A Model Problem for an Open Robotics Controller

Scott A. Hissam
Mark Klein

July 2004

Predictable Assembly from Certifiable Components Initiative

Unlimited distribution subject to the copyright.

Technical Note
CMU/SEI-2004-TN-030

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.

Copyright 2004 Carnegie Mellon University.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

External use. Requests for permission to reproduce this document or prepare derivative works of this document for external and commercial use should be addressed to the SEI Licensing Agent.

This work was created in the performance of Federal Government Contract Number F19628-00-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.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

List of Figures

List of Tables

Acknowledgements

The authors would like to gratefully acknowledge the assistance of Anders Wall for his patience and thoughtfulness in answering all the questions raised regarding the abstractions created for the Open Robotics Controller and in working through the details so that we had the model problem correct.

Abstract

This report describes the model problem created to support the continued enhancement and development of the prediction-enabled component technology (PECT) reasoning frameworks for an industrial trial in the domain of industrial robotics. The model problem described in this report is an abstract representation of the parallel tasking and component configuration typically seen in a successful industrial robotics controller. Although motivated by the domain of industrial robotics, the model problem is applicable to other domains typified by embedded control systems consisting of both periodic and stochastic behavior and using fixed-priority scheduling with real-time performance characteristics.

1 Introduction

The Predictable Assembly from Certifiable Components (PACC) Initiative at the Carnegie Mellon® Software Engineering Institute (SEI) investigates the technology and methods for reliably predicting the runtime behavior of assemblies of components from their certifiable properties.[1] The approach to achieving assemblies that are *predictable by construction* is based on the development of prediction-enabled component technologies (PECTs) [Wallnau 03].

A PECT comprises a *construction framework* and one or more *reasoning frameworks*. The construction framework is associated with a specific component technology that provides the essential means to construct any one assembly. A reasoning framework is based on a theory and used to analyze the behavior of an assembly with respect to specific runtime properties (for example, latency, safety, or liveness).

Two previous applications of the PECT approach have been documented [Hissam 01, Hissam 02]. The first was a proof of concept for the PECT approach using a component technology developed initially to support the U.S. Environmental Protection Agency's (EPA's) Department of Water Quality. The second was an industrial trial in substation automation systems (within the domain of power generation and transmission) conducted in partnership with the ABB Corporate Research Center (ABB/CRC).

In both prior applications, the reasoning frameworks constructed to achieve predictable assembly were illustrated using the development of a model problem.

1.1 About This Report

This paper describes the model problem created to support the continued enhancement and development of the PECT reasoning frameworks for a third industrial trial. That trial is in the domain of industrial robotics and is being conducted in partnership with the ABB/CRC; ABB Automation Technology Products, Robotics;[2] and the Department of Computer Science and Engineering, Mälardalen University.

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

1 For more information on the PACC Initiative, go to http://www.sei.cmu.edu/pacc/pacc_init.html.

2 For more information on ABB robotics, go to http://www.abb.com/robotics.

The model problem described in this report is an abstract representation of the parallel tasking and component configuration typically seen in ABB's S4 generation robotics controller.

1.2 What Is a Model Problem?

Essentially, a model problem is a reduction of a design issue to its simplest form from which one or more model solutions can be investigated. A model solution is a demonstration of a design, implementation, or example that addresses the issue posed by the model problem (see Figure 1). The model problem approach is an adaptation of model problems discussed by Wallnau, Hissam, and Seacord [Wallnau 01].

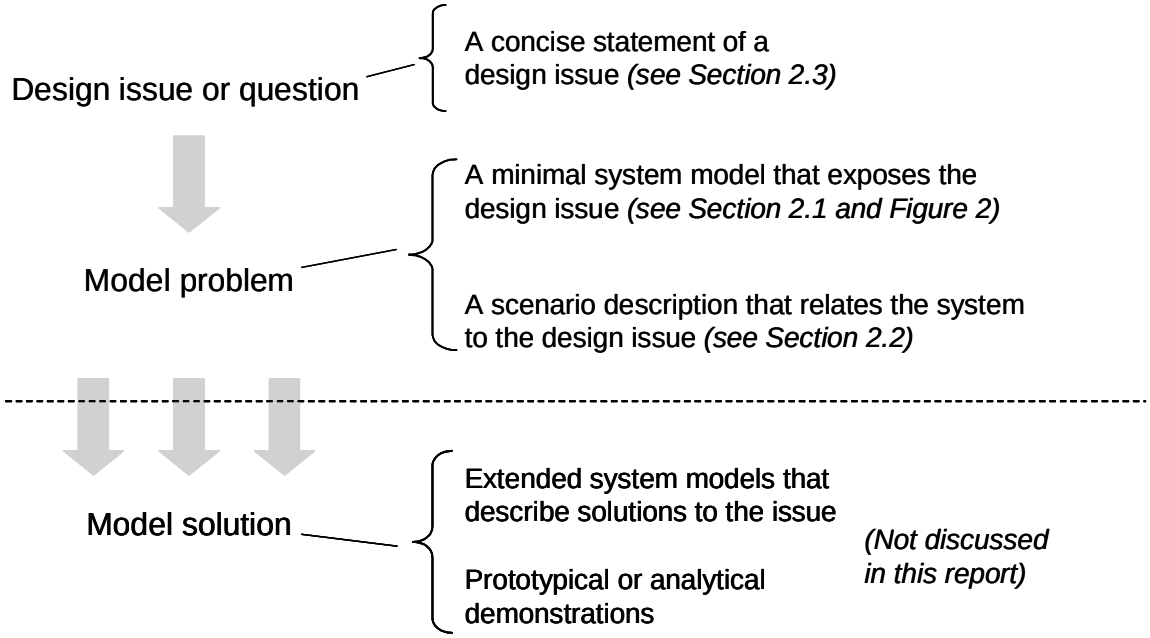

Figure 1: High-Level Structure of a Model Problem

In this report, we set up the model problem and describe the design issues, the minimal system model, and the states of the model problem significant to the design issues. Any specific model solutions to address those issues are the focus of future work and beyond the scope of this report.

1.3 About the Open Robotics Controller (ORC)

ABB Robotics is contemplating its future robotics platform, currently dubbed the Open Robotics Controller (ORC). A critical design feature of this new platform is the ability to customize the controller with user-added extensions. These extensions, made by ABB or other companies, are augmentations to the core controller platform. Although such capability enables ABB to give continued strategic business value to its customers, such extensions to

the platform could introduce technical problems in controlling or predicting the impact any one extension (once incorporated) may have on the robotics platform.

The ORC's design is understandably focused on two quality attributes: performance and safety. It is not difficult to foresee the potential poor performance or instability introduced through a user-added extension. By analogy, common off-the-shelf operating systems (OSs) permit third-party device drivers that, when flawed, cause unexpected or unwanted behavior potentially impacting quality attributes of the system as a whole. ABB Robotics wants to be able to permit the incorporation of third-party extensions to the ORC and predict the impact the extensions will have on the core controller platform's quality attributes.

The performance aspects of the ORC motivate this model problem. ABB wants to predict the invasiveness one or more extensions will have on the core controller platform *a priori*. In this way, ABB will be able to do two things: (1) know what the new performance characteristics of the ORC will be once one or more extensions are incorporated and (2) determine whether those characteristics still fall within the required performance envelope. Ultimately, such a capability will enable the creation of component specifications that vendors can adhere to, thereby ensuring that extensions will work in context without breaking the core controller platform.

2 Model Problem for the ORC

The ORC can be thought of as a number of parallel, intercommunicating threads of execution within the core controller platform. That platform typically consists of a single Intel Celeron processor running VxWorks and is referred to as the *main computer*. The main computer communicates with one or more computers called *axis computers*.

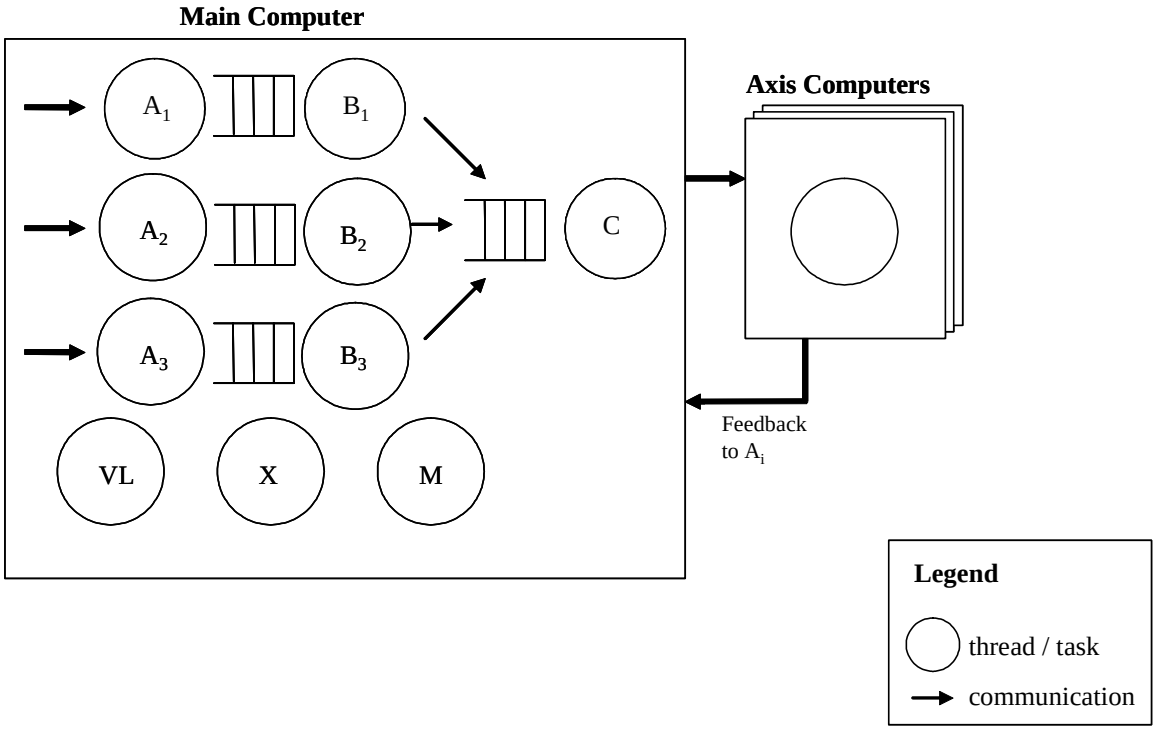

Figure 2: Tasks on the Main Computer

The model problem focuses on the interaction between tasks in the main computer. That computer is responsible for running programs (written in the high-level Rapid robot programming language) that generate work orders.[3] Those orders are decomposed into subwork orders that ultimately result in communicating microcoordinates to an axis computer that contains the device drivers responsible for the actual movement of the robotic arm (or arms).

[3] A Rapid program consists of one or more commands to a robot (much like setting a goal, such as "move here at this speed") that are broken down into one or more subwork orders (for instance, steps to achieving that goal).

The threads that execute on the main computer are a mix of periodic[4] and aperiodic[5] tasks, performing a mix of synchronous and asynchronous interthread communication. Much of the asynchronous type of communication germane to this report is conducted through first in, first out (FIFO) queues. Extensions, at this time, are envisioned to be separate threads of control. For the purpose of this model problem, only a subset of threads housed on the main computer are deemed critical. Those threads of control are shown in Figure 2 and described in the remainder of this section.

2.1 Main Computer Task Descriptions

There are six main tasks: A_i, B_i, C, X, M, and VL, each described below.

A_i tasks. Three A_i tasks—where *i* stands for 1, 2, or 3—carry out planning activities: primarily, they receive work orders and create plans. Each plan produced by Task A_i results in a sequence of subwork orders that are asynchronously passed to Task B_i. Typically, the queue size for subwork orders is set to 30 in the core controller platform. When Task A_i produces subwork orders and attempts to place one into a full queue, Task A_i becomes blocked until Task B_i removes an item from that queue, thereby making room for the new subwork order. This system behavior is considered normal.

The axis computer sends feedback to Task A_i indirectly—that is, it delivers feedback first synchronously to Task C (described below) in the main computer, and then Task C "publishes" it to those tasks subscribed to that feedback (Task A_i being one of them). Because this feedback informs the planning process; planning can't be done entirely in advance.

The arrival rates of work orders to Task A_i vary ranging from 10 Hz (i.e., one every 100 ms) to 15 Hz[6] (i.e., one every 66.7 ms). For this model problem, we assume that the arrivals are describable via Exponential(75) (i.e., exponential random distribution with a mean of 75 ms).

The execution time of Task A_i—the time needed to decompose a work order into a sequence of subwork orders—is also highly variable. For this model problem, we assume that Task A_i's execution times are describable using Exponential(9).

Currently, Tasks A_1 - A_3 execute at a relatively low priority because they have a relatively long execution time (in the aggregate). That is, although the execution time to decompose one work order is fairly short [describable using Exponential(9)], given the relatively high interarrival interval of the work orders to Tasks A_1, - A_3, a sudden large number of work

---

[4] A periodic task is one that implements the response to a periodic event (one of a sequence of events having constant interarrival intervals) and thus becomes ready to execute at fixed intervals [Klein 93].

[5] An aperiodic task is one that implements the response to an aperiodic event (one of a sequence of events ***not*** having constant interarrival intervals).

[6] This is the maximum frequency; in practice, the frequency is less.

orders could cause those tasks to monopolize the central processing unit (CPU), if the tasks are given a high priority.

The only timing requirement for the A_i tasks is to ensure that the queue between Task A_i and its respective Task B_i does not become empty during the processing of a work order (see Section 2.2).

B_i tasks. The B_i tasks—where *i* stands for 1, 2, or 3—also participate in planning. A queue between Task A_i and Task B_i contains the subwork orders placed in that queue by Task A_i. The B_i tasks have an execution time of between 1 and 2 ms and execute periodically with a period of 24 ms. Once, each 24 ms period, Task B_i will remove one subwork order from this queue, generate six individual microcoordinates, and place them in Task C's queue. Task B_i's execution time includes the time it takes to transform a subwork order to six microcoordinates and place them in Task C's queue.

The B_i tasks execute at a much higher priority than the A_i tasks do. The B_i tasks must complete their work before the end of their period.

Task C. The single Task C executes with a period of 4 ms. Task C receives microcoordinates (i.e., robot movement command) from one or more B_i tasks and sends them regularly to the various axis computers controlling the various robot arms—in this case, to three different axis computers controlling three different robot arms. (Each Task A_i - B_i pair is associated with its own robot arm.) Task C will read only one microcoordinate from the queue during its 4 ms period. The execution time of Task C is 0.5 to 1 ms, and its priority is very high. The task's deadline is the end of its period.

The queue between Task B_i and Task C must never become empty while the robot is turned on. If it does, the robot's controller will consider it to be an unsafe condition and abnormally halt the robot.

Task X. Task X represents high-priority OS functions that occur rarely and execute for a short amount of time.

M tasks. The M tasks are medium-priority tasks that represent the third-party controller extensions. M tasks are important because, on one hand, they have timing requirements of their own that must be satisfied; on the other hand, they can delay the execution of Task A_i and therefore interfere with that task's ability to keep at least one work order in the queue. For this model problem, two types of M tasks are considered: one with stochastic characteristics and one with deterministic characteristics. The characteristics of two M tasks are listed in Table 2 below. We assume that the deterministic M task (M_1) has a deadline at the end of its period, while the stochastic M task (M_2) has a soft deadline.

Task VL. Task VL represents miscellaneous work that is carried out at the lowest priority in the system. We assume that this work arrives periodically with a period of 5 seconds and executes for 250 ms (at a utilization of .02).

2.2 Significant States for the Model Problem

Although the ORC has a significant number of states in which it operates, representing all of them is not necessary for the model problem. Instead, two operating states need to be considered: *in motion* or *not in motion*. Further, the only transition is one that takes the model problem from its initial *not in motion* state to an *in motion* state.[7]

That transition is triggered after Task A_i has received the first work order and successfully placed the subwork orders in the queue between Task A_i and Task B_i. Tasks A_i, B_i, and C have different behaviors when the model problem is in a *not in motion* state versus an *in motion* state. Those behaviors are characterized in Table 1.[8]

TASK	BEHAVIOR WHILE *NOT IN MOTION*	BEHAVIOR WHILE *IN MOTION*
A_i	Waiting for the first work order[9]	Consuming work orders and producing subwork orders
B_i	Only producing "stand still" microcoordinates	Consuming subwork orders and producing "movement" microcoordinates
C	Consuming microcoordinates	Consuming microcoordinates

Table 1: Behavior of Tasks A_i, B_i, and C in Different States

Once *in motion*, we assume (for the purpose of analysis) a steady state condition never to return to a *not in motion* state.

2.3 Design Issues

Given the system context, there are two distinct design issues of interest:

1. the addition of platform extensions to the core controller platform
2. the potential for FIFO queue underflow conditions

With respect to the first issue, one of the main problems facing the ORC is how to predict the consequences of extensions. Extensions are intended to be augmentations to the core controller platform and are created by ABB or third-party "extension" developers (such as

[7] Returning to a *not in motion* state was deemed unnecessary because the design issues concerned behavior in the steady state (i.e., *in motion*).

[8] n.b.: Because the M_1, M_2, X, and VL tasks behave in the same manner for the model problem regardless of the state, they are not included.

[9] Task A_i is considered *not in motion* until the first work order is received—at which point it is *in motion*. The remaining tasks aren't considered *in motion* until after the initial subwork orders are produced and queued.

end users). For those reasons, it is critical to know what timing consequences can be introduced into the core controller platform as a result of "plugging in" an extension. A goal for this model problem, then, is to identify one or more model solutions that permit the specification of performance parameters for extensions so those extensions will not cause performance problems. In particular, extensions are envisioned for M tasks.

The second issue concerns the probability of a queue underflow exception, mainly between A_i and B_i tasks. Since A_i tasks run at a priority relatively lower than M_i tasks, the introduction of one or more extensions could conceivably starve Task A_i enough that it doesn't receive enough execution time to sufficiently feed the queue before Task B_i—thus resulting in an underflow exception. In practice, prior to the concept of introducing platform extensions to the controller, this particular queue has been known to empty in very rare situations. Therefore, an additional goal for the model problem is to identify one or more model solutions that would significantly decrease or absolutely eliminate the possibility of an underflow exception occurring for this queue.

2.4 Simplifications to the Model Problem

Based on an understanding of the execution behavior and context of the ORC, we made a few simplifications (of the abstractions presented in the prior section).

First, Task VL was deemed uninteresting because it operates at a priority below Task A_i and, therefore, cannot impact Task A_i or any other task in the system. If the starvation of Task VL becomes a design issue later, the task could be reintroduced into the model problem.

Although in the actual core controller platform, the A_i tasks cannot process all the work orders without considering feedback from the axis computer, the feedback was removed as an explicit interaction from the model problem. However, the time to process the feedback was factored into the execution time for the A_i tasks.

Lastly, to keep the model problem as simple as possible, we decided to only allow one A_i/B_i task pair (i.e., i=1). Therefore, the model problem will assume that the ORC being modeled has only one robotic arm.

However, design issues might be identified later that will dictate the reintroduction of the components we chose to remove.

2.5 Summary Task Performance Specifications

Table 2 summarizes the types of tasks in the model problem and their relevant performance characteristics.

Periodic tasks are characterized as having constant arrivals. Aperiodic tasks have random arrivals following an exponential distribution. Traffic intensity (ρ) is the quotient of expected execution time (E[S]) over the expected interarrival interval (E[T]).

TASK	PRIORITY	ARRIVALS	EXECUTION TIME	ρ = E[S]/E[T]	COMMENTS
A_1	Low	Exponential 75 ms	Exponential 9 ms	.12	Planner: takes work orders and produces subwork orders; could result from a joystick or a Rapid program. Must feed queue so that it never under runs
B_1	High	Constant 24 ms	Uniform 1-2 ms	.06	Works on subwork orders
C	Very high	Constant 4 ms	Constant 0.5-1 ms	.19	Sends command to axis computer
M_1	Medium	Constant 100 ms	Constant 10 ms	.10	Might be involved with operating specific hardware. Has a deadline at the end of its period
M_2	Medium	Exponential 100 ms	Uniform 15-25 ms	.20	Has a soft deadline of 750 ms after its arrival; it must complete before this deadline with a probability of .9.
VL	Very low	Constant 250 ms	Constant 5 ms	.02	
X	High	Exponential 500 ms	Constant 1 ms	.002	Represents rare OS services with fairly constant execution times

Table 2: Performance Description of Model Problem Tasks

3 Future Work

To address the two primary design issues discussed in this paper, we intend to apply the Sporadic Server design pattern [Sprunt 89, Gonzalez Harbour 91]. This pattern offers a possible solution for limiting the invasiveness of the introduced extensions on the remainder of the core controller platform. Further, we intend to use that pattern to guarantee a sufficient number of CPU cycles to Task A_i such that the task is guaranteed to supply the subwork order queue at a sufficient rate.

We intend to use this model problem to further the development of the performance reasoning framework for the PECT currently supporting our third industrial trial. In addition to the design issues, this work will include the following research tasks:

- Generalize our performance reasoning framework to include the use of real-time queuing theory and heavy traffic queuing theory.
- Develop a common approach to measuring task execution traces in the model problem that is sufficient and compatible with the ORC and supporting trace queries.
- Specify the construction and analysis invariants that must be satisfied by components and assemblies in order for the reasoning framework to produce valid results.

This performance reasoning framework will help predict the behavior of assemblies when the following conditions are true:

- Embedded controllers are confined to a single CPU.
- Each periodic event is handled by one task or one sequence of tasks.
- Each aperiodic event is handled by one task or one sequence of tasks.
- Aperiodic arrivals are describable by a random, exponential distribution.
- Scheduling between the tasks in the controller is managed by a fixed-priority scheduler.
- Each task is describable as having a hard or soft deadline and has a fixed execution time.

The above characteristics apply to the model problem described in this report. The problem also extends to other domains with similar characteristics.

4 Summary

The model problem approach has been used by the PACC Initiative in two prior application domains to illustrate predictable assembly through the development of PECTs. This report describes the model problem in the domain of industrial robotics (e.g., the ORC) and serves as a sufficient abstraction of a real-world system (see Section 2.1) with real-world design issues (see Section 2.3). Such an abstraction can result in potential solutions for the ORC. Although situated in the domain of industrial robotics, this model problem can also be extended to other domains and real-world situations with characteristics similar to those described in this report.

References

URLs are valid as of the publication date of this document.

[Gonzalez Harbour 91] Gonzalez Harbour, M. & Sha, L. *An Application-Level Implementation of the Sporadic Server* (CMU/SEI-91-TR-026, ADA242129). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1991. http://www.sei.cmu.edu/publications/documents/91.reports/91.tr.026.html

[Hissam 01] Hissam, S.; Moreno, G.; Stafford, J.; & Wallnau, K. *Packaging Predictable Assembly with Prediction-Enabled Component Technology* (CMU/SEI-2001-TR-024, ADA399793). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. http://www.sei.cmu.edu/publications/documents/01.reports/01tr024.html

[Hissam 02] Hissam, S.; Hudak, J.; Ivers, J.; Klein, M.; Larsson, M.; Moreno, G.; Northrop, L.; Plakosh, D.; Stafford, J.; Wallnau, K.; & Wood, W. *Predictable Assembly of Substation Automation Systems: An Experiment Report, Second Edition* (CMU/SEI-2002-TR-031, ADA418441). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu/publications/documents/02.reports/02tr031.html

[Klein 93] Klein, M.; Ralya, T.; Pollak, B.; Obenza, R.; and Gonzalez Harbour, M. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Boston, MA: Kluwer Academic Publishers, 1993. http://www.sei.cmu.edu/publications/books/other-books/rma.hndbk.html

[Sprunt 89] Sprunt, B.; Sha, L.; & Lehoczky, J. *Scheduling Sporadic and Aperiodic Events in a Hard Real-Time System* (CMU/SEI-89-TR-11, ADA211344). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1989. http://www.sei.cmu.edu/publications/documents/89.reports/89.tr.011.html

[Wallnau 01] Wallnau, K.; Hissam, S.; & Seacord, R. *Building Systems from Commercial Components.* Boston, MA: Addison-Wesley, 2001 (ISBN 0201700646). http://www.sei.cmu.edu/publications/books/engineering/building-systems.html

[Wallnau 03] Wallnau, K. *Volume III: A Technology for Predictable Assembly from Certifiable Components* (CMU/SEI-2003-TR-009, ADA413574). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports/03tr009.html

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	July 2004	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
A Model Problem for an Open Robotics Controller	F19628-00-C-0003

6. AUTHOR(S)

Scott A. Hissam, Mark Klein

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2004-TN-030

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	

11. SUPPLEMENTARY NOTES

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

13. ABSTRACT (MAXIMUM 200 WORDS)

This report describes the model problem created to support the continued enhancement and development of the prediction-enabled component technology (PECT) reasoning frameworks for an industrial trial in the domain of industrial robotics. The model problem described in this report is an abstract representation of the parallel tasking and component configuration typically seen in a successful industrial robotics controller. Although motivated by the domain of industrial robotics, the model problem is applicable to other domains typified by embedded control systems consisting of both periodic and stochastic behavior and using fixed-priority scheduling with real-time performance characteristics.

14. SUBJECT TERMS	15. NUMBER OF PAGES
model problem; sporadic server; robotics; performance; performance theory; PACC; PECT; component technology; software engineering	26

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