subprogram.

2.3.4.1 Information Flow Through Ports and Parameters

Port connections represent the main sources of information flows between components and through each individual component. Connections between processes features are directional such that information flow along a connection can be unidirectional or bidirectional.

An AADL port group represents an aggregation of ports and nested port groups. A security level is associated with each individual port that is part of a port group; it is assigned in a port group type declaration. In general, information flows of different security levels may pass through the same port group. To evaluate MILS characteristics of a port group, it would then be necessary to inspect the security level of each port inside a port group and flows between individual ports. In the following, we limit the discussion to models without port groups.

Parameter connections are very similar to port connections. The only difference is that at least one end of a parameter connection must be a subprogram parameter.

2.3.4.2 Shared Data Access

If processes that share data are deployed on the same execution platform, shared data can be implemented by mapping the same portion of physical address space into these processes' virtual address spaces. This way all processes that share the data have direct access to the same memory area containing shared data. Even though the memory mapping may constrain the access to read-only or write-only access, this is not sufficient because it is not based on the security level of the data. From a MILS perspective, access to this shared data area happens outside the control of the execution platform such that it cannot enforce constraints on information flow via the shared data. In practice this means that all participating processes must be SLS processes of the same security level because the shared data area breaks the encapsulation that is otherwise guaranteed by the process. In an AADL model, a situation where shared data is mapped into the virtual address space of multiple processes, the shared data component is placed inside a system component that also contains processes that access the data.

Another implementation option is to place the data in the virtual address space of one participating process and have all access by other processes under the control of the execution platform. The process that owns the shared data has full access to the shared data and can read or write data of any security level. A certain security level for the shared data can only be enforced if it is owned by an SLS process. In AADL, we model this by making a shared data component a subcomponent of a process. This process makes the data available to others via access features.

The execution platform has full control over all accesses to shared data when the data is placed inside a separate address space. In AADL this is modeled as a data subcomponent inside a process that contains only data subcomponents and no threads. Such a process represents a passive data store, and if all data subcomponents have the same security level, the process is an SLS component. The individual data subcomponents can also have different security levels such that the process is an MSLS component.

2.3.5 Execution Platform Bindings

Processes are deployed on an execution platform. The deployment assigns a processor for executing the process's threads, and memory to store its code and data. The deployment includes the configuration of communication channels between processes on the same processor and assignment of network communications to connections between processes deployed on different processors.

2.3.5.1 Binding Processes to Processors

A trusted processor includes an SK. Such a processor supports execution of partitions as isolated components (space and time partitioning). In an AADL model this means that multiple processes with property MILS::Partition=>true can be bound to an AADL processor with property MILS::Trusted=>true. The trusted processor also supports enforcement of constraints on the information flows between partitions. It must be configured to allow only those communication paths between partitions that are present in the AADL model. The configuration can be derived directly from the AADL model. Allowed communication paths include

1. semantic connections between thread subcomponents of different partitions
2. bindings of subprogram calls and server subprograms (as recorded in Actual_Subprogram_Call property associations), where the call crosses a partition boundary

An SK needs certain hardware support to provide space and time partitioning.

For an untrusted processor one must assume that arbitrary communication between processes can occur. Such a processor does not provide time partitioning and may not be able to enforce constraints on communication paths. As a result, only SLS processes of the same security level may be bound to an untrusted processor. Note that, from a MILS perspective, it is irrelevant if an untrusted processor provides space partitioning, such that an untrusted processor can have any combination of hardware and operating system.

2.3.5.2 Binding Devices to Processors

An AADL device component represents a physical device and kernel mode driver components. The binding to a processor determines which process executes the kernel-mode driver. Secure bindings are the same as for processes: If the processor is untrusted, devices bound to it must be SLS devices of the same security level as the processes on the processor. MSLS and MLS devices must be bound to a trusted processor.

2.3.5.3 Binding Processes to Memory

Processes are bound to memory components that provide the address space for the process. AADL allows binding of processes to overlapping memory address ranges. An overlapping binding is only allowed if the involved processes are SLS components of the same security level. Overlapping memory binding can be used to share data among SLS processes. MSLS and MLS processes must always be bound to disjoint memory address ranges.

2.3.5.4 Binding Connections to Buses

Connections between processes that are bound to different processors must be bound to AADL bus components. These buses represent a physical communication medium together with the communication protocols. In MILS terms, a bus represents the trusted middleware layer. A trusted bus includes a protocol that can be configured to allow only certain explicitly specified communication paths. The configuration can be derived from the connections and actual subprogram call properties in an AADL model. A trusted bus can carry information at different security levels.

An untrusted bus can only carry data at a single security level because it does not guarantee delivery to the intended recipient only. For example, any node can listen to all packages traveling on an Ethernet bus. This implies that encryption must be added to make such an Ethernet trusted.

2.4 Collaborations

This project was done in collaboration with Professor Sang H. Son, Professor John A. Stankovic, and Vibha Prasad (all with the University of Virginia, Charlottesville), and Julien Delange (Ecole Nationale Supérieure des Télécommunications).

Valuable comments on the approach and work were also provided by Mark Vanfleet and Matthew Benke (both with the National Security Agency).

2.5 Evaluation Criteria

The key criteria for evaluating this project have been the ability to model and validate security and confidentiality using a MILS approach.

2.6 Results

Architectural modeling provides a platform for multi-dimensional, multi-fidelity analysis that is conducive for use with Bell-LaPadula, Biba, and MILS approaches and enables a system designer to exercise various architectural design options for confidentiality and data integrity prior to system realization. In that way, architectural modeling can efficiently be used to verify security of system architectures and thus gain confidence in the system design. Using AADL and OSATE, the SEI has developed analytical techniques to represent standard security protocols for enforcing confidentiality and integrity, such as Bell-LaPadula, Chinese wall, role-based access control, and Biba model; and model and verify security using system architecture according to flow-based approaches early and often in the life cycle. In this project we have demonstrated that an MBE approach is conducive to the validation concerns most critical to MILS, including:

- validating the structural rigidity of architecture, such as the enforcement of legal architectural refinement patterns with security components. This decomposition can be applied to components, connectors, and ports. Confidence in validation of an architecture increases with the fidelity of the modeling; MBE analysis can be applied at different architectural refinement levels.
- architectural modeling and validation of assumptions underlying MILS, such as assumptions with respect to damage limitation and partitioning, and validation of separation in time (and space). The AADL supports the modeling of partitions and virtual processors. The virtual

machine mechanism is recognized as a key concept for providing robustness through fault containment, because it provides time and space partitioning to isolate application components and subsystems from affecting each other due to sharing of resources. This architecture pattern can be found in the ARINC 653 standard.

- architectural modeling validation that software applications executing on top of a secure operating system map to a protected and secured hardware memory space and communicate over secure communication channels. It also enables the analysis of security measures early and throughout the development life cycle.

The validation through architectural modeling of system security given confidentiality requirements of data objects and security clearance by users must include validation of (i) software architecture and (ii) system architecture where the software architecture is mapped to hardware components. Through an MBE approach, by mapping the entities of a software architecture (e.g., processes, threads, and partitions) to a hardware architecture (consisting of, for example, CPUs, communication channels, and memory), we can ensure that the hardware architecture supports required security levels.

2.7 Publications and Presentations

Presentations have been given in a number of forums including the SAE AADL Standards User Group meeting, the Open Group RT-Forum Workshop, Institute for Defense and Government Advancement (IDGA), Aerospace Vehicle Systems Institute (AVSI), Lockheed Martin, and University of Illinois, Urbana-Champaign.

The following publications are related to this project. All conference papers were accompanied by a presentation.

[Hansson 2008a]
J. Hansson, P. H. Feiler, & A. Greenhouse. "Enforcement of Quality Attributes for Net-centric Systems through Modeling and Validation with Architecture Description Languages." Fourth Congress on Embedded Real-Time Systems (ERTS), January 2008.

[Hansson 2008b]
J. Hansson, P. H. Feiler, & J. Morley. "Building Secure Systems Using Model-Based Engineering and Architectural Models." *Crosstalk 21*, 9 (September 2008).

[Hansson 2009]
J. Hansson, P. H. Feiler, J. Hugues, B. Lewis, & J. Morley. "Model-Based Validation of Security and Non-Functional Behavior Using AADL." submitted to *IEEE Security & Privacy*, Special Issue on Complex Architectures (2009).

3 Relating Business Goals to Architecturally Significant Requirements for Software Systems

Paul Clements and Len Bass

3.1 Purpose

The purpose of this project is to facilitate better elicitation and capture of high-pedigree quality attribute requirements. Towards this end, we want to be able to reliably elicit business goals and understand how those business goals influence quality attribute requirements and architectures.

The elicitation approaches produced by this project can be used by

- requirements engineers who want to produce a set of requirements helpful to the software architect
- outside parties such as those running a Quality Attribute Workshop
- the architect in case nobody else has done it

3.2 Background

A quality attribute of a system is a characteristic that helps to determine whether the system is fit to use. A runtime quality attribute is one that can be observed and measured as the system executes; these include security, performance, availability, and usability. Development time quality attributes are those that can be measured by observing the development or maintenance of the system and include many that are not included in standard lists of quality attributes [Donaldson 1995] such as modifiability, reusability, shortening time to market, conformance with legal and regulatory requirements, etc. All of these, however, affect the fitness for use of a system to its stakeholders (the definition of quality in ISO standard 9126) and have, potentially, strong and definable influence on the architecture of the system being constructed.

Properties of a system that help determine whether it satisfies the fitness criteria derive from business goals. If we ask, for example, "*Why* do you want this system to have a really fast response time?" we might hear that this will differentiate the product from its competition and let the developing organization capture market share, or that this will make the end user more effective and this is the mission of the acquiring organization, or other reasons having to do with the satisfaction of some business goal.

Not all business goals for an organization are achieved through the construction of a system. For example, "reduce cost" may come about by lowering the facility's thermostats in the winter or reducing employees' pensions. Other business goals may directly affect the system without precipitating a standard quality attribute requirement per se. For example, we know of a case where a manager pressed for an architecture to include a database because the organization's database group was currently sitting idle. No requirements specification would capture such a "requirement." And yet that architecture, if delivered without a database, would be just as deficient from the point of view of the manager as if it had failed to deliver an important user function.

Figure 3-1 illustrates the salient points. In the figure, the arrows mean "leads to"; the solid arrows highlight the relationships of most interest to architects.

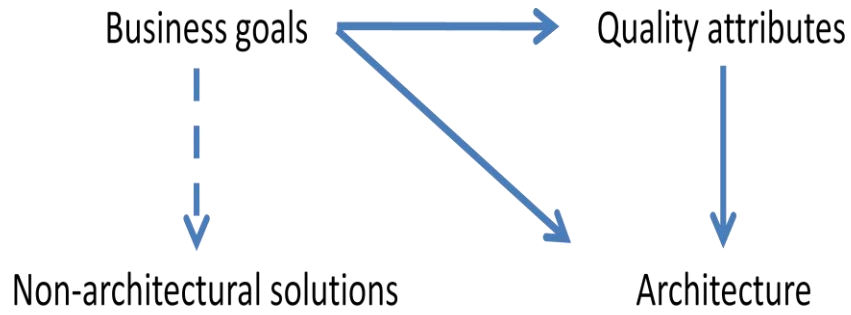


Figure 3-1: Some business goals lead to quality attribute requirements (which lead to architectures; others lead directly to architectural decisions; still others lead to non-architectural solutions).

3.3 Approach

Our approach was four-fold:

1. We conducted a thorough search of the business literature to collect examples of common business goals. We had hoped to discover and use an existing taxonomy of business goals based on decades, if not centuries, of businesses going about achieving specific goals. Such a taxonomy was not forthcoming. However, the business literature did provide a plethora of papers on business goals or business models for organizations. Our literature search used the Proquest ABI/Inform Global database, which covers over 3,000 business-oriented publications, as well as a simple Google search. In both cases, various combinations of "business goals," "business models," and "survey" or "studies" were used as search terms.
2. We used the results of the survey to produce a canonical set of ten business goal categories. Our belief is that any specific business goal will fall into at least one of our categories.
3. We crafted a seven-part syntactic structure to unambiguously express a business goal.
4. We designed (and piloted) a method, which we call the Pedigreed Attribute eLicitation Method (PALM), for articulating and capturing the business goals that underlie the development and/or acquisition of a software system. Those business goals can then be used as the basis for an investigation into relevant quality attribute requirements for the system.

3.4 Collaborations

The SEI team consisted of Paul Clements and Len Bass, assisted in the early stages by John Bergey. Our collaborators (providing their own support) included participants in two workshops held in Pittsburgh and Amsterdam, respectively, plus the pilot application of our business goal elicitation method.

The workshops were intended for us to try out the ideas behind our work, and to receive feedback from architects and acquirers as to the role of business goals in architecture. At the Pittsburgh workshop, we were joined by attendees from the U.S. Army, Boeing, Raytheon, Lockheed Martin, the U.S. Naval Research Laboratory, and VistaPrint, Inc. At the Amsterdam workshop, we

were joined by participants from Logica, Vreije Universiteit, Atos Consulting, and the Dutch Ministry of Home Affairs.

Finally, we conducted a full pilot exercise using the method derived from our work at the Air Traffic Management business unit of Boeing.

3.5 Evaluation Criteria

Peer review accounts for our primary evaluation criterion. Attendees at both workshops were enthusiastic about a method to help put clearly articulated business goals on the table, where they could be examined and used as the explicit pedigree for quality attribute requirements. We expect to publish reports in relevant forums and the acceptance of these publications will constitute another measure of project success. Finally, the incorporation of our method into an SEI offering (either as a standalone method or as part of an existing architecture analysis activity) will constitute a measure of success.

3.6 Results

A canonical categorization of business goals

We conducted an affinity (clustering) exercise with the goals uncovered from our search of the business goal literature. The result is the set of ten business goal categories shown in Figure 3-2. Space limitations prevent us from showing the goals we captured in our literature search and how each one maps to one or more of these categories.

1. Growth and continuity of the organization	6. Meeting responsibility to country
2. Meeting financial objectives	7. Meeting responsibility to shareholders
3. Meeting personal objectives	8. Managing market position
4. Meeting responsibility to employees	9. Improving business processes
5. Meeting responsibility to society	10. Managing quality and reputation of products

Figure 3-2: Ten Categories of Business Goals

In addition to papers enumerating and describing business goals, we also discovered two other relevant studies. In the first, Osterwalder and Pigneur 0 describe how managers should deal with changing forces in the social, legal, competitive, customer, and technological realms (Figure 3-3). When we elicit a business goal, this gives us insight into asking how that business goal might change over time.

The second relevant study from our literature survey comes from a field of research in the business/management world called “stakeholder theory.” Here, organizations (e.g., corporations) are viewed as a collection of stakeholders all working out their vested interests with each other. Donaldson and Preston [Donaldson 1995] provide a stakeholder-centric model of a corporation, as shown in Figure 3-4. In this view, all persons or groups with legitimate interests participating in an enterprise do so to obtain benefits and there is no prima facie priority of one set of interests and benefits over another. Hence, the input–output arrows between the firm and its stakeholder constituents run in both directions.

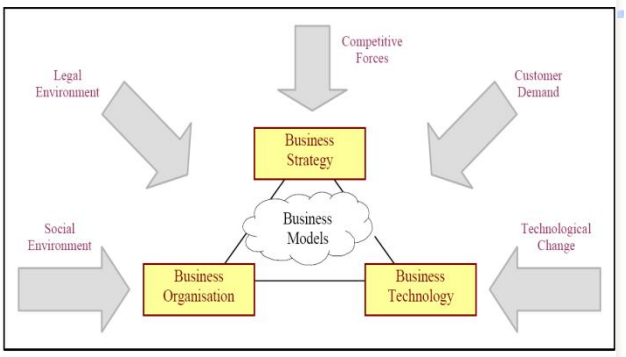


Figure 3-3: Business Models Lie at the Intersection of Strategy, Organization, and Technology (reproduced from Osterwalder [Osterwalder 2004])

A stakeholder is broadly viewed as "any group or individual who can affect or is affected by the achievement of the organization's objectives" [Freeman 1984] although narrower definitions exist that try to focus on stakeholders of the greatest importance.

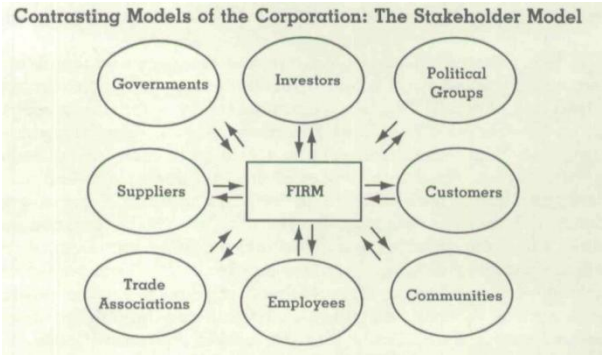


Figure 3-4: Stakeholder Model of a Corporation (reproduced from Donaldson [Donaldson 1995])

Stakeholder theory gives us another way to view business goals by categorizing them according to the stakeholders they are intended to address and by helping us identify sources of business goals.

A Syntax for Expressing Business Goals

Capturing business goals and then expressing them in a standard form will let them be discussed, analyzed, argued over, rejected, improved, reviewed—in short, all of the same activities that result from capturing any kind of requirement. To this end, we introduce *business goal scenarios*. The purpose of a business goal scenario is to ensure that all business goals are expressed clearly, in a consistent fashion, and contain sufficient information to enable their processing through the further steps of our technique.

Our business goal scenario has seven parts. They all relate to the system under development, the identity of which is implicit. Together, they provide a provenance for a business goal that will contribute to its understanding and its interplay with other goals. The seven parts are

1. **Goal-source.** Who (or what artifact) provided the statement of the goal?
2. **Goal-subject**. To express a business goal meaningfully, as well as capture information to resolve goal conflicts, we need to know the person who owns the goal. This we call the *goal-*

subject. If the business goal is, for example, "maximize dividends for the stakeholders" (a goal described in Fulmer [Fulmer 1978]), who is it that cares about that? It is probably not the programmers, the system's end users, or (if an acquired system) anyone in the acquisition organization. Stakeholder theory [Donaldson 1995] will help us identify the goal-subject(s) of a business goal, as well as help us identify people who might have business goals to contribute. We will seek stakeholders with high "salience" from whom to elicit business goals, and record those stakeholders as the goal-subjects.

3. **Goal-object**. A goal's *object* (in the sense of a verb's object in a sentence) lets us ask, "What do you wish to be true for or about **X** as a result of developing or acquiring this system?" The placeholder **X** is the goal's object. All goals have goal-objects—we want something to be true about something (or someone) that (or whom) we care about. Seen in this light, the goals captured in our literature search can all be re-elaborated by making their respective goal-objects explicit. By doing so, we were able to discern the goal-objects listed in Table 3-1.

Table 3-1: Business Goals' Goal-Objects

Goal-Object	Remarks
Individual	The individual who has these goals has them for him/herself or his/her family.
System	These can be goals for a system being developed or acquired.
Portfolio	These goals apply either to a single system, or to an organization's entire portfolio that the organization is building or acquiring to achieve organization-wide goals.
Organization's employees	Before we get to the organization as a whole, there are some goals aimed at specific subsets of the organization.
Organization's shareholders	
Organization	These are goals for the organization as a whole. The organization can be a development or acquisition organization.
Nation	Before we get to society at large, this goal-object is specifically limited to the goal-owner's own country.
Society	Some interpret "society" as "my society," which puts this category closer to the Nation goal-object, but we are taking a broader view.

4. **Environment.** This is the context for this goal. It acts as a rationale for the goal. One source for this entry is the five different environmental factors of Osterwalder and Pigneur [Osterwalder 2004] (Social, Legal, Competitive, Customer, and Technological).
5. **Goal.** This is any business goal (whether mentioned in this paper or not) able to be articulated by the person being interviewed.
6. **Goal-measure.** This is a measurement to determine how one would know if the goal has been achieved.
7. **Value.** This is how much the goal is worth. It might be expressed in terms of what might be willingly paid to achieve it, or a ranking against other goals in a collection.

A method for eliciting business goals and comparing them to quality attribute requirements

The Pedigreed Attribute eLicitation Method (PALM) is a seven-step method. The steps are:

1. **PALM overview presentation**: Overview of PALM, the problem it solves, its steps, its expected outcomes.
2. **Business drivers presentation**: Briefing of business drivers by project management. What are the goals of the customer organization for this system? What are the goals of the development organization? This is a lengthy discussion that gives the opportunity to ask questions about the business goals as presented by project management.
3. **Architecture drivers presentation**. Briefing by the architect on the driving (shaping) business and qual**i**ty attribute requirements.
4. **Business goals elicitation.** Using the standard business goal categories to guide discussion, we capture the set of important business goals for this system. Business goals are elaborated, and expressed as business goal scenarios. We consolidate almost-alike business goals to eliminate duplication. Participants then prioritize the resulting set to identify the most important ones.
5. **Identifying potential quality attributes from business goals**. For each important business goal scenario, participants describe a quality attribute that (if architected into the system) would help achieve it. If the QA is not already a requirement, this is recorded as a finding.
6. **Assignment of pedigree to existing quality attribute drivers.** For each architectural driver named in Step 3, we identify which business goal(s) it is there to support. If none, that is recorded as a finding. Otherwise, we establish its pedigree by asking for the source of the quantitative part: e.g.: Why is there a 40ms performance requirement? Why isn't 60ms adequate? Or 80ms?
7. **Exercise conclusion.** Results, next steps, and participant feedback are reviewed.

Using PALM

We see PALM as helping architects in the following ways:

1. PALM can be used to sniff out missing requirements early in the life cycle. For example, having stakeholders subscribe to the business goal of improving the quality and reputation of their products may very well lead to (for example) security, availability, and performance requirements that otherwise might not have been considered.
2. PALM can be used to inform the architect of business goals that directly affect the architecture without precipitating new requirements. This is the diagonal arrow in Figure 3-1. For example, if an organization has the ambition to use the product as the first offering in a new product line, this might not affect any of the requirements for that product (and therefore not merit a mention in the project's requirements specification). But this is a crucial piece of information that the architect needs to know early so it can be accommodated in the design.
3. PALM can be used to discover and carry along additional information about existing requirements. For example, a business goal might be to produce a product that out-competes a rival's market entry. This might precipitate a performance requirement for, say, half-second turnaround when the rival features one-second turnaround. But if the competitor releases a new product with half-second turnaround, then what does our requirement become? A conventional requirements document will continue to carry the half-second requirement, but the goal-savvy architect will know that the real requirement is to beat the competitor, which may mean even faster performance is needed.

4. PALM can be used to examine particularly difficult quality attribute requirements to see if they can be relaxed. We know of more than one system where a quality attribute requirement proved quite expensive to provide, and only after great effort, money, and time were expended trying to meet it was it revealed that the requirement had no analytic basis, but was merely someone's best guess or fond wish at the time.
5. Different stakeholders have different business goals for any individual system being constructed. The acquirer may want to use the system to support their mission; the developer may want to use the system to launch a new product line. PALM provides a forum for these competing goals to be aired and resolved.

PALM can be useful to developing organizations as well as acquiring organizations. Acquirers can use PALM to sort out their own goals for acquiring a system, which will help them to write a more complete request for proposals (RFP). Developing organizations can use PALM to make sure their goals are aligned with the goals of their customers. We do not see PALM as anointing architects to be the arbiter of requirements, unilaterally introducing new ones and discarding vexing ones. Rather, the purpose of PALM is to empower the architect to gather necessary information in a systematic fashion.

3.7 Publications and Presentations

[Bass 2010]
Len Bass & Paul Clements. "Business Goals as a Basis for Architecturally Significant Requirements," tutorial, 2010 International Conference on Requirements Engineering (in progress).

[Clements 2009]
Paul Clements & Len Bass. *Relating Business Goals to Architecturally Significant Requirements for Software Systems* (CMU/SEI-2009-TN-026). Software Engineering Institute, Carnegie Mellon University, 2009. http://www.sei.cmu.edu/library/abstracts/reports/09tn026.cfm

[Clements 2010]
Paul Clements & Len Bass. "Eliciting and Capturing Business Goals to Help Architects Design Systems: Experience with the Pedigreed Attribute eLicitation Method," 2010 International Conference on Software Engineering (submitted).

3.8 Bibliography

[Donaldson 1995]
Thomas Donaldson & Lee E. Preston. "The Stakeholder Theory of the Corporation: Concepts, Evidence and Implications." *The Academy of Management Review 20*, 1 (January 1995): 65.

[Freeman 1984]
R. Edward Freeman. *Strategic Management: A Stakeholder Approach*. Pitman, 1984 (ISBN-10: 0273019139).

[Fulmer 1978]
R. M. Fulmer. "Questions on CEO Succession," American Management Association, 1978.

[ISO 1998]
International Standardization Organization "Quality Characteristics and Sub-Characteristics." *Information Technology—Software Quality Characteristics and Metrics, ISO/IEC FCD 9126–1.2*, 1998.

[Mitchell 1997]
Ronald K. Mitchell, Bradley R. Agle, & Donna J. Wood. "Toward a Theory of Stakeholder Identification and Salience: Defining the Principle of Who and What Really Counts." *The Academy of Management Revie*w *22*, 4 (October 1997): 853-886. http://www.jstor.org/stable /259247

[Osterwalder 2004]
Alexander Osterwalder & Yves Pigneur, "An Ontology for E-Business Models," in W. Currie (Ed.), *Value Creation from E-Business Models*, Oxford: Butterworth-Heinemann, 65-97 (2004).

4 Achieving Predictable Performance in Multicore Embedded Real-Time Systems

Dionisio de Niz, Jeffrey Hansen, Gabriel Moreno, Daniel Plakosh, Jorgen Hanson, Mark Klein, Karthik Lakshmanan, and Raj Rajkumar

4.1 Purpose

The community at large has already recognized the risk involved in the trend to base the speedup of processors on an increasing number of cores instead of faster single cores. To harvest such a speedup enough executable elements (tasks, processes, threads, instructions depending on the level of parallelism desired) to be executed in parallel in each of the cores need to be found. New allocation, scheduling and synchronization techniques are needed in order to utilize the capacity of the cores. There are two aspects: (1) ensuring that the system maintains its predictable timing behavior when executing on a multicore, and (2) maximizing the parallelism in order to minimize idleness of some of the cores. It is paramount that we get a predictable timing of such execution to be able to guarantee deadlines. For any power-constrained or energy-conserving embedded systems it is also essential that no core is idly consuming energy (one can adjust and voltage scale a multicore chip, but not necessarily turn off a single core). On top of this, the consequence of wasting processor performance in embedded systems thus can have a higher penalty provided that cost is an important driving force in embedded systems as is the case in, for example, automotive systems, where economy of scale magnifies the cost of hardware underutilization. As a result, missing the performance improvements of multicore systems in embedded systems can have huge economic consequences. Current multicore chips contain typically four to eight homogeneous cores. Within three to five years, multicore chips will exceed 32 cores, and also be heterogeneous, that is, they will be able to run at different speeds, and possibly be tailored for specific functions.

The goal of this proposed research is to identify anomalies that can jeopardize timeliness, and the problems that prevent the full utilization of multicore systems for embedded real-time systems. Our focus is to develop analysis techniques and architectural abstractions to build real-time systems with predictable timing behavior that efficiently utilizes all the cores available in the platform. The practicality of such techniques and abstractions will be tested and evaluated by applying them to a real system from the automotive domain.

4.2 Background

In the embedded system arena, the need to preserve performance improvements and to lower cost is significant. A recent example that demonstrates how increased performance enables new features in embedded systems is the Mercedes Benz F700 concept car. Voelcker discusses how the engine of this car combines features from diesel and gasoline engines, but more importantly he highlights the fact that even though the concept had existed for years it was not until recently that the performance of the processors was sufficient to be able to implement the system that controls it [Voelcker 2008]. This is just one of the many examples where improvement in the performance of processors enables new features. Of significant importance is also the fact that the complexity and richness of features in cars, phones, aircraft, and multiple other embedded systems keep increasing along with the pressure to reduce cost.

Real-time systems are inherently parallel; this is a function of the intrinsic parallelism of the external and physical environment in which the real-time system actuates in response to the events in the environment. Real-time systems are already being designed with threads and tasks that can be executed in parallel. To be more precise, on a single processor only one task can execute instructions on the processor at any point in time. Consequently, the execution of tasks is interleaved. When a task cannot continue its execution due to a resource being held by another task, it becomes blocked. For the single-processor case the blocking can be bounded and made predictable in a number of fashions, for example, by deploying synchronization mechanisms or defining preemption points in tasks. In the case a task is blocked, the scheduler changes the execution to another task. For the multi-processor case, a task can become blocked due to a task, possibly on another core, holding a resource. An allocation and synchronization scheme needs to minimize the inter-core dependencies that can cause unpredictable blocking. Thus, when migrating legacy embedded real-time software to multicore, we need to expand our understanding of the task dependencies.

Even after threads without logical dependencies are created some other dependencies related to the access to shared memory areas can reduce the potential parallelism of these threads. While these dependencies do not explicitly impose a specific order of the executions of application threads, they restrict the execution of the code that accesses shared memory to avoid conflicts. One technique to avoid these conflicting accesses is the use of locks to ensure that only one thread accesses a shared memory area at a time (mutually exclusive access). However, locks block the execution of a thread if it wants to access a shared area that is in use by another thread. As a result, the amount of parallel computation that can be preformed is also limited by the time threads spend waiting for a lock to be released.

Mutual exclusion is of paramount importance in real-time systems because it is a source of priority inversions [Sha 1990]. In the case of multicore systems the solutions developed for single processor exhibit important inefficiencies. This is the case when tasks deployed on different cores share locks. In this case it is possible for a task τ_i to wait on a lock that is being held by a remote task τ_j. Then if the lock is not released until the last instant in which the task τ_i can meet its deadline at the end of the period (period=deadline) then a second activation of task τ_i can arrive immediately after the completion of the current activation and impose a back-to-back preemption (not considered in the RMA assumptions) against lower priority tasks. Some techniques [Rajkumar 1991] have already been created to alleviate this problem but we foresee further opportunities to improve this situation.

The scheduling of multicore systems also has important problems that need to be addressed. One of the key observations that evidence these problems is Dhall and Liu anomaly [Dhall 1978]. This anomaly occurs in a multiprocessor system using a fixed-priority global scheduler. This global scheduler has the objective to always keep the m highest priority tasks running in a system of m processors. This means that if a task becomes ready to run and all the processors are busy, it can preempt any of the running tasks with lower priority than the new one. While the objective of this scheduler seems to be obvious and straight forward to implement Dhall and Liu discovered cases that can lead to very low utilization. One such case can be constructed with m-1 tasks ($\tau_1,\ldots, \tau_{m-1}$) with computation 2ϵ and period 1 and one task with computation 1 and period $1+\epsilon$. In this case the last task τ_m cannot be scheduled on less than m processors (when using rate-monotonic scheduling to assign their priorities) even though for a very small ϵ and a large m the utilization can be

arbitrarily low. This is because if the last task τ_m is scheduled to run in a processor together with one of the other tasks, say $\tau_{i,,}$ then τ_i will run first (due to its higher priority) leaving only 1-ϵ time left for task τ_m to complete. In such a case, τ_m will miss its deadline at the end of the period.

Recent work on scheduling [Andersson 2001, 2003] resulted in an increase of the global scheduling utilization to a 33% for periodic tasks and 50% for aperiodic tasks. Some other approaches [Srinivasan 2001, Anderson 2006], have chosen to use quantized assignments of processor cycles to tasks with a scheduler that calculates a scheduling window at fixed intervals. However, none of these works takes into account the task interactions and different task and application structures. In our project we will explore the combination of task scheduling and task synchronization along with different application structures that can increase the utilization of multicore systems for real-time applications.

4.3 Approach

Our approach to address the issues of the current software development theory and practice for embedded real-time systems when using multicore platforms is as follows:

First, we characterized the limitations and applicability of current real-time scheduling theory for multicore computers. In particular, we characterized the critical anomalies coming from assumptions that are rooted in a single core and the resulting effects that those assumptions impose, for example, on predictability and system utilization. This includes the specific characteristics of multicore systems such as memory hierarchy.

Secondly, we studied the current limitations of existing real-time synchronization protocols with respect to multicore platforms and developed new protocols. This includes the exploration of new synchronization mechanisms arising from the non-real-time arena that are considered amenable for adaptation to achieve predictable timing performance.

Thirdly, we specified and analyzed architectural patterns to ensure timeliness and efficient utilization of multicore systems. These new patterns are key to unleashing the potential parallelism of the applications to seize the full potential of multicore systems.

Finally, we developed experiments to evaluate our solutions and demonstrate relevant problems specific to real-time systems. These experiments were identified with the help of Lockheed Martin.

4.4 Collaborations

Our collaborators included

- Prof. Raj Rajkumar, Carnegie Mellon University, Electrical and Computer Engineering, co-director of the General Motors Collaborative Research Laboratory (host of the Boss autonomous vehicle) and co-director of the GM Autonomous Driving Laboratory.
- Karthik Lakshmanan, a graduate student under Prof. Rajkumar.
- Ben Watson, Russell Kegley, Daniel Waddington, et al. from Lockheed Martin.

4.5 Evaluation Criteria

The evaluation of this project was based on the following deliverables:

- A paper on collaborative synchronization, allocation, and scheduling. Published in RTSS09.
- A paper on the scheduling of mixed criticality tasksets. Published in RTSS09.
- A demonstration of the mixed-criticality scheduler with a radar surveillance system.
- Application of each of our solutions to a model problem defined by Lockheed Martin.

4.6 Results

Our results can be grouped into three research topics and the application of such criteria to a model problem. We now discuss the results in each of these areas.

4.6.1 Coordinated Allocation, Synchronization, and Scheduling

Task synchronization in real-time systems is a well-known problem. Fixed-priority scheduling schemes employ techniques like priority inheritance and priority ceiling protocols to enable resource sharing across real-time tasks. Dynamic priority scheduling schemes also use mechanisms like the stack-based resource policy (SRP) to handle real-time task synchronization. In the context of fixed-priority multiprocessor scheduling, the priority ceiling protocol has been extended to realize the multiprocessor priority ceiling protocol (MPCP). Synchronization schemes have also been developed for other related scheduling paradigms like PFair. Multiprocessor extensions to SRP have also been considered and performance comparisons have been done with MPCP. This paper adopts a more holistic approach to partitioned task scheduling by explicitly considering MPCP synchronization penalties during task allocation and investigating the impact of different ECPs.

The Priority Ceiling Protocol (PCP) is a real-time synchronization protocol that minimizes the time a high-priority task waits for a low-priority one to release the lock on the shared resource, known as blocking time. When PCP is used by tasks deployed on different processors, this blocking time can lead to idle times in processors. In Figure 4-1, there are three tasks, τ_1 and τ_2 running in processor P1 and τ_3 running in processor P2. In addition, a resource is shared between tasks τ_2 and τ_3 using PCP. The figure depicts how τ_2 locks the resource at time 9 making τ_3 wait for the lock up to time 51 when the lock is released by τ_2. This waiting leaves processor P2 idle because the only task deployed there, τ_3, is waiting for the lock (known as remote blocking). Furthermore, during the time τ_2 holds the lock it suffers multiple preemptions from the higher priority task τ_1. As a consequence, τ_3 misses its deadline at time 68. Such a problem is removed if, instead of sharing the resource across processors, it is shared on the same processor, that is, tasks τ_2 and τ_3 are deployed together, say in processor P2. The key aspect of the example in Figure 4-1 is that processor utilization is wasted during remote blocking. This is because a task that could be scheduled in the remote processor is blocked leaving the cycles reserved for it idle. This contrasts with local blocking because the task holding the lock uses the cycles the blocked task leaves idle. This is the core motivation of our synchronization-aware task allocation algorithm. In the worst case, however, some degree of global resource sharing may be unavoidable. As a result, techniques to mitigate its consequences were also explored.

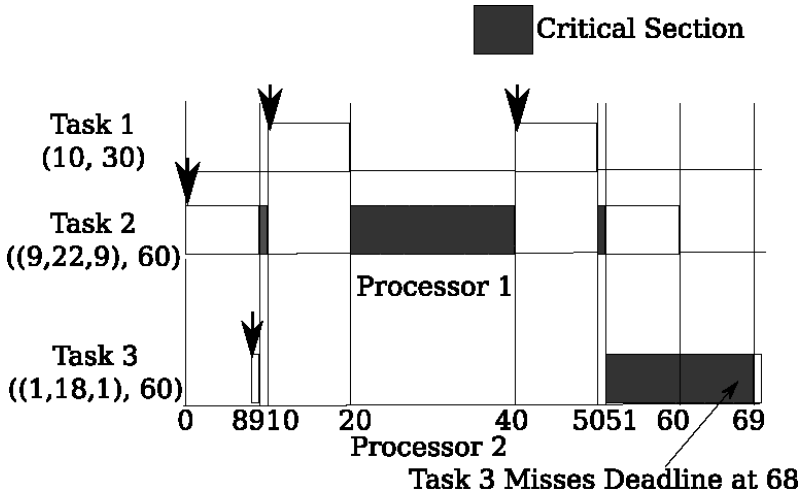


Figure 4-1: Penalty of Global Synchronization

To mitigate the remote blocking problem we considered two bin-packing algorithms, the synchronization-agnostic and the synchronization-aware algorithms. The former packs objects exclusively by size and the latter tries to pack together tasks that share mutexes. The result is that, if we allocate these tasks to the same processor, the shared mutex becomes a local mutex and local PCP can be used. The strategy of the synchronization-aware packer is twofold. First, tasks that share a mutex are bundled together. This bundling is transitive, that is, if a task A shares a mutex with task B, and B shares a mutex with C, all three of them are bundled together. Then, each task bundle is attempted to be fitted together as a single task into a processor. Secondly, the task bundles that do not fit are put aside until all bundles and tasks that fit are allocated without adding processors. At the end, these bundles are broken and fitted into the processors, adding new processors if necessary.

4.6.2 Additional Inefficiencies in MPCP

The key scheduling inefficiency resulting from the remote blocking behavior of tasks is that of multiple-priority inversions due to lower priority critical sections. For example, consider the scenario shown in Figure 4-2. Whenever task τ_2 suspends, task τ_3 can get a chance to execute, and block to access the global critical section shared with τ_1. When τ_1 releases the global critical section, τ_3 preempts τ_2 due to its higher priority ceiling and interferes with the normal execution of τ_2 twice. In the worst case, each normal execution-segment (of duration $C_{i,k}$ $1 \leq k \leq s(i)$) of a task τ_i can be preempted at most once by each of the lower priority tasks τ_j ($j > i$) executing their global critical sections released from remote processors.

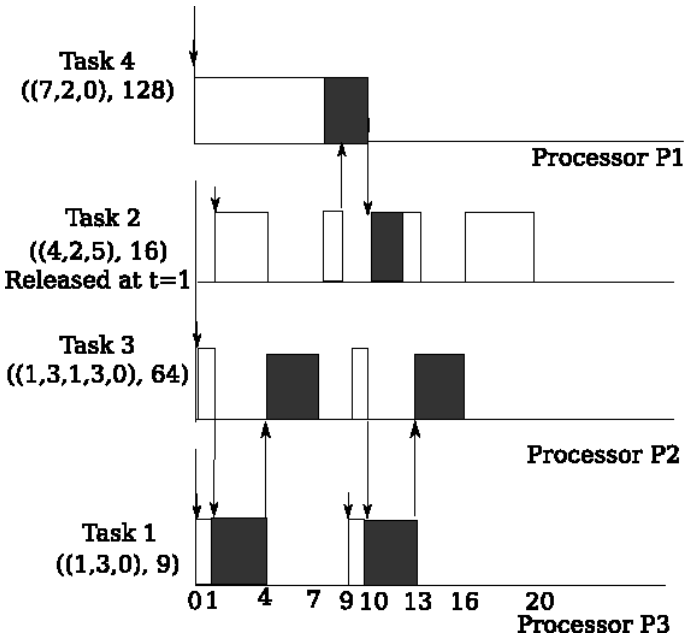


Figure 4-2: Multiple-Priority Inversions Due to Suspension

To solve the inefficiencies of MPCP, we developed execution control policies called spin-like locking. The spin-like locking prevents lower priority tasks from requesting any mutex whenever it runs during the time a higher priority task is blocked waiting for another mutex. This prevents the multiple-priority inversion by not allowing the lower priority tasks to be awakened when a mutex with a higher priority than the high-priority task.

Among the most interesting results of our algorithms is the efficiency of the synchronization-aware bin-packing algorithm. Figure 4-3 depicts the difference between our synchronization-aware bin-packing and a synchronization-agnostic bin-packing. The figure shows the number of bins needed to pack the same amount of workload (eight fully packed processors).

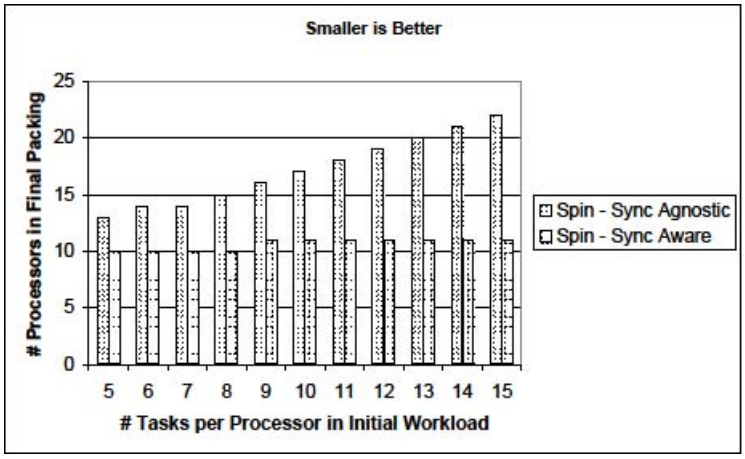


Figure 4-3: Efficiency of Task-Aware Packing

4.6.3 Multicore Real-Time Queuing Theory

Real-Time Queuing Theory (RTQT) is a method for analyzing the fraction of jobs that will miss their deadlines given a generic set of information about each task. Given the mean and standard deviation of the job inter-arrival and service times, and the mean deadline of jobs for each task, RTQT determines the fraction of jobs that miss their deadline. We use RTQT to evaluate the impact of different scheduling policies in multicore processors and the effect of dropping jobs to the general workload.

In our investigation we compared global scheduling vs. partitioned scheduling vs. single server as fast as the combined number of cores. To understand this comparison, let us first introduce the concepts of global and partitioned scheduling. Global scheduling allows a task to run on any available core. In contrast, in partitioned scheduling a task is assigned to a core and it can only run on it. We used RTQT to investigate the number of deadline misses that can be expected from these schedulers when the number of cores increases.

Figure 4-4 shows a comparison of global scheduling vs. partitioned scheduling vs. a fast processor. It is worth noting that the scheme that drops the lesser number of deadlines as the number of cores increases is the fast processor (its speed increases to a speed equivalent to the sum of the speed of all the cores). In addition, we also note that the global scheduler performs better than the partitioned scheduler.

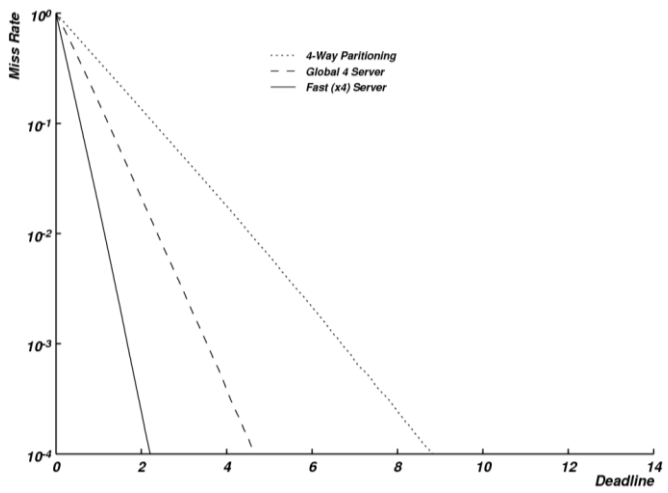


Figure 4-4: Comparison of Global Scheduling, Partition Scheduling, and Fast Server

While the results from Figure 4-4 may be discouraging from the point of view of multicore technology, Figure 4-5 shows us one of the advantages. In particular, Figure 4-5 shows that as the number of cores increases, the fraction of jobs that miss their deadline decreases. This is the case when the workload of the jobs has high variance. This implies that multicore processors are more tolerant to workload variation.

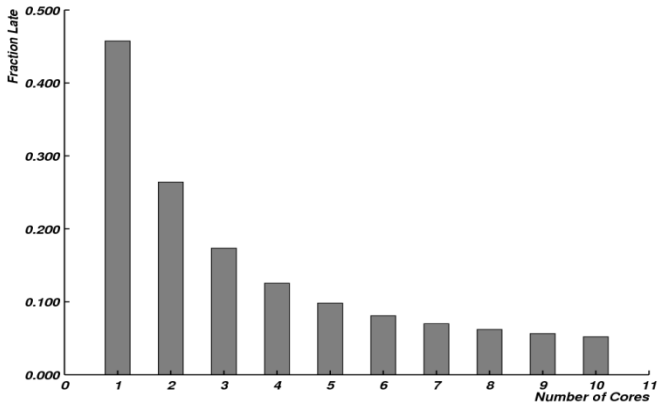


Figure 4-5: Effect of Increasing Number of Cores in Highly Variable Workloads

The second issue that we investigated with RTQT was the effect of dropping jobs that already miss their deadlines. Figure 4-6 shows the effect of two types of droppings: (1) immediate dropping as soon as the deadline is missed and (2) dropping the next periodic job. While the figure shows us that most of the benefits can be obtained when we immediately drop a job that missed its deadline, it also shows that we can still improve significantly if we drop the next job (to stop the overload). This is an important result, since the complications of stopping immediately can be significant due to locked mutexes, stopping modification to shared data, leaving inconsistent state, and related issues.

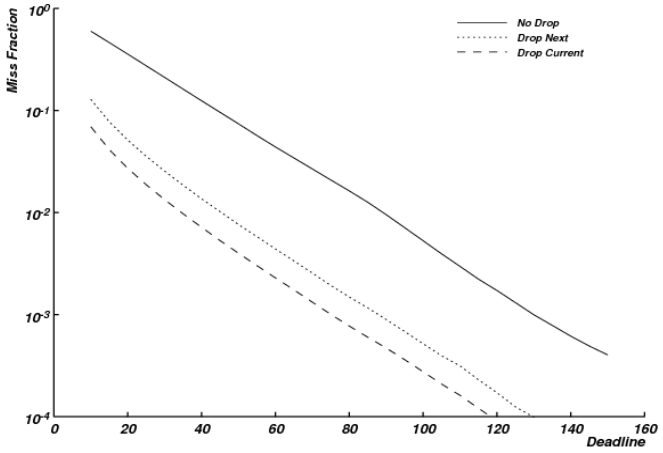


Figure 4-6: Benefit of Dropping a Job with Missed Deadlines

4.6.4 Mixed-Criticality Scheduling of Real-Time Tasksets on Multiprocessors

Priority-based preemptive scheduling policies assign priorities to tasks, and at runtime, attempt to schedule the highest priority ready task. The priority assignment can be fixed across task instances (fixed-priority schedulers) or it can change across task instances (dynamic priority schedulers). In traditional real-time scheduling, priorities are assigned with the purpose of maximizing schedulable utilization while respecting the deadlines of all the tasks in the set. The utilization maximization approach of traditional real-time schedulers makes two important assumptions: (1) all tasks are equally important, and (2) the utilization never goes beyond the allowable threshold(s). These two assumptions seldom hold in mixed-criticality systems. In particular, tasks can have different

criticality levels. Hence, if there is ever a situation where we can only satisfy the deadline of one task, then we should choose to meet the one with higher criticality. The second assumption does not hold for a more subtle reason. Specifically, in mixed-criticality systems there is a need to ensure that the tasks do not execute longer than their specified worst-case execution time. This is to prevent low-criticality tasks from interfering with higher criticality ones. While this is, in general, considered a fault, an explicit protection against it is needed because it can create (temporal) overload situations. Since the traditional priority assignment made by the scheduler was tailored to increase schedulable utilization, it is agnostic to criticality. In particular, under priority-based preemptive scheduling, a low criticality task can have a higher scheduling priority than a higher criticality task. Such priority assignment would schedule the low criticality task earlier than the higher criticality task, potentially making the latter miss its deadline. This is known as a *criticality inversion*. On the other hand, if we assign priorities based on criticality, then we eliminate criticality inversion. However, this assignment can potentially create significant priority inversion from the perspective of priority-based preemptive schedulers. To consider both scheduling priorities and task criticalities, and explicitly capture the overload execution requirements, let us define a task τ_i as:

$$\tau_i = (C_i, C^o_i, T_i, D_i, \zeta_i)$$

where:

- C_i is its worst-case execution time under non-overloaded conditions
- C^o_i is the overload execution budget
- T_i is the period of the task
- D_i is the deadline of the task (with $Di \leq Ti$)
- ζ_i is the criticality of the task. We follow the same convention as with priorities: the lower its value, the higher the criticality. It is worth noting that the overload budget C^o_i is used to create a precise definition of our guarantee, that is, how much overload is guaranteed. As we will show later, it is possible to reserve a conservative (large) C^o_i since it does not add up across criticality levels. The priority blocking utilization PB_i for task τ_i is defined as:

$$PB_i = \sum_{\tau_j | \zeta_j < \zeta_i} \frac{pb_i^j}{T_i}$$

where:

- pb^j_i is the maximum time that a higher criticality task τ_j can block the execution of an instance of τ_i during which the scheduling priority of τ_i is higher than that of τ_j . Similarly, the criticality blocking utilization CB_i for task τ_i is:

$$CB_i = \sum_{\tau_j | \zeta_i < \zeta_j} \frac{cb_i^j}{T_i}$$

where:

- cb^j_i is the maximum time that a lower criticality task τ_j can block the execution of an instance of τ_i during which the scheduling priority of τ_j is higher than that of τ_i. Both pb^j_i and cb^j_i vary

depending on how and when the scheduler assigns priorities and the strategies used to tradeoff priority and criticality blocking. In the next section we discuss the strategies taken in our new scheduling scheme.

4.6.5 Zero-Slack Scheduling

Our scheduling policy works on top of traditional priority-based preemptive real-time schedulers. It is based on the observation that criticality inversion only matters under overload conditions. We use this observation to create two execution zones for each task τ_i. In the first zone, every task is included while in the second zone, every task $\tau_j | \tau_j > \tau_i$ is suspended. This suspension effectively blocks the interference of lower criticality tasks in the case of an overload condition, up to the completion of the task activation. It must be noted that τ_i itself can also be suspended by a task $\tau_c | \tau_c < \tau_i$ in τ_c's second zone. The execution zones partition the execution of each task into two modes: the normal mode (N mode) and the critical mode (C mode). Our scheduling algorithm then calculates the execution time for each mode. We now define our scheduling guarantee: a task τ_i is guaranteed to run up to C^o_i if no higher criticality tasks exceed its C_i.

To evaluate the effectiveness of the ZSRM to schedule a mixed-criticality task set we use the concept of *laxity utilization*. The laxity utilization of a task τ_i measures the available utilization for τ_i after we discount the preemptions of higher priority tasks and the blocking factors left by the scheduling algorithm of interest for this task. The laxity utilization for a task τ_i is then defined as:

$$LaxU_i = A(i) - \frac{C_i^o}{T_i} - (PB_i + CB_i) - \sum_{\tau_j \in H_i^{hc}} \frac{C_j}{T_j}$$

where:

- A(i) is the available utilization for task τ_i for a specific priority-based preemptive scheduler. For instance, for rate-monotonic scheduling with implicit-deadline tasks, this is $n(2^{\frac{1}{n}} - 1)$
- $\frac{C_i^o}{T_i}$ is the overload density of τ_i
- $PB_i + CB_i$ is the blocking factor for the scheduling algorithm of interest
- $\sum_{\tau_j \in H_i^{hc}} \frac{C_j}{T_j}$ is the utilization consumed by the higher priority and higher criticality tasks H_i^{hc}

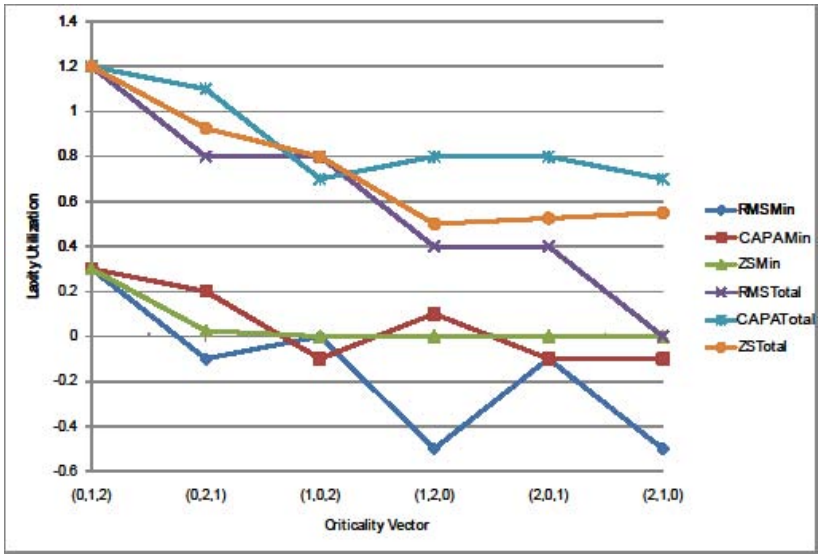


Figure 4-7: Laxity Utilization Comparison

Figure 4-7 shows a comparison of the laxity utilization between ZSRM (ZS), criticality-as-priority-assignment (CAPA), and RM scheduler. It is worth noting that while other algorithms can get more total laxity utilization, ZSRM is able to keep the minimum laxity at or above zero in all cases making the tasks schedulable.

4.6.6 Zero-Slack Bin-Packing

Multiprocessor scheduling has taken two main forms: global scheduling and partitioned scheduling. In global scheduling jobs from the same task are allowed to be executed in any processor. In contrast, in partitioned scheduling tasks are assigned to a processor and all the jobs from a particular task always run on the processor the task was assigned to. In this case, to achieve a high level of utilization, the assignment of tasks to processors must minimize the number of processors required to schedule the task set. Given that an optimal solution to this problem is NP hard, the dominant solution is a class of near-optimal algorithms known as bin-packing. As a result, the level of utilization that it is possible to achieve in a partitioned scheduling scheme depends on both the uniprocessor scheduler and the task set partitioning algorithm. Bin-packing algorithms are defined in terms of object sizes and the level of fullness of the bins where the objects are packed. These sizes and fullness levels are used to order objects and bins to try their matching with a single pass with an efficient greedy algorithm. One of the most common bin-packing algorithms is best-fit decreasing (BFD). This algorithm tries to fit each object in non-increasing order of size into a list of bins ordered in non-increasing order of fullness level. The resulting strategy is to match the largest unassigned object to the fullest bin that is able to host it. This order allows this algorithm to have a worst-case ratio against the perfect packing in the order of 11/9. In contrast, if no order is used, then the algorithm behaves as a next-fit algorithm with a worst-case bound ratio of 2.

When bin-packing algorithms are used to pack tasks, the task utilization is used as its size. This size matches the level of interference (either because it preempts the other tasks or because such cycles are reserved for the task) that the task imposes on other tasks when used with traditional uniprocessor real-time schedulers such as RMS or EDF. This level of interference can be used to

represent the level of fullness of a bin. Unfortunately, in ZSRM, tasks do not have one utilization figure but two: a normal utilization (U) and an overloaded utilization (U^o). In this case, the interference that the task imposes on other tasks depends on both their relative criticality level and their RM priority. In particular, a task τ_i, with lower criticality level but a higher RM priority than a task τ_j, imposes an interference equal to its overloaded utilization (U^o_i) to task τ_j . However, the same task τ_i imposes an interference equal to its normal utilization (U_i) to a task τ_k that has lower criticality but higher RM priority. This relative interference prevents the mapping of the utilization to the size to be used with traditional bin-packing algorithms. In this paper we present a new bin-packing algorithm with an absolute size figure that reflects the capacity of the ZSRM to schedule tasks.

To build a total order that is congruent with the ZSRM scheduler it is necessary to take into account the two aspects: (1) it should reflect an absolute size (as opposed to relative) and (2) it must reflect the opportunities for higher-to-lower criticality cycle stealing.

To use an absolute size it is enough to use the overloaded execution time of the tasks. However, this still does not provide a concept of the size of the higher-to-lower cycle stealing opportunities. To use an absolute size it is enough to use the overloaded execution time of the tasks. In order to add to the size the cycle stealing opportunities we add the total blocking of the tasks to the overloaded utilization. The two components of the total blocking of a task τ_i CB_i and PB_i quantifies the higher-to-lower criticality cycle stealing opportunities (possible reduction of the criticality blocking) and the cost of such a reduction (potential increase of priority blocking) that ZSRM incurs in a taskset. It should be noted that the total blocking of the tasks is calculated with respect to all the taskset. While this calculation provides an imprecise figure when the tasks are split into different processors, it keeps the size figure stable (and hence the order).

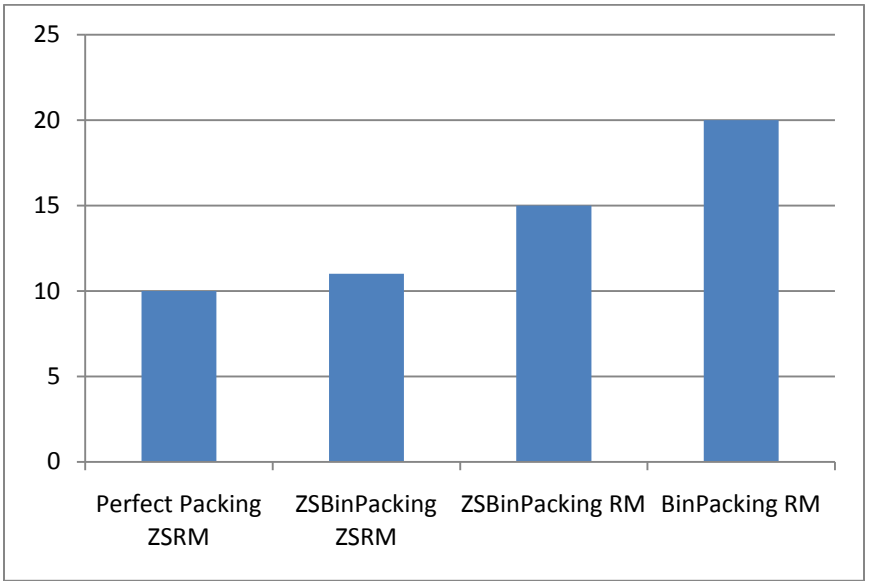


Figure 4-8: Efficiency of ZSBin-Packing

Figure 4-8 depicts a comparison of the ZS bin-packing algorithm with and without ZSRM. It is worth noting that ZS bin-packing in conjunction with ZSRM is able to use half the processors that a traditional bin-packing with RM scheduling would use.

4.7 Publications and Presentations

[de Niz 2009]
Dionisio de Niz, Karthik Lakshmanan, & Raj Rajkumar. "On the Scheduling or Mixed-Criticality Real-Time Tasksets." Real-Time Systems Symposium, December 2009. Washington, D. C.

[Lakshmanan 2009]
Karthik Lakshmanan, Dionisio de Niz, & Raj Rajkumar. "Coordinated Task Scheduling, Allocation, and Synchronization in Multiprocessors." Best student paper award at the Real-Time Systems Symposium, December 2009. Washington, D. C.

4.8 Bibliography

[Anderson 2001]
Björn Andersson, Sanjoy Baruah, & Jan Jonsson. "Static-Priority Scheduling on Multiprocessors," 193-202. Proceedings of the IEEE International Real-Time Systems Symposium, London, December 2001.

[Anderson 2003]
Björn Andersson, Tarek Abdelzaher, & Jan Jonsson. "Global Priority-Driven Aperiodic Scheduling on Multiprocessors." International Parallel and Distributed Processing Symposium. Nice, France, 2003.

[Anderson 2006]
James H. Anderson, John M. Calandrino, & UmaMaheswari C. Devi. "Real-Time Scheduling on Multicore Platforms," 179-190. *12th IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2006.

[Belloch 1996]
Guy E. Belloch. "Programming Parallel Algorithms." *Communications of the ACM 39*, 3 (March 1996): 85-97.

[Chu 2008]
Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, & Kunle Olukotun. "Map-Reduce for Machine Learning on Multicore." In *Neural Information Processing Systems 20 (NIPS-07)*, 2008.

[de Niz 2009]
Dionisio de Niz, Peter Feiler, Jorgen Hansson, & John Hudak. *Near-Optimal Cache Partitioning for Real-Time Systems.* SEI technical report, forthcoming.

[Dhall 1978]
Sudarshan K. Dhall, & C. L. Liu. "On a Real-Time Scheduling Problem." *Operations Research 26*, 1 (1978): 127-140.

[Hill 2008]
Mark D. Hill & Michael R. Marty. "Amdahl's Law in the Multicore Era." *IEEE Computer* (July 2008): 33-38.

[Hudak 2008]
John Hudak, Peter Feiler, & Jorgen Hansson. "Performance Challenges of Modern Hardware Architectures for Real-Time Systems." in *Results of SEI Independent Research and Development Projects* (CMU/SEI-2008-TR-017). Software Engineering Institute, Carnegie Mellon University, 2008. http:/www.sei.cmu.edu/library/abstracts/08tr017.cfm

[Jones 1987]
Geraint Jones. *Programming in Occam*. Prentice Hall, 1987.

[Rajkumar 1991]
Ragunathan Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, 1991 (ISBN: 0792392116).

[Rawal 2008]
G. Rawal. "Intel Multi-Core University Initiative Charges Ahead." Intel Corporation (July 2008).

[Reza 2007]
Ali-Reza Adl-Tabatabai, Christos Kozyrakis, & Bratin Saha. "Unlocking Concurrency." *ACM Queue 4*, 10 (December-January 2006-2007): 24-33.

[Sha 1990]
Lui Sha, Ragunathan Rajkumar, & John P. Lehoczky. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization." *IEEE Transactions on Computers 39*, 9 (1990): 1175-1185.

[Srinivasan 2001]
Anand Srinivasan & James H. Anderson. "Optimal Rate-Based Scheduling on Multiprocessors." *34th Annual ACM Symposium on Theory of Computing*, 2001.

[Vachharajani 2005]
Neil Vachharajani, Matthew Iyer, Chinmay Ashok, Manish Vachharajani, David I. August, & Daniel Connors. "Chip Multi-Processor Scalability for Single-Threaded Applications." *ACM SIGARCH Computer Architecture News 33,* 4 (November 2005): 44-53.

[Voelcker 2008]
John Voelcker. "Top 10 Tech Cars." *IEEE Spectrum* (April 2008). http://spectrum.ieee.org/green-tech/advanced-cars/top-10-tech-cars-2008

5 Parallel Distributed Acquisition System

Kristopher Rush, Matthew Geiger, and Cal Waits

5.1 Introduction

The CERT Forensics Team is working to develop a new method of collecting forensics data. With disk size nearly doubling each year from 1995 to 2007 forensics analysts have seen their jobs get increasingly difficult in many ways. The days of seizing a handful of drives with a total capacity of several gigabytes are over. Drive size is soaring and we are now seeing drives in excess of 1TB. The larger the capacity of the drive the longer it takes to image. Collection of forensic data is currently done on a serial basis (one drive at a time) and results in the collected data being scattered across multiple destination drives. This requires more time and manpower than is feasible by most agencies.

The industry has responded to this problem by pushing out portable (and expensive) RAID storage arrays. This is akin to taking a bigger bucket when trying to clean up a flood. It still gets filled up one disk at a time. What is really needed is a series of sponges that can be distributed simultaneously. Currently this is a gap area in the field of computer forensics and is not addressed by commercial vendors or private development efforts. This project would represent an alternative (improved) approach to the serial acquisition model deployed.

5.2 Purpose and Goals of the Research

There currently exists no commercial or government solution that enables field personnel to acquire multiple drives in parallel and to have all of the digital evidence/data written to a centrally managed collection of storage devices.

Under the current serial paradigm, field personnel acquire disk images on a one-to-one basis. For example, DISA may send out members of the Field Security Operations (FSO) Division in response to a suspected computer security incident. Once on the scene the FSO personnel determine they need to image three 1TB drives, four 500GB drives, and two laptops with 200GB drives. Typically the target drive is imaged to a separate evidence drive. Depending on equipment and training, field personnel may be able to have two sets of identical equipment operating simultaneously; this doubling of effort yields very little impact because of the serial nature of the current process. A 1TB drive will take approximately six to eight hours to hash, another six to eight hours to image, and yet another six to eight hours to verify the post image hash with the acquired image. This represents approximately 24 hours to acquire just one image. Even with a doubling of effort there would still be five days of imaging. With almost 5.5TB to acquire, the FSO team is in for a long stretch.

The proposed parallel acquisition solution will overcome these problems by achieving the following characteristics:

- The solution will provide for efficient utilization of a centrally managed set of storage media.
- It will be capable of imaging multiple target drives in parallel.

- It will use a smart algorithm to balance the DISK I/O demands across multiple storage nodes, which will achieve the most efficient READ/WRITE(S).
- It will be developed and transferred to embedded devices making it affordable and extremely deployable.

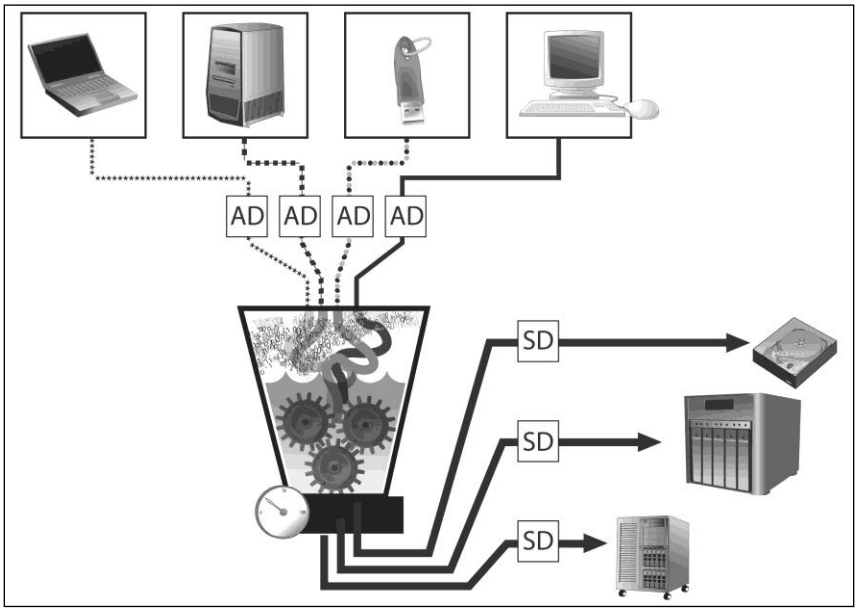


Figure 5-1 Parallel Distributed Acquisition System

5.3 Background

The CERT Forensics Team is in a unique position that enables us to interact directly with a large number of federal, state, and local law enforcement groups for both training and investigative support. Drawing on the team's past experience in federal law enforcement, intelligence, and corporate security we have become a trusted authority on issues including forensics counter encryption techniques, gap area tool development, and large acquisition/operational support. This has given us the necessary insight into current capabilities and obstacles that exist. Therefore, we are able to focus our research and development efforts in areas where true gaps in technology exist. The issue of large-scale data acquisition is an area facing real operational changes and more importantly one where little progress or research is being focused.

As data storage gets cheaper and larger the forensics community is faced with finding a solution for acquiring large amounts of digital evidence in the most efficient and effective manner possible. To date, there are no solutions that offer forensic investigators the ability to seize and acquire digital images from multiple drives in a manner that is both centralized and concurrent. This results in forensics images that are spread across numerous media and requires constant user interaction in order to image all drives. The current scenarios also require that investigators be equipped with a large variety of computers, storage, and specialized imaging equipment. All of this equipment can be very expensive and difficult to transport.

As an example of the increasing size of data to be acquired and analyzed, we recently supported a very large U.S. Secret Service (USSS) investigation. This effort included providing on-site search warrant support. This one search warrant yielded 15TB of data. It took four CERT members and three USSS agents five days to process the data. This situation posed a major challenge in acquiring and transporting the sensitive forensics data. Agents often are not given adequate hardware resources to facilitate all the facets of an investigation's list size. Had it not been for our on-site support, much of the usable data would not have been processed.

In addition to increasing size, there has been an alarming increase in the number and type of devices that require imaging at a crime scene. Desktops, laptops, external drives, DVRs, PDAs, GPS devices, MP3 players, cameras, phones, thumb drives, Xboxes, and just about any other electronic device may contain critical evidence. A typical living room can hold all of the above and more. Imaging each device one at a time can stretch the process out for days or weeks.

The CERT Forensics Team is fast building a history of bringing leading-edge solutions to our customers and to the forensics community. We have successfully created numerous forensic gap area tools that are widely distributed and routinely requested from law enforcement agencies across the globe. The Forensics Team has successfully designed and implemented the Clustered-Computing Analysis Platform (C-CAP) (http://www.cert.org/forensics/c-cap.html), a state-of-the-art forensic analysis platform, for multiple clients with increasing requests for additional units. This next project addresses another desperately needed capability missing in today's technology.

5.4 Approach

The involved team members are all members of the highly skilled CERT Forensics Team. This project combines our collective skills and abilities with our real-world experience in forensics investigations to develop a practical and relevant solution to a growing problem in the field of digital forensics. Our unique ability to engage investigators across numerous federal, state, and local investigative agencies allows us to fully capture the needs of our target audience during the development stages and will ensure a useful solution that can then be transitioned to the forensics community.

Our solution is to build a distributed system that enables field personnel to acquire multiple drives simultaneously. This will maximize efficiency and reduce response time. There are three components in our proposed design: the acquisition drone (AD), the storage drone (SD), and the central controller (CC). The ADs are attached to the suspect drive and report to the CC the size and I/O speed of the disk. The SDs are attached to the evidence drives and report similar information to the CC. The key to this system, and the focus of the research, is a smart algorithm used by the central controller to balance the bandwidth and DISK I/O to achieve the most efficient distribution of data images across the available storage media. This capability will be unique in that it will have a minimal footprint (transferring this algorithm to embedded devices) allowing for easy transportation and field deployment. This new capability will not only reduce the resources necessary to perform an acquisition, but more importantly improve both the efficiency and speed by leveraging parallel processing combined with our developed algorithm.

This work has the potential to cause a paradigm shift in the collection of digital evidence. By enabling investigators to perform fast, parallel collections to either a series of separate disks or to a large storage array we can overcome the need for a large collection of special equipment, mi-

nimize the human interaction required to image the disk and potentially use the time savings to perform further pre-processing of the data, which will give investigators the ability to begin investigations faster and find critical evidence, whether it be the be actionable intelligence data or data necessary for pursuing further law enforcement investigation or prosecution.

5.5 Success Criteria and Initial Results

1. Our initial work resulted in a working prototype of the systems as envisioned. We were able to successfully create a system that could intelligently image multiple drives over a series of closed network connections as well as provide an investigator-friendly interface for capturing metadata about the evidence. Further development of the concept is now being pursued to improve upon our existing work and to capture evidence using the same methodology, but using a single system with multiple buses. Along with this work we are investigating possibilities in pre-processing of the digital evidence in the same amount of time required to perform the drive acquisition.
2. We successfully designed and prototyped
   - a fully functioning imaging system
     - capable of at least 10 imaging nodes
     - multiple imaging nodes
     - a centralized management station
3. We designed a purpose-built self-contained field deployable case, capable of housing the system with no need for removal of any components to perform the drive imaging.

5.6 External Collaborators

There were no official external collaborators for this project. We engaged current customers during our development efforts and used their input to assist in ensuring optimal usability. We utilized graduate students from both the H. John Heinz III School of Public Policy and Management and The Information Networking Institute at Carnegie Mellon University to perform some of the initial prototyping and benchmarking. There is a solid history of graduate student work from both of these institutions that have contributed to our position as a leading-edge development group in the field of forensics.

6 Programming Models for the Multicore Era

Scott Hissam, James Ivers, Dan Plakosh, and Kurt Wallnau

6.1 Purpose

Consumers have grown accustomed to a computer industry that regularly produces new computers that are less expensive, have faster processors, and more memory. Processor manufacturers, however, have recently encountered physical limits that prevent further speed improvements. Figure 6-1 shows that clock frequency hit a wall around 2003, with excessive power consumption, heat generation, processor reliability, and rising manufacturing costs driving industry to new approaches to processor design. Among these new approaches, "multicore" is arguably the most significant, at least in terms of its potential to disrupt software engineering practice.

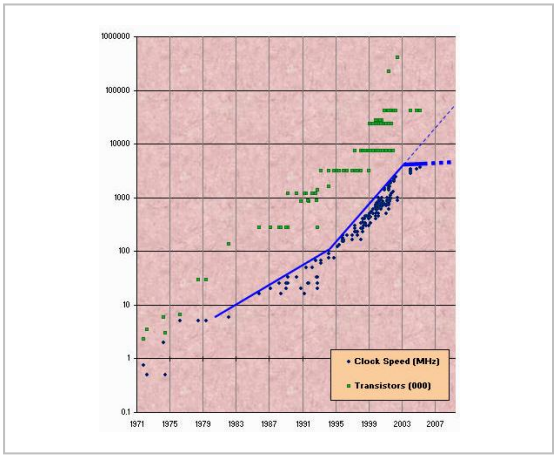


Figure 6 1: Processor Frequencies 1971-2007

Computer programs cannot simply gain the benefits of multicore "for free," though. They must be designed to exploit the potential parallelism possible with multiple independent cores, for example through informed choice of algorithms and data structures. Unfortunately, writing "parallel software" can be quite challenging, and doing so well requires specialized skills and considerable experience. Parallelism introduces a number of profound difficulties in testing, and is a notorious source of pitfalls for even the most skilled programmers,[2] and the resulting bugs can be subtle and difficult to diagnose. If multicore is to succeed, the skills required to develop parallel software must become routine rather than specialized, and the technologies used to develop parallel software must become more widely available and more robust, and must scale to arbitrarily many cores.

The purpose of this work was to understand how multicore computing will affect the broad practice of computer programming and computer programmers. Programmers are strongly affected by the programming languages, libraries, and environments that they use on a day-to-day basis, re-

[2] See http://www.cs.kent.ac.uk/projects/ofa/java-threads/0.html (last accessed December 3, 2009) for a discussion of a thread starvation "bug" in a product release of the Java software development kit. This case is revealing because thread safety is a key design goal of the Java programming model.

ferred to collectively as "programming models." We aimed to understand current and emerging multicore programming models and gaps in these models, and to identify where the SEI might have the biggest impact in filling these gaps.

6.2 Background

Multicore processors represent a fundamental change in strategy to improving computer performance, from manufacturing processors with faster clock speeds, to manufacturing processors with two or more independent processing *cores*. The premise is that additional computing cores will allow *parallel* execution of computer programs on the generally sound principle that doing two things at the same time (i.e., "in parallel") will be faster than doing one thing and then another (i.e., "sequentially"). Thus, multicore for the programmer is fundamentally about issues of parallelism.

6.2.1 Varieties of Multicore

"Multicore" covers a wide range of processor architectures, and should not be identified with the specific multicore architectural choices made by Intel and AMD in their various processor product lines.

A multicore processor has at least two independent processors on one piece of silicon that makes up a single integrated circuit. Using terminology introduced by Flynn long before multicore, our work focused on two classes of multicore architecture that are now available on conventional desktop platforms (Figure 6-2)—multiple instruction multiple data (MIMD) and single instruction multiple data (SIMD).

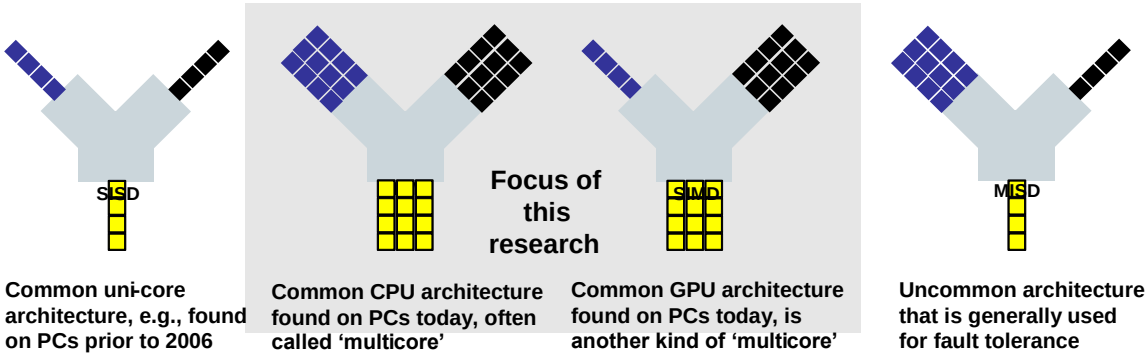


Figure 6-1: Flynn's Taxonomy

As always, however, "there is no free lunch." Conventional programming models apply to MIMD architectures, but radically different programming models are required for SIMD architectures, with Nvidia's CUDA being a well-known, but not the only, example.[3] Deciding how to decompose problems in a way that exploits the capabilities of SIMD and MIMD is challenging; reconciling different programming models to make different processor architectures work together adds additional difficulty.

[3] See http://www.nvidia.com/object/cuda_home.html for details on CUDA (last accessed December 3, 2009).

6.2.2 Concurrency and Parallelism

Though the terms "concurrent" and "parallel" are sometimes used interchangeably, there is a distinction. The difference between a concurrent and parallel program arises not from the code (in fact, they often use the same programming models), but from the execution environment:

- *Concurrent*: Pertaining to the occurrence of two or more activities within the same interval of time, achieved either by interleaving the activities or by simultaneous execution.
- *Parallel*: Pertaining to the simultaneous processing of the individual parts of the whole using separate facilities for the various parts.

Because parallelism is a special case of concurrency, we will use the term "concurrent" in places to refer to the general phenomenon where confusion will not result.

Here an important point to note is that concurrent programs that *appear* to behave correctly when running on a single processor may exhibit hidden defects when run as parallel programs on multiple processors; indeed there is substantial anecdotal evidence that this is likely to be the case even for programs written in languages such as Java that provide built-in support for programs that specify multiple independent threads of control.[4]

6.2.3 Programming Models

Wikipedia defines *parallel programming model* as "a set of software technologies to express parallel algorithms and match applications with the underlying parallel systems. It encloses the areas of applications, programming languages, compilers, libraries, communications systems, and parallel I/O."[5]

We strongly endorse the idea expressed in the underlined passage—a programming model includes much more than what is defined by the programming language. However, for our purposes, we considered the following elements of parallel programming models: theories, platforms, languages, and patterns. These elements comprehend but are not restricted to the concurrency.

Theories

These are the logical or other mathematical constructs that programmers use to reason about, specify, and possibly prove important properties of program behavior—for example, algorithm complexity and program correctness.

Theories of particular importance to multicore programming include process algebraic theories concurrency, of which there are a wide variety that differ in ways that are mostly significant to theorists rather than programmers. Significantly, the "theory" of concurrency and parallelism—at least that part of the theory of direct relevance to the programmer—is relatively unchanged since the early 1980s.

---

[4] One vendor of high-end Java hardware platforms (with hundreds of core processors) reported in a closed door briefing of IFIP Working Group 2.4 (see http://wg24.cs.uvic.ca/ContentWG24.shtml) that mission critical, multi-threaded applications currently in use in major financial institutions that appear to work correctly on uni-core hardware platforms routinely failed when executed on highly parallel architectures: parallelism exposed latent defects, most notably race conditions, in code.

[5] See http://en.wikipedia.org/wiki/Parallel_programming_model (last accessed December 3, 2009).

On the other hand, theories for reasoning about so-called real-time behavior of software are undergoing substantial renovation to accommodate multicore; see Section 6.5 on page 46 for an overview of some results in this area.

Platforms

These are the operating systems, middleware, libraries and exolinguistic tools, such as program generators, profilers, monitors, static analyzers, and testing tools, that programmers use to develop programs.

Platforms of particular importance to multicore programming include: recent enhancements of thread libraries, such as Intel's Thread Building Blocks (TBB);[6] GPU programming libraries and tools such as CUDA; and hardware and/or software transactional memory.

Languages

These are, to programmers, the most obvious and familiar components of a programming model, and include popular languages such as C, C++, C# and Java. Not all programming languages provide direct support for concurrency; C and C++, for example, have no provisions for concurrency but instead rely on platform services. Ada, Java and C#, on the other hand, provide language support for concurrency, but only as a peripheral concern of the language, and rely for the most part on primitive locking and synchronization mechanisms, virtually unchanged since their invention in the 1970s.

Various niche languages, such as Erlang and Haskell regard concurrency as a central concern, although each of these takes quite a different approach to addressing the concern. For example, Occam exposes concurrency as a fundamental unit of program composition, while Haskell uses transactional memory to make concurrency as transparent as possible to programmers. The tension between making concurrency abstractions explicit or implicit is a recurring one in current language research. The high-performance computing community is also investigating new programming languages, notably X10 and Chapel, although as noted earlier it is not clear that the concerns being abstracted by these languages (for example, non-uniform memory hierarchy) are necessarily critical beyond the supercomputing niche.

Patterns

These are the solution templates that are defined by recurring and intended interrelationships among a collection of component parts. Patterns appear at program scale and at architecture scale.

The numerous reprintings of Mattson, Sanders, and Massingill's *Patterns for Parallel Programming* in the one year since its publication reflects a keen interest by programmers to understand the basic structures programs must exhibit if they are to exploit multicore architecture. However, there remains still a paucity of guidance on how to compose parallel programming patterns, or how to combine programming patterns with architecture patterns, or how to isolate patterns from (or conversely, how to inform the selection of patterns by) the details of processor and memory architecture now exposed (but formerly abstracted) by multicore.

---

[6] See http://TBB.html (last accessed December 3, 2009).

6.3 Approach

We combined traditional market surveys and literature surveys with the development of non-trivial prototype software using selected, emerging multicore programming technology.

The prototype was developed to investigate whether existing programming models could be used to implement real-time face recognition on a mid-range multicore machine. We also used the prototype to identify challenges likely to confront programmers who attempt to

- integrate software packages that have each been designed to exploit multicore, but which have made different and possibly conflicting choices in their approaches
- understand the issues that arise when combining various programming and architecture patterns, and the relative effectiveness of specific patterns
- obtain a limited though well-grounded understanding of the maturity and usability of selected programming technologies

For the prototype we used Intel's TBB, OpenMP, and used the open source Open Computer Vision Library (OpenCV) in combination with various open source and otherwise publicly available software packages for face recognition, image scaling, and related tasks.

6.4 Collaborations

The SEI joined the Multicore Association (MCA)[7], an organization that works to create open industry standards that improve organizations' ability to develop multicore and multiprocessor systems. In particular, we focused our efforts on MCA's Programming Practices working group, where we collaborated with other volunteers (including representatives from Intel, Critical Blue, Freescale Semiconductor, imec, PolyCore, Samsung, and Virtutech) on the creation of a multicore programming practices (MPP) guide.

The MPP guide, not yet released, contains guidance on how to migrate today's sequential C/C++ applications to a multicore environment. It serves as a primer on fundamental concepts of multicore development, including

- an overview of several current technology options
- advice for analysis of existing applications
- design and implementation strategies for multicore environments
- advice for debugging and improving the performance of multicore applications

The MPP guide will be an important asset for organizations migrating existing applications to multicore in the near term. Its focus on current, established technologies rather than new research ideas provides practical help for those that must move to multicore while new approaches are still being developed. We used much of the MPP material during our experiments in migrating the face recognition application to a multicore environment, and used our experiences to help improve the MPP.

---

[7] http://www.multicore-association.org (last accessed December 3, 2009).

6.5 Observations

Our research and experiments reinforced many lessons from previous experience with concurrency, all of which are still relevant but bear repeating. Principally, the following four observations seemed key

1. More programmers need to learn to "think parallel."
2. Multicore will be prevalent, but not all programmers will see the same impact.
3. Multicore is mostly about improving performance.
4. Like other new technologies, there will be growing pains.

6.5.1 Programmers Need to "Think Parallel"

The vast majority of today's software programmers and university programs are grounded in sequential development practices. Concurrent software, though, is substantially different from sequential software. Control and coordination follow different patterns that many programmers are not as adept at creating. This is simply a reality stemming from the trends of recent decades:

- Many programmers, until recently, have had little access to parallel hardware.
- Many of today's mainstream languages, like C and C++, are sequential languages.
- Many universities have largely restricted concurrency to advanced topics or high-level courses for some time.

In short, the commercial marketplace has not had substantial demand for concurrent development skills. This is now changing.

Software must be written to exploit available parallelism. Parallel code faces different challenges in control and coordination patterns, algorithm design, and resource allocation. Good decisions for sequential applications can be bad decisions in parallel applications, and programmers need to learn how to think through these problems and embrace parallelism as a primary design consideration.

Table 6-1 summarizes some of the challenges facing programmers in a parallel world. This is not a comprehensive list, but identifies some challenges relating to different aspects of multicore programming models that may be encountered during different development activities.

Table 6 1: Some Development Challenges in a Parallel World

Developer Activities	Theory	Platform	Languages	Patterns & Architecture
Writing code	Stable, but adequate concurrency theory…but many programmers are not familiar with these.	Variety of good piecewise solutions emerging, but it is unclear how they play together or where they will interfere.	Variety of good language ideas emerging, but still shackled by legacy C/C++ codebases.	Patterns for data parallelism and task parallelism are well understood, but patterns that combine both are not.
Debugging	This is a major and unresolved problem for all aspects of programming models.			
Testing	Testing sequential code is already challenging (exhaustive testing is infeasible), concurrent code is much worse.			
Analysis	Many quality attribute theories need to be adapted for multi-core; this is not always trivial.	We don't know that new platforms will conform to analytic assumptions; aggravated by churn.	Existing tools may need serious modifications to accommodate new execution models.	What patterns lead to predictable system qualities?
Documentation and Specification		What do we need to know about platforms to use them alone or in combination? Will that information be available?	Many good, but trivial code examples are available, but more serious exemplars are needed.	How to document concurrency in architecture is not well understood, but increasingly important.
Integration		All of the familiar "COTS" integration issues of mismatched assumptions.		All of the familiar "COTS" integration issues of mismatched assumptions.
Design	Not yet at a point where we can hide concurrency theory the way we do other theories (e.g., type theory).	Can platforms be designed to be hardware-aware, but still portable?	Do languages expose important abstractions (e.g., information locality)?	New techniques and heuristics needed for problem decomposition – "thinking parallel."

6.5.2 Not All Programmers Will See the Same Impact

While multicore computing platforms are inevitable, almost unavoidable, not all programmers will see the same impact. Some communities will see much less impact than others. In some cases, such as in the HPC community and among operating system and real-time embedded programmers, there is already a great deal of experience in developing parallel software and thinking meticulously about the capabilities and limits of different hardware platforms. These communities already have many of the skills necessary to successfully develop parallel software.

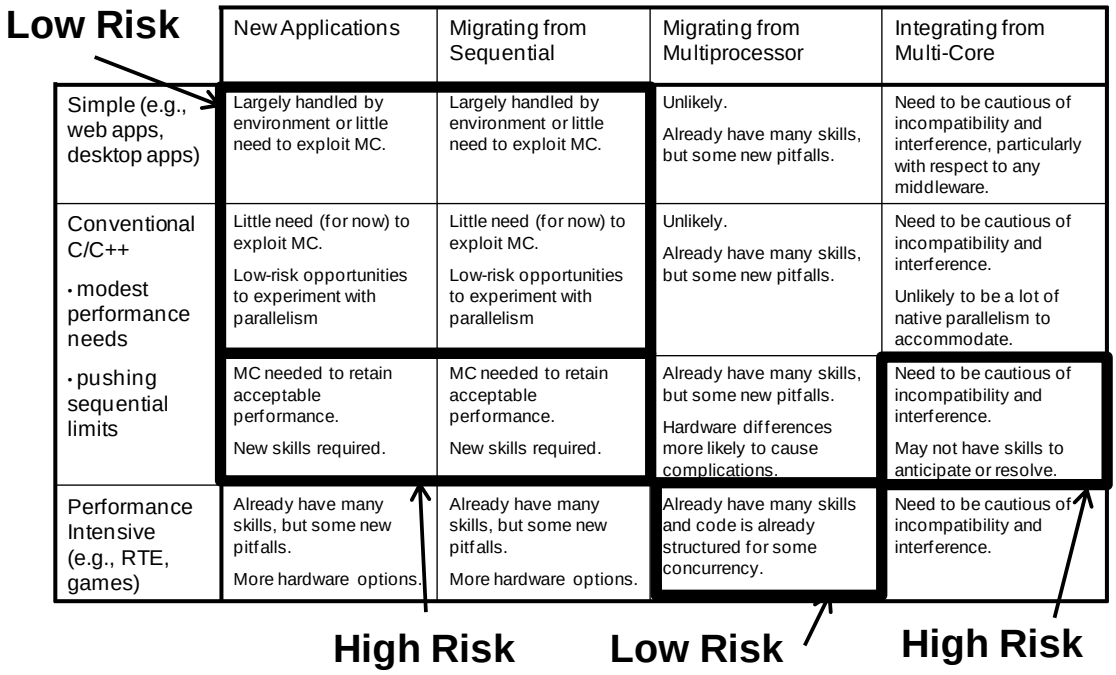
In other cases, there will be little need to explicitly manage parallelism. In some types of applications, such as web applications, middleware can and will manage much of the complexity. In such cases, programmers can continue to develop sequential software that is executed in parallel with other sequential applications. In other cases, such as some classes of desktop applications, there is no driving performance problem. If a sequential application can execute quickly enough on a single core, there is no need to reengineer it to spread the load across multiple cores.

However, there is another extreme. Those currently developing sequential software that needs additional computing resources will have to migrate to multicore. Shifting from sequential development to parallel development without prior concurrency experience will be a more difficult and risk prone endeavor.

Table 6-2 summarizes some of these trends. Columns reflect different kinds of development efforts, from writing new applications from scratch to different migration and integration possibilities. Rows reflect a crude distinction of different types of applications, largely differentiated by performance needs and likelihood of prior concurrency experience. Different cells in the table

present different kinds and degrees of risk. For simplicity we have identified regions of high, medium, and low risk, which are labeled in the table.

Table 6 2: Potential Development Impact in Different Areas



6.5.3 Multicore is Mostly About Performance

The primary reason for migrating to multicore development is to be able to exploit more cores to get work done faster—that is, it's about performance. There are certainly other benefits to using multicore, including power consumption, weight, and space benefits, but most programmers will focus on performance. This is reflected in current research and development efforts, as well, which push to keep as many cores as possible as busy as possible.

Two broad strategies dominate current discussions:

- Get the same work done faster: This strategy requires programmers to parallelize the algorithms performing work, and can be a complex activity. However, if a unit's work cannot be completed by a deadline, it is a necessary change.
- Do more work in the same period of time: This strategy exploits the fact that today's systems often do many kinds of activities, many of which have no direct or intricate dependencies. These kinds of tasks can be executed in parallel on different cores with relatively little work.

It is important to keep a balanced perspective, however, during a move to multicore. While better and better performance is an appealing notion, programmers must recognize the tradeoffs being made during parallelization. Changes that improve performance can also introduce many complexities, which potentially introduce incorrect behavior and maintenance headaches, degrade usability or portability, or compromise other important system qualities. Programmers should make these sorts of tradeoffs consciously.

6.5.4 Multicore is New; There Will Be Growing Pains

Multicore hardware and the programming models that support them are relatively new and still evolving at a rapid rate. This results in the same kinds of difficulties and risks that accompany most new technologies. For example, software that exploits particular hardware features (much more likely given the current erosion of hardware abstraction boundaries) risks being tied to particular vendors or even processor models.

Likewise, there are several different viable multicore programming models at the moment, and it is not yet clear which, if any, will dominate. Also, not surprisingly, not all models are compatible with each other. For example, attempting to integrate applications using different programming models may require dealing with different middleware, each assuming that it has exclusive access to all cores for scheduling.

6.6 Results

This study of multicore programming models confirmed many assumptions about the applicability of longstanding concurrency research and techniques in multicore environments. These findings are summarized in the previous section. Additionally, this study shed light on different approaches to addressing multicore issues and important areas for additional work.

While most multicore work is focused on improving performance, the particular techniques used present their own challenges. Two significant examples are the need to balance performance considerations with other system qualities and the ability to choose the right concurrency approaches for today's needs and tomorrow's.

It is well established that optimizing some system qualities, like performance, often comes at the cost of degrading other system qualities, like modifiability or portability. More work is needed to understand how multicore impacts the way we architect our software and to take advantage of the differing properties of different concurrency strategies.

Even absent the consideration of other system qualities, choosing a concurrency strategy is difficult. Coarse-grained strategies (e.g., parallelizing large, independent tasks) and fine-grained strategies (parallelizing loops) are both viable options, but not always as solutions to the same problem. Moreover, many complex applications can make effective use of both strategies, though more work is needed to ensure that they are used in complementary, rather than interfering ways. This issue will become increasingly important as more integration efforts have to accommodate code that has already been optimized for multicore.

The prototype real-time face recognition system was instrumental in grounding the conclusions presented here, and is itself a platform that can be used to investigate a range of programming model pragmatics.

In summary, programmers making use of multicore programming models face a complex landscape, and should be mindful of new developments and cautious in the face of new challenges. Commercial and research communities continue to make progress on addressing the challenges of developing concurrent software for multicore processors, but progress means continuing to learn and adapt.

6.7 Bibliography

[Armstrong 2008]
Joe Armstrong. *Programming Erlang: Software for a Concurrent World*. The Pragmatic Bookshelf, 2008. http://pragprog.com/titles/jaerlang/programming-erlang

[Bergstra 2001]
Jan A. Bergstra, A. Ponse, & S. A. Smolka (Eds). Handbook of Process Algebras. Elsevier Science & Technology, 2001 (0444828303).

[Buschmann 1996]
Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, & Michael Stal. *Patterns of Software Architecture*, John Wiley & Sons, 1996 (ISBN-10: 0471958697).

[Chamberlain 2007]
Bradford Lee Chamberlain, David R. Callahan, & H. P. Zima. "Parallel Programmability and the Chapel Language." *International Journal of High Performance Computing Applications 21*, 3 (August 2007): 291-312. http://portal.acm.org/citation.cfm?id=1286123

[Charles 2005]
Philippe Charles, Christopher Donawa, Kemal Ebcioglu, Christian Grothoff, Allan Kielstra, Christoph von Praun, Vijay Saraswat, & Vivek Sarkar. "X10: An Object-Oriented Approach to Non-Uniform Clustered Computing," in *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2005*, October 16-20, 2005.

[Flynn 1972]
Michael J. Flynn. "Some Computer Organizations and Their Effectiveness," *IEEE Transactions of Computing C-21*, 9 (September 1972): 948.

[Gamma 1996]
Erich Gamma, Richard Helm, Ralph Johnson, & John Vlissides. *Design Patterns*, Addison Wesley, 1996 (ISBN:3-540-57120-5).

[Halfhill 2008]
Tom R. Halfhill. "Parallel Processing with CUDA," *Microprocessor Report* (January 28, 2008). www.mdronline.com/watch/watch_Issue.asp?Volname=Issue+%23012808&on=1#item2

[Hudak 2000]
Paul Hudak. *The Haskell School of Expression.* Cambridge University Press, 2000 (ISBN: 0521644089). http://www.haskell.org/soe/

[Mattson 2008]
Timothy G. Mattson, Beverly A. Sanders, & Berna L. Massingill. *Patterns for Parallel Programming*, Addison-Wesley Professional, 2008 (ISBN-10: 0321228111).

[Sutter 2005]
Herb Sutter. "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency." *Dr. Dobb's Journal 30*, 3 (March 2005).
http://mavdisk.mnsu.edu/alleng/courses/EE%20613/Reading/No_Free_Lunch.pdf

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	December 2009	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Results of SEI Independent Research and Development Projects	FA8721-05-C-0003

6. AUTHOR(S)

Len Bass, Paul Clements, Dionisio de Niz, Peter Feiler, Matthew Geiger, Jeffrey Hansen, Jörgen Hansson, Scott Hissam, James Ivers, Mark Klein, Karthik Lakshmanan, Gabriel Moreno, Daniel Plakosh, Raj Rajkumar, Kristopher Rush, Cal Waits, Kurt Wallnau, & Lutz Wrage

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2009-TR-025

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2009-025

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The Software Engineering Institute (SEI) annually undertakes several independent research and development (IRAD) projects. These projects serve to (1) support feasibility studies investigating whether further work by the SEI would be of potential benefit and (2) support further exploratory work to determine whether there is sufficient value in eventually funding the feasibility study work as an SEI initiative. Projects are chosen based on their potential to mature and/or transition software engineering practices, develop information that will help in deciding whether further work is worth funding, and set new directions for SEI work. This report describes the IRAD projects that were conducted during fiscal year 2009 (October 2008 through September 2009).

14. SUBJECT TERMS	15. NUMBER OF PAGES
Security Architecture, Software Requirements, Multicore, Real-Time Systems, Parallel Distributed Acquisition System	51

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL