Open Architecture Controllers: 
Real-Time Capabilities and Interoperability

by

Cemile BuKe Otkunc

B.Sc. (Control and Computer Engineering)
Istanbul Technical University, Turkey, 1996

A THESIS SUBMITTED IN THE PARTIAL FULFILLMENT OF
THE REQUIREMENT FOR THE DEGREE OF
Master of Applied Science
in
THE FACULTY OF GRADUATE STUDIES
(Department of Electrical and Computer Engineering)

We accept this thesis as conforming
to the required standard

The University of British Columbia
June 2000
© Cemile Buke Otkünç, 2000
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical and Computer Eng.

The University of British Columbia
Vancouver, Canada

Date July 15, 2000
Abstract

Open architecture controllers are used in manufacturing systems for providing reconfigurability, extendibility, and portability of control modules. One drawback of current open architecture controller standards is that they do not include specifications for real-time requirements. Another drawback is that it is not possible to use a control application defined for one standard on another open architecture controller. Furthermore, an application running on one open architecture controller cannot communicate with an application implemented using a different standard.

In this thesis, the application programmers interfaces of two open architecture controllers, namely OSACA and ORTS, are examined in terms of their hard real-time specifications. Two interfaces are developed and implemented for exchanging data between ORTS, a small memory real-time operating system, and other open architecture control systems. The first interface provides a bridge between ORTS and the European initiated open architecture control standard, OSACA. This TCP/IP based interface enables data transfer between ORTS and OSACA applications which can be implemented on a wide range of different platforms. The second interface provides a bridge between ORTS and the open architecture CNC, ISG NC Kernel. The interface uses shared memory and is therefore advantageous for small memory, high speed applications such as intelligent machining, monitoring and control applications.
Contents

Abstract ii

Contents iii

List of Figures vi

List of Acronyms viii

Acknowledgements x

1 Introduction and Background 1

1.1 Open Architecture Controller Definition and Implementations 2

1.2 Motivation and Research Goals 5

1.3 Contributions 7

1.4 Outline of Thesis 10

2 Evaluation of OSACA from a Real-Time Perspective 11

2.1 Scheduling Frameworks 12

2.1.1 Cyclic Executive Based Scheduling Frameworks 12

2.1.2 Multithread and Multiprocess Based Scheduling Frameworks 13

2.1.3 Preemptive and Non-preemptive Schedulers 14

2.1.4 Schedulability and Rate Monotonic Scheduling 15

2.2 OSACA 15

2.2.1 Message Transport System of OSACA 17

2.2.2 Application Services System of OSACA 18

2.2.3 Communication Object Manager of OSACA 18

2.2.4 Communication Objects in OSACA - Application Protocol 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.5</td>
<td>OSACA Real-Time Constraints</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>Additional Requirements for Making OSACA API Real-Time Compliant</td>
<td>24</td>
</tr>
<tr>
<td>2.4</td>
<td>Conclusion</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>Evaluation of ORTS from a Real-Time Perspective</td>
<td>28</td>
</tr>
<tr>
<td>3.1</td>
<td>ORTS - Open Architecture Real-Time Operating System</td>
<td>29</td>
</tr>
<tr>
<td>3.1.1</td>
<td>ORTS Processes and Scheduler</td>
<td>30</td>
</tr>
<tr>
<td>3.1.2</td>
<td>ORTS Shared Memory Access and ORTS Process Communication</td>
<td>31</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Timing Facilities of ORTS</td>
<td>33</td>
</tr>
<tr>
<td>3.1.4</td>
<td>ORTS DSP Real-Time Operating System and Host System Communication</td>
<td>37</td>
</tr>
<tr>
<td>3.2</td>
<td>Discussion of ORTS from Real-Time Perspective</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>Fast Cyclic Executive</td>
<td>40</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Design Structure of FCExec</td>
<td>40</td>
</tr>
<tr>
<td>3.4</td>
<td>FCExec in Comparison with ORTS</td>
<td>44</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Adaptive Control Application</td>
<td>44</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Axis Control Application</td>
<td>45</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Advantages and Disadvantages of ORTS and FCExec</td>
<td>46</td>
</tr>
<tr>
<td>3.5</td>
<td>Conclusion</td>
<td>47</td>
</tr>
<tr>
<td>4</td>
<td>Open Architecture Data Interfaces for ORTS</td>
<td>49</td>
</tr>
<tr>
<td>4.1</td>
<td>The Data Interfaces of Other Open Architecture Controllers</td>
<td>50</td>
</tr>
<tr>
<td>4.2</td>
<td>OOI - ORTS OSACA Data Interface</td>
<td>52</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Comparison of OSACA and ORTS Application Structures</td>
<td>52</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Details of ORTS OSACA Interface</td>
<td>59</td>
</tr>
</tbody>
</table>
4.2.3 Advantages and Disadvantages of OOI ................................................. 62
4.3 PCM - Process Control and Monitoring Interface ................................. 63
  4.3.1 Serial Buffers ................................................................. 65
  4.3.2 Parallel Buffers ............................................................... 66
4.4 Sample Application for Testing the PCM ............................................. 67
  4.4.1 Adaptive Control in Real-Time ............................................... 68
  4.4.2 Adaptive Control During Simulation ......................................... 69
4.5 Conclusion .................................................................................... 71

5  Conclusions and Future Work ............................................................ 74
  5.1 Summary ..................................................................................... 74
  5.2 Future Work ............................................................................... 76

References ............................................................................................ 78
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>OSACA system platform</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>OSACA layers in comparison with ISO/OSI protocol stack</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>OSACA Server and Client Communication Objects</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>ORTS Host System Software Architecture</td>
<td>30</td>
</tr>
<tr>
<td>3.2</td>
<td>ORTS Hardware and Software Architectures</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>FCExec Hardware and Software Structure</td>
<td>41</td>
</tr>
<tr>
<td>3.4</td>
<td>FCExec shared memory distribution for DSP-PC communication</td>
<td>42</td>
</tr>
<tr>
<td>3.5</td>
<td>A Sample FCExec Application</td>
<td>45</td>
</tr>
<tr>
<td>3.6</td>
<td>A Sample ORTS Application</td>
<td>46</td>
</tr>
<tr>
<td>4.1</td>
<td>OOI Hardware and Software Structure</td>
<td>52</td>
</tr>
<tr>
<td>4.2</td>
<td>Example control application for comparing OSACA and ORTS</td>
<td>53</td>
</tr>
<tr>
<td>4.3</td>
<td>Sample implementation of the control application using ORTS</td>
<td>54</td>
</tr>
<tr>
<td>4.4</td>
<td>Sample implementation of the control application using OSACA</td>
<td>57</td>
</tr>
<tr>
<td>4.5</td>
<td>OSACA Application Data Exchange Detail</td>
<td>58</td>
</tr>
<tr>
<td>4.6</td>
<td>Sample implementation of the control application using OOI</td>
<td>61</td>
</tr>
<tr>
<td>4.7</td>
<td>Sample OOI Application Data Exchange Detail</td>
<td>62</td>
</tr>
<tr>
<td>4.8</td>
<td>ISG NC Kernel and ORTS Interface Architecture</td>
<td>64</td>
</tr>
<tr>
<td>4.9</td>
<td>Serial Buffer Messaging Protocol</td>
<td>65</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>4.10 Message Exchange Protocol for Creating Parallel Buffers</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>4.11 Test results of adaptive control during a real-time experiment</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>4.12 Test results of adaptive control during a simulation</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>4.13 System Design</td>
<td>72</td>
<td></td>
</tr>
</tbody>
</table>
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASS</td>
<td>Application Services System</td>
</tr>
<tr>
<td>CNC</td>
<td>Computer Numerical Control</td>
</tr>
<tr>
<td>CO</td>
<td>Communication Object</td>
</tr>
<tr>
<td>COM</td>
<td>Communication Object Manager</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processor</td>
</tr>
<tr>
<td>FCEexec</td>
<td>Fast Cyclic Executive</td>
</tr>
<tr>
<td>ICLP</td>
<td>Intelligent Closed Loop Processing</td>
</tr>
<tr>
<td>ISW</td>
<td>Institute for Control Engineering of Machine Tools and Manufacturing Units</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organisation</td>
</tr>
<tr>
<td>MAL</td>
<td>Manufacturing Automation Laboratory</td>
</tr>
<tr>
<td>MTS</td>
<td>Message Transport System</td>
</tr>
<tr>
<td>NC</td>
<td>Numerical Control</td>
</tr>
<tr>
<td>OAC</td>
<td>Open Architecture Controller</td>
</tr>
<tr>
<td>OMAC</td>
<td>Open Modular Architecture Controller</td>
</tr>
<tr>
<td>OOI</td>
<td>ORTS OSACA Interface</td>
</tr>
<tr>
<td>OOIM</td>
<td>Object Oriented Information Modelling</td>
</tr>
<tr>
<td>ORTS</td>
<td>Open Architecture Real-Time Operating System</td>
</tr>
<tr>
<td>OSACA</td>
<td>Open System Architecture for Controls within Automation Systems</td>
</tr>
<tr>
<td>OSEC</td>
<td>Open System Environment for Controllers</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>PLC</td>
<td>Programmable Logic Control</td>
</tr>
<tr>
<td>PCM</td>
<td>Process Control and Monitoring Interface</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real-Time Operating System</td>
</tr>
</tbody>
</table>
TEAM  Technologies Enabling Agile Manufacturing
TI    Texas Instruments
UBC   University of British Columbia
UML   Unified Modeling Language
Acknowledgements

First of all I would like to thank to my parents, Sema and Rebii, and my sister, Arbil, who provided great love and support at great distance. I would also like to thank for their confidence in me.

My research would not become reality without the very valuable supervision of Prof. Dr. Yusuf Altintas, for he lead me to the world of mechanical engineers, and provided a stable and well-progressed research base. I would also like to thank him for the friendly research environment that he has provided, and for his personal help in matters not related to my research.

I wish to express my sincere gratitude to Dr. Greg Bond, for his valuable supervision, for the valuable reference he has provided me and for his encouragement and support which were all extremely important to me and the successful completion of this thesis.

I would like to thank to Prof. Dr. Esref Adali, for his support and encouragement for this research and for leading me to the world of academia and providing me a base for computer engineering.

I wish to express my sincere gratitude to Prof. Dr. Gunther Pritschow and my colleagues Alexander Schweiker and Sebastian Fritz for their valuable support during my research at ISW, University of Stuttgart, Germany.

Also many thanks to Alvina Woods and Nesrin Altintas for their love and caring during my stay in Canada.
I would like to thank Ismail Lazoglu and Simon Parks for proof reading as I was writing the thesis. I also wish to express my gratitude to all my colleagues in Manufacturing Automation Lab most of whom are back in their home countries throughout the world, for their invaluable friendship. In this regard, I say thank you to my colleague Ahmet Gurcan for sharing his experience. I also would like to thank to my colleague Paria Noorkami for her valuable friendship and support.

My regards to my peers and thanks to both Electrical and Computer Engineering and Mechanical Engineering departments.

CEMILE BÜKE ÖTKÜNÇ

The University of British Columbia

January 2000
Chapter 1

Introduction and Background

Efficient manufacturing application design and development requires real-time, modular, portable, and reconfigurable control system hardware and software architecture. For example a control application designed for a three axis milling machine should be easily ported to a five axis robot arm by replacing the kinematics, hardware interface, and auxiliary functions.

Traditional control systems were designed as black boxes which were composed of proprietary real-time systems, communication interfaces, APIs, processors and I/O hardware schemes. The first attempts to achieve openness in control systems was the addition of a personal computer front end to these systems. Nowadays, there are international initiatives such as the European Open System Architecture for Controls within Automation Systems (OSACA) [6, 7], the Japanese Open System Environment for Controllers (OSEC) [15, 16, 17], the Canadian Open Architecture Real-Time Operating System (ORTS) [2, 3, 4], and the American Open Modular Architecture Controller (OMAC) [14], which are the results of in-progress standards.

The requirements of controls systems are addressed by Open Architecture Controllers (OAC) [1]. OACs are flexible and reconfigurable both in hardware and software. Therefore in an OAC, system components can be added, replaced, reconfigured, or ex-
tended based on the functionality required and the OAC provides a consistent application programmer interface (API) and user interface. An OAC enables rapid integration of advanced hardware and software components such as sophisticated control algorithms and higher performance sensors. In an OAC, expansion of the system is not limited to only one vendor. End users of OACs can integrate their own modules without disclosing proprietary contents to a control vendor. Integration of new functionalities improves product quality and increases system productivity.

The purpose of this research is the analysis of two OACs from their suitability to real-time applications by examining their API specifications. In particular, ORTS and OSACA are chosen for this analysis. The second objective of this research is bridging ORTS and OSACA in order to share data between them.

1.1 Open Architecture Controller Definition and Implementations

The terms used to define OACs vary among the users and developers of such systems. An open architecture control system has the following properties:

**interoperability**: the ability of peer-to-peer information exchange between modules of a control system via a standard communication interface

**portability**: the ease with which modules of a control system can be transferred among different platforms without changing capabilities of the individual modules

**scaleability**: the ability to change the performance and functionality of a control system by simply adding and removing modules
interchangeability: the ability to substitute individual modules with others

extendibility: the ease at which a third party can add new modules not provided by the original controller

real-time: the controller has to support real-time capabilities

The definition of an open control system according to the IEEE 1003.0 [22] is that it “provides capabilities that enable properly implemented applications to run on a variety of platforms from multiple vendors, interoperate with other system applications, and present a consistent style of interaction with the user”.

Different manufacturing groups from all over the world have defined their own standards for OACs and implemented different systems [1]. These include the European Community standardization effort named OSACA which started in 1990 [6, 7, 8, 9, 10, 11, 12]. OSACA aims to define a vendor-neutral, open controller architecture for the suppliers and users of control systems. OSACA was initiated by the Institute for Control Engineering of Machine Tools and Manufacturing Units, University of Stuttgart, Germany (ISW). Some of the well-known members of the consortium are Bosch, Siemens and Grundig. There are also machine tool manufacturers such as Index, Huron and Research Institutes such as WZL and FISW. Implementations of OSACA can run on OS/2, Win 3.11, Win95, WinNT, VxWorks, VxWin, SunOS, SOLARIS 2.4, FAGOR OS and OS-9. Another open architecture system, named OSEC was started in Japan in 1994 [15, 16, 17]. In USA the Technologies Enabling Agile Manufacturing (TEAM) program for Intelligent Closed Loop Processing (ICLP) sponsors an Application Programming Interface (API) workgroup to specify
standard APIs for a set of OACs [13, 14]. An OAC development and prototyping system designed in the Manufacturing Automation Laboratory (MAL), University of British Columbia (UBC), is called ORTS [2, 3, 4, 5]. ORTS is a small memory real-time operating system (RTOS) that runs on Texas Instruments (TI) based DSP boards and uses WinNT as the host system and the user interface. ORTS is used for process control and monitoring of systems such as machine tools, robots and xy-tables. Companies such as Hewlett Packard [18], Siemens, and Fanuc are also working on defining their own OAC standards. Among these OACs, OSACA and ORTS are the main platforms used in this thesis.

The similarities between the implemented open architecture platforms are:

- Decomposition of the overall control system into basic modules i.e. human machine interface, man machine control, motion control, axis control, process control to form a modular reference architecture;
- A standard API to satisfy reusability of the applications on different platforms and to prevent access to the proprietary technology information within the controllers;
- Standard guidelines for the implementation of modules in order to increase reusability and maintainability of codes and ease porting of modules to different platforms;
- A platform independent communication mechanism for data exchanging between modules of a control system;
- A tool for configuring and reconfiguring modules of a control system.
1.2 Motivation and Research Goals

It is important for users and developers of control systems to adapt new technologies to the existing control systems rapidly while keeping the systems functioning. This leads to the demand for open architecture control systems which enables controllers to be built from multi-vendor, standards based components. With openness, software or hardware modules of a control system can be rapidly added, replaced, configured, or extended based on the required functionality. For example, a PID control designed for an x/y table can be rapidly configured for a five axis robot by replacing its existing two axis cartesian coordinates trajectory generator with a five axis trajectory generator while keeping rest of the system intact. The PID control of a milling machine can be extended with a friction compensation component for improving process quality while the other components remain the same.

The target applications of OACs, such as axis control, motion control, computer numerical control (CNC), programmable logic control (PLC) are all real-time systems. However, there has not been any research on the real-time properties of OACs. One of the goals of this research is to explore OACs from a real-time perspective. In this thesis, OSACA and ORTS, two OACs which are used for manufacturing applications are examined by comparing their API’s with the specifications for a hard real-time system. ORTS has been progressively developed during the past ten years in MAL at UBC. OSACA was initiated by ISW, at University of Stuttgart of Germany. ISW and MAL have established a cooperative relationship. The research work presented in this thesis was motivated by the problems that were expressed at the meetings between the two laboratories.
As listed before, there are several projects on standardizing OACs. However, application modules implemented for one standard cannot be used on another. Also, two applications running on different OACs cannot communicate with each other. Another goal of this thesis is to bridge the APIs of OSACA and ORTS in order to enable them to exchange data and application modules transparently.

One approach for an interface between OSACA and ORTS would be to implement a scaled down version of the OSACA communication stack so that ORTS applications can run on any OSACA environment. However, due to the memory limitations of ORTS hardware and the computational overhead that the OSACA stack would introduce to the ORTS systems, it was not practical. Another possible approach would be to encapsulate ORTS as an OSACA application so that OSACA and ORTS applications could exchange data without any modification. The latter approach was implemented in MAL at UBC [21]. However, the implemented gateway was not portable, or flexible. In addition to that, it required the ORTS users to have a complete understanding of the OSACA API in order to benefit from the interface. The approach which is implemented in this thesis enables ORTS applications to communicate with different OSACA compatible machine tools and hardware without any customization of the ORTS system. It is also lightweight and supports faster execution rates.

Most CNC systems have non-standard interfaces and do not allow extendibility of the system. Such systems only allow the user to change certain parameters such as feed rate and spindle speed. By using a process control and monitoring system, such as
ORTS, the forces, sound pressure vibrations, and machine vibrations can be measured. Then the feed rate and the spindle speed of the machine tool can be changed accordingly to improve productivity and quality of the machining process. Therefore, it is advantageous to use the two systems together. In this thesis, the design and implementation of a gateway between ORTS and a commercially available CNC system, ISG NC Kernel is designed and implemented to overcome this problem.

The initial task in this research was to assess OSACA and ORTS open architecture controllers from real-time perspective. The next task was to implement OSACA on ORTS in order to run process control and monitoring applications in real-time. None of these tasks are not addressed in other research work. The real-time evaluation of OACs, and the gateways between the OACs are the results of addressing the given tasks during this research work.

1.3 Contributions

In this thesis the API specifications of two OACs, OSACA and ORTS, are compared with the specifications for a hard real-time system. The result of this analysis showed that OSACA is not adequate for real-time applications. The current specifications of OSACA is more suitable as a non real-time mode manager which controls the general states of a machine tool (such as initialize, halt, and run). It is also concluded that ORTS is not suitable for hard real-time applications which require high execution rates. For the OSACA API,
a list of suggested improvements is provided as a guideline for designing a real-time OSACA.

The thesis also describes the design and implementation of a small memory footprint, cyclic-scheduler based Fast Cyclic Executive (FCExec) control application development environment which is implemented on Texas Instruments based -Digital Signal Processor (DSP) boards. The development of the FCExec was motivated by the inadequacy of ORTS as a hard real-time system. ORTS has large memory footprint and overhead due to task switching for applications such as intelligent machining and data acquisition which do not require a multitask operating system. FCExec is currently used for a research project which involves applying sliding mode control to the piezo actuator for turning operations in the MAL at UBC. As an example of the difference in execution time overhead between ORTS and FCExec, an average control application implemented and run on FCExec can have a sampling frequency of 10 KHz, while in ORTS, the sampling frequency can only go up to 1 KHz.

Another motivation for this thesis is the need for exchanging data and application modules between different OAC standards. This thesis also describes the design and implementation of a bridge between OSACA and ORTS. The bridge is called ORTS OSACA Interface (OOI). Interfacing ORTS with OSACA enables ORTS process control and monitoring applications to be accessible by OSACA applications which are on the cell control, axis control, or human machine interface level. The implemented interface runs successfully and exchanges data between OSACA and ORTS applications. It also enables users
of ORTS to run OSACA applications from the ORTS environment without the need to learn the OSACA system. However it is observed that the overhead introduced to the overall system due to the openness required by OSACA applications results in non real-time functioning of the application interface.

This result motivated the need to bridge ORTS with the ISG NC Kernel [19, 20] via a shared memory based interface. The ISG NC Kernel is a CNC system initially prepared by the ISW, Germany as part of the ISG Software for Automation Systems package. The ISG Software package is used for the design of numerical controls (NC). It runs on VxWorks and WinNT. Now the ISG NC Kernel is a property of ISG, Industrielle Steuerungstechnik GmbH, Germany. The NC Kernel’s system structure, in general, is a cyclic executive. In this thesis, the design and implementation of a bridge between ORTS and ISG NC Kernel, called the Process Control and Monitoring Interface (PCM) is presented. The PCM demonstrates that it is possible to interface an RTOS with another CNC system in an open architecture manner without the use of large memory print and undeterministic addressing and naming mechanism that OSACA introduces. The PCM was successfully used to implement a simulation of an adaptive control application which changes the feed rate of the CNC depending on the measured forces. The adaptive control application which was implemented using ORTS was tested in real-time on the Ex-Cell-O High Speed Machining Center in Germany. The simulation showed that PCM demonstrates real-time communication capabilities for OAC applications. The development of the PCM was done by the author on site at ISW, Germany.
1.4 Outline of Thesis

The first chapter introduces the thesis topic. First, the concept of OAC is defined and the significant OACs are listed. Finally the objective and the achievements of the thesis are stated.

In Chapter 2, first the scheduling frameworks are discussed. Then, the OSACA system and its API are introduced. An analysis of the OSACA API from real-time perspective is provided. The failings of the OSACA API with respect to real-time specifications is highlighted. The chapter concludes with suggestions for improving the real-time compatibility of OSACA.

In Chapter 3, a similar overview and analysis of the ORTS system is provided. The FCExec cyclic scheduler is described. Finally, two sample applications of both systems are presented for a better understanding of applications intended for both of the systems.

Chapter 4, first presents a review of the data interfaces of significant OACs. Then, it describes the bridges developed for ORTS with two other OACs. A sample, closed loop control application is used to illustrate the approach used for the design of the bridge between OSACA and ORTS. Next, the bridge between ORTS and the ISG NC Kernel is described. This chapter also compares a simulation of an adaptive control application simulation using the PCM with results obtained from a real-time execution of the same application. The chapter concludes with a comparison of the two bridges.

The results of the research presented in this thesis and proposed future work are summarized in Chapter 5.
Chapter 2

Evaluation of OSACA from a Real-Time Perspective

One of the goals of this thesis is assessing OACs from real-time perspective. OSACA is chosen as one of the OACs to be evaluated as a result of the cooperative relationship between ISW of University of Stuttgart and MAL of UBC.

OSACA is a joint European project and is an in-progress standard which aims to simplify the process of adapting control systems to machine tools [7]. In this project, a standard interface for controllers is defined [8, 9, 10, 11]. Since controllers are real-time systems, the interface that joins different controllers has to support real-time features.

OSACA addresses some of the real-time constraints that have to be considered during the implementation of the OSACA API. However, these constraints are not sufficient as shown in the following sections. In this chapter, suggestions for a real-time OSACA API are presented.

In the following section, definitions of different scheduling frameworks are given in order to define some of the real-time concepts. In the third section, the OSACA application interface is introduced and the real-time constraints defined by OSACA are listed. Suggestions to make the OSACA API real-time compliant are presented in the fourth section.
2.1 Scheduling Frameworks

This section introduces brief descriptions of some of the real-time system concepts that are being referred to in the following sections of this chapter.

Schedulers are used in real-time systems to allocate system resources to the processes and to meet real-time constraints of the system. Multitask scheduling frameworks vary from simple cyclic executives to multithreaded or multiprocess schedulers.

In the following subsections, these three basic scheduling frameworks will be explained and their comparison will be given. Additionally, rate monotonic scheduling, which is the most common static schedulability analysis method, and schedulability will be defined.

2.1.1 Cyclic Executive Based Scheduling Frameworks

A cyclic executive is a simple program that has a fixed scheduling order. Tasks in a cyclic executive are simple procedures that are called during the execution of the cyclic executive. A cyclic executive can be run each time the timer expires or each time an interrupt is received.

One of the main advantages of cyclic executives is that there is no overhead due to task switching, task synchronization, or intertask communication. Tasks share the same address space, therefore, data shared between the tasks can be passed easily. Since no operating system support is required to run cyclic executives, faster execution times compared to the multitask schedulers can be achieved with the same hardware. The other advantage
is that timing requirements of cyclic executives are predictable and easy to calculate. However, they are difficult to upgrade and modify, because the timing characteristics of each task affects the timing characteristics of all the other tasks. Designing the main loop of the cyclic executive can be difficult if the periods of the single tasks are not harmonic with the period of the main loop. All the task periods must be a multiple of the period of the main loop. A process with a sizeable computation time may have to be split into fixed sized procedures. These constraints of cyclic schedulers result in poor software architecture designs. In other words, cyclic executives are not easily reconfigurable. Cyclic executives are still preferred in safety critical systems and low memory systems such as microcontrollers or programmable logic controllers.

2.1.2 Multithread and Multiprocess Based Scheduling Frameworks

In multithreaded scheduling frameworks, a single multithreaded program is scheduled by an operating system or as part of a language run-time system. Tasks which are threads use the same address space.

In contrast, processes in multiprocessing scheduling frameworks, use separate address spaces which results in the address space of a process not being accessible by another process. In this type of scheduling framework, interprocess communication mechanisms, which increase the computational time of an application, have to be used.

Multithread or multiprocess scheduling frameworks are also called multitask frameworks. A task represents a thread or a process. In multitask-based scheduling frameworks
tasks can be in one of four main states: running, ready-to-run, suspended waiting for a resource, or sleeping. In real-time systems, tasks are scheduled depending on their priorities.

Context switching, which occurs when the CPU is switched from one task to another, requires saving the state of the old task and loading the saved state for the new process. The context state that has to be saved during context switch is less in multi-threaded scheduling systems compared to the multiprocess scheduling systems. In the case of multithreaded schedulers, it is sufficient to save and restore the program counter, thread stack pointer, and used registers. However, in multiprocess schedulers, the process address spaces also has to be switched.

2.1.3 Preemptive and Non-preemptive Schedulers

In real-time applications, tasks have priorities which are assigned using different methods. In the cyclic executive approach the notion of priority is not applicable since each task is a procedure. In the multithreaded or multiprocess approach, processes can be released at any time. If a higher priority task can take the processor from a lower priority process when the higher priority process is released then the scheduler is called preemptive. In non-preemptive schedulers even if a high priority task is released, it is not scheduled until the low priority task finishes the execution.

In real-time systems, if a task is given a high priority then it the task is required to run before the lower priority tasks. Preemptive schedulers enable high priority processes to
be more reactive because they run when they are released if no other higher priority task is occupying the processor.

2.1.4 Schedulability and Rate Monotonic Scheduling

The basic problem in real-time systems is to make sure that tasks meet their deadlines while allocating resources to them [23]. Schedulability is the measure for this requirement. Rate monotonic scheduling is a simple and optimal priority assignment scheme [26]. In this scheduling approach, the task with the shortest period takes the highest priority. If a set of tasks cannot meet their deadlines when their priorities are set with rate monotonic scheduling then they cannot be scheduled with any other fixed priority assignment.

2.2 OSACA

In OSACA, the system platform is defined to be composed of the system hardware, the operating system, and the communication system. This hierarchy can be seen in Figure 2.1. The operating system provides independence from the specific hardware and also the basic operating system functionality such as task scheduling and task synchronization. The communication system provides data exchange between modules in a standardized way. Portability of applications is guaranteed with the use of a standard API. The application programs in OSACA are designed in an object oriented software architecture. For this reason, they are named Architecture Objects. In the remainder of this thesis, the terms “Architecture Object” and “Application Program” will be used interchangeably. The OSACA communication system with the application programs is shown in Figure 2.1.
OSACA defines a standard communication system in order to meet the requirements of an open architecture system. The OSACA communication system is based on the seven layer Reference Model for Open Systems Interconnection (OSI) adopted by the International Standards Organisation (ISO) [22]. A comparison of the OSACA communication system layers and the seven layer ISO/OSI Reference Model is shown in Figure 2.2.

In the OSACA API, the message transport system (MTS) provides an ISO/OSI Reference Model layer 4 interface for the application services system (ASS). The MTS is responsible for transparent transport of messages between communication partners. The ASS provides the ISO/OSI Reference Model layer 7 interface for message processing. These two layers are absolutely necessary for OSACA conformance. The communication object manager (COM) is an optional layer which allows uniform access to the information in the applications in order to simplify application development.
2.2.1 Message Transport System of OSACA

The MTS is the only hardware dependent layer in the OSACA communication system.

The MTS uses a connection oriented protocol to communicate with other MTS peers. The messages that have to be transferred between architecture objects are transported via the MTS. A message at the MTS level is called a protocol data unit (PDU). The MTS can receive the messages from the underlying system in either polling mode or event driven mode. In polling mode, the MTS checks the underlying system communication mechanisms, which can be network sockets or message queues, to see whether a message is available. However, in the event driven mode the underlying system notifies the MTS
when there is an available message. The MTS is also responsible for buffering messages if the related architecture object or the MTS is not ready to receive it.

2.2.2 Application Services System of OSACA

The ASS is responsible for message processing such as assembling and disassembling messages and data conversion.

The ASS uses a connection oriented protocol for communicating with other ASS peers. The ASS offers asynchronous and synchronous connection services for information exchange between applications. Synchronous services block the caller until a reply is received. Time-outs can be used for limiting the blocking. Asynchronous services can be used either with or without confirmation.

If the applications are running on the same ASS, messages are transferred through the ASS, i.e. the MTS is not used. If the applications are not on the same ASS then the data is split into several PDUs and transferred via the MTS. The received messages are built up from the PDUs.

2.2.3 Communication Object Manager of OSACA

The COM is an advanced interface for applications.

The COM is responsible for the administration of applications. The COM controls the execution of application specific management tasks which are necessary to execute services in an architecture object (user application) that were requested through the com-
munication system. Creation, deletion, activation, and protection of architecture objects are the management tasks that the COM carries out.

2.2.4 Communication Objects in OSACA - Application Protocol

Any data that is going to be accessed from other applications is presented as an object in OSACA applications. There are a predefined set of object classes, and a fixed set of services (object member functions) for each object class. This is named Object Oriented Information Modelling (OOIM) in OSACA terminology. The defined object classes and their functionality are as follows:

- variable object: for accessing data
- process object: to handle state machines in an architecture object
- event object: for asynchronous notification of variable or process objects when a message is received
- transaction object: for handling reports from event objects

The services of each object are predefined by the OOIM. The communication system (MTS + ASS) is responsible for exchanging information between the COs. These services can be synchronous or asynchronous. When a client uses a synchronous service it blocks until the response is received. However, when the client requests an asynchronous service it continues execution immediately after the request without waiting for the response.
Any architecture object that is going to provide information to another architecture object is a server. If a client architecture object wants to access a server CO, the client has to have a corresponding CO. So all the shared information is composed of a server and a client communication object (CO) pair. This structure is shown in Figure 2.3.

**Variable Objects:** These are used to make server application data visible and accessible to any client application. The variable objects have *set* and *get* services to read and update their data.

**Process Objects:** OSACA supports state machines to control applications. While the server process object defines the state machine, the client process object controls the state transitions of the server application using the state machine. This forces application programmers to put their applications in a state machine based framework. The current state of the state machine defined in the process object is changed via the use of the *action* service.
**Event Objects:** In order to read the current value of a variable object, the \textit{get} service of the variable object can be used. If the client wants to be notified when the value of a server object has changed, or at a certain period, event objects are used.

**Transaction Objects:** These are used on the client application for handling reports that are received by the event objects. The application programmer has to define the processing of the received information.

### 2.2.5 OSACA Real-Time Constraints

Target users of OSACA are control application developers and machine tool builders. An interface designed for these real-time systems has to support real-time. In this subsection, the real-time constraints which are defined as part of the OSACA API are listed.

#### 2.2.5.1 Constraints Related to Message Transport System

1. An MTS can be used by more than one ASS layer. If there are several threads accessing the same MTS layer, then it is required that platform programmers provide and use \textit{enterCritical} and \textit{leaveCritical} functions for controlling access to shared objects. The details of the implementation of these two functions are left to the implementers of the API.

2. A high or low priority can be assigned for the services of the MTS layer. If the ASS uses services of different priorities, a separate ASS connection has to be established for each priority level on the MTS.

3. The MTS is the service provider for the ASS layer. These services can be provided
in either callback mode or listen mode. In the callback mode, the MTS calls the ASS when a message is received. In the listen mode, the ASS checks to see whether the message has been received or not. In the listen mode, the ASS uses polling which is not a desired method because of the inefficient CPU usage and because the computational time of other processes are used up.

4. MTS has to provide layer 4 functionality of the ISO/OSI Reference Model to the above layers. Therefore the MTS has to cover the layer 4 functionality that the underlying system does not support. The MTS can be designed in different operating modes, namely, multithreaded operation mode, callback mode, and polling mode. The platform designer decides which mode to use. The functions, such as task handling, semaphore management, timer management, and transport system specific calls which are to be used for these different modes define the real-time capabilities of a system but are not specified by the OSACA standard.

5. The MTS has to be designed to support synchronous ASS services. Therefore, it has to provide a mechanism for blocking service execution until a response is received. This can be accomplished using semaphores, condition variables, or monitors [25]. After the response for a blocking service is received, the blocked task is woken up by calling the \texttt{wakeUp()} function. Also to avoid deadlocks, a time-out function \texttt{(setTimeOut())} has to be included in this blocking mechanism.

6. It is defined in [7] that in order to support real-time applications the MTS has to conform to the following requirements:
• deterministic execution of services
• integration of a real-time transport system
• minimization of system calls, avoiding deadlocks

2.2.5.2 Constraints Related to Application Services System

The ASS is a passive layer since it never independently calls the services of the upper layer (COM) or the lower layer (MTS) but it is always commanded from other layers.

1. The ASS utilizes two different priority levels that are defined in the MTS for exchanging data. Therefore, specific functionality of the underlying transport system has to be used in order to provide different priority levels.

2. If several threads use the synchronous services of one ASS layer, all critical sections in the ASS have to be protected via a synchronization mechanism which is defined by the platform designer.

3. The services that the ASS provides to the MTS can be run either in the callback mode or the listen mode. The mode used for the implementation of MTS effects the timing requirements of real-time systems.

2.2.5.3 Constraints Related to Communication Object Manager

The OSACA API states that the COM does not have a thread attached to it. However the COM has to execute the following tasks:

• Updating event objects of the client if any change request is received from the server and sending event reports to the client if any server CO value has changed.
• Executing state transitions of the server process object that have been requested either internally or from a client.

Considering the tasks that the COM has to carry out the following real-time constraints are defined for the COM:

1. The tasks that COM has to handle have to be called explicitly since there is not a thread attached to them. If they have to be updated periodically, then they have to be called periodically.

2. Instead of using a polling mechanism, a semaphore mechanism, which is controlled with \textit{wakeUp (V)} and \textit{wait (P)}), functions are defined for the COM. The \textit{wait} function is called by the COM when external \textit{get} requests are received and the \textit{wakeUp} function is called when external \textit{action} requests are received. The \textit{wakeUp} function may also be called by the user to enable the COM to continue execution.

2.3 Additional Requirements for Making OSACA API Real-Time Compliant

As a result of the constraints listed in the previous section, there are a number of issues that have to be considered regarding the real-time aspects of OSACA. OSACA leaves the details of implementation of the threads, synchronization mechanisms, and messaging systems to the OSACA platform designer. However, these are the issues that define the real-time characteristics of a system.
With the current specifications, OSACA is more suitable as a non real-time mode manager which controls the general states of a machine tool such as initialize, stand-by, run and shut-down. OSACA can be used as a real-time system only if the underlying transport system is real-time. It is not practical to use OSACA as a real-time system for intelligent machining i.e. assisted process control and monitoring where fast sampling periods (microseconds) are needed.

To support hard real-time operation the OSACA API has to be extended by considering the following points:

- It is stated in the OSACA API specifications that the MTS can use a multithreaded or cyclic executive scheduler. As explained in the definitions, multithreaded scheduling frameworks do not waste CPU cycles polling for events like cyclic executive scheduling frameworks do. Therefore, real-time OSACA should use a multithreaded scheduler.

- As explained in the section on real-time constraints related to the MTS, the MTS has to support synchronous ASS services. In order to provide deterministic execution times and to maximize schedulability these services should use protocols such as the ceiling protocol [25], priority ceiling emulation, or absolute time-outs [25] which are suitable for synchronization. The ceiling protocol guarantees schedulability, avoids priority inversion, deadlocks, and chained blocking. Absolute time-outs are required for deterministic execution times.

- Only two priorities are currently defined for the MTS services. However, in real-
time applications it is desirable to assign each task a unique priority level. Poor schedulability results when tasks are forced to share priority levels.

- Current OSACA API specifications do not require a specific scheduling framework for task and resource allocation. A preemptive, prioritized scheduling framework has to be used to maximize schedulability.

- OSACA API specifications do not define the synchronization mechanism to be used in the implementation of the API. However, synchronization mechanisms affect the schedulability and deterministic execution of a real-time system. A synchronization protocol such as priority ceiling emulation or ceiling protocol has to be used so that deadlocks and priority inversion can be prevented and blocking of high priority processes can be bounded.

- Currently OSACA requires that the communication between the MTS and the ASS to be either in the callback or the polling mode. OSACA also suggests that polling mode should not be preferred due to the excessive usage of CPU time. For schedulability purposes the execution time of these services, either in callback or polling mode should be bounded by the OSACA API.

- The MTS supports synchronous and asynchronous ASS services. Since it is important to know the total execution time of a service for schedulability concerns, these services should be limited to synchronous services only.

- OSACA specifications do not include a delay function which is important in defining the timing behaviour of real-time systems. An absolute delay function has to be
defined as part of the OSACA API to prevent jitter [25].

2.4 Conclusion

The OSACA API offers the services of the OSACA system platform with a standard interface. By using the OSACA API, portability, scaleability, and interchangeability of manufacturing control applications are provided. The API hides the details of the communication system, operating system, and hardware from the application programmers.

From an application programmer’s point of view, the OSACA interface simplifies the task of implementing control applications. However, the details of the implementation of OSACA platform are flexible and totally dependent on the underlying transport system which may be composed of both the operating system and the communications system. Even though the OSACA API is intended for real-time systems, it does not necessarily provide real-time support unless the underlying transport system is real-time. Therefore, OSACA API has to be enhanced further in order to conform to these requirements.
Chapter 3

Evaluation of ORTS from a Real-Time Perspective

In this chapter another open architecture control system, ORTS [2], is introduced with its real-time facilities. ORTS has been progressively developed at the author's laboratory, MAL at UBC. The real-time conformance of ORTS is discussed in terms of the scheduling protocol, the synchronization protocol, the methods for delaying tasks, the real-time clock, and the communication mechanisms. Improvements to the current ORTS system that were implemented as part of this research are presented in this chapter. In particular, a deadline miss warning function, and a sleep function that prevents cumulative drift are described.

The real-time evaluation of ORTS reveals some shortcomings. As a result, a cyclic executive based real-time scheduler, FCExec was designed and implemented as part of the thesis. Two sample applications, which are implemented on each system, are given to emphasize the use of the two systems.

The chapter starts with a discussion of ORTS in terms of real-time conformance and proceeds with the description of additions to the ORTS system that were done as part of the thesis research. Next, the implementation of the Fast Cyclic Executive (FCExec) real-time system is shown. The two systems are compared to each other in the last section.
The section also contains two sample applications which are given to show the usage of the two systems.

### 3.1 ORTS - Open Architecture Real-Time Operating System

ORTS [2] has been progressively developed at MAL, UBC. It is composed of a RTOS that runs on TI-C3X processor based DSP boards and a WinNT based host application. The host application, which is shown in Figure 3.1, is based on WinNT threads and serves as an interface between the user and the real-time kernel. The host application is used for data logging, receiving user inputs, application configuration, and reconfiguration. The real-time hardware and software structure of ORTS, which is located on a DSP board, is shown in Figure 3.2. The user processes, kernel processes, and device handlers are implemented and run independent of the underlying hardware. The host system and the real-time kernel communicate through a dual-port memory which is located on the DSP board. Multiple ORTS systems or an ORTS system and a WinNT host application can be connected using WinNT pipes [28] based ORTS communication mechanisms.

The software architecture of ORTS enables the system to be open and easily ported to different platforms. ORTS has already been ported to three different TI C3x based DSP boards, namely the Spectrum PCC31 33MHz, the Spectrum PCC32 50MHz and the Spectrum Indy 60MHz.
3.1.1 ORTS Processes and Scheduler

A process can be in one of the four main process states: ready-to-run, suspended, waiting or sleeping. The processes are managed in three main lists:

1. Prioritized ready-to-run list
2. Time ordered sleep list
3. Waiting processes list for each semaphore

If a process is in the ready-to-run state then it is placed in the *ready-to-run list*. If the process is sleeping either for the end of its period or voluntarily, then it is placed in the *sleep list*. If the process is blocked on a semaphore for accessing a resource, then it is placed in the *waiting processes list* of the semaphore.

ORTS has a round-robin, preemptive, and prioritized scheduler. There are eight priority levels defined for processes. A circular queue is defined for each priority level. The
scheduler goes around the highest priority level queue, allocating CPU to each process for a predefined time interval which is called the \textit{TIME\_QUANTUM}.

\subsection{ORTS Shared Memory Access and ORTS Process Communication}

ORTS uses semaphores for controlling access to shared resources. The priority inheritance protocol is used for avoiding unbounded priority inversion. However, if a task is sharing more than one resource with lower priority tasks, chained blocking can occur and the higher priority task can be blocked each time it needs to access a shared resource.

ORTS offers \textit{links} for communication among tasks. Links are shared buffers which are accessed via semaphores. Multiple producers can write to the same link and multiple consumers can read from the same link. Links are unidirectional, therefore, if a task is
required to both send and receive data to and from another task, then two links have to be defined between the tasks.

Links can be synchronous, asynchronous, or buffered. A buffered link is a synchronized queue with a user defined size. Consumers can read from a buffered link only if the link is not empty and if there is no other task reading from the link. Producers can write to a buffered link if the link is not full and if there's no other task writing to the link. Synchronous links are same as buffered links with a queue size of one. In asynchronous links, concurrent access of more than one process to the same link is prevented. However, producers can destructively write to a full link and consumers can re-read old data if no new data is produced.

3.1.2.1 Priority Inversion Protocol in ORTS

ORTS uses a prioritized preemptive scheduler. In prioritized schedulers, priority inversion occurs when the execution of a ready-to-run task is delayed by a lower priority task. During priority inversion, the effective priority of the high priority task is same as the priority of the low priority task. ORTS uses priority inheritance protocol for bounding priority inversion. In the priority inheritance protocol if a higher priority task is suspended by a lower priority task then the priority of the lower priority task is increased temporarily to the same priority as the higher priority task. Priority inheritance prevents a medium priority process from starting execution when a high priority task is suspended.
3.1.3 Timing Facilities of ORTS

Given the importance of time in real-time systems, the introduction of time to ORTS system has to be discussed as part of its real-time aspects. ORTS has the necessary functions and variables for interfacing with time such as access to a clock, delaying processes until some future time, representing timing requirements such as specifying periods, and checking timing requirements by checking for missed deadlines. These timing facilities of ORTS will be explored in the following subsections.

3.1.3.1 Real-Time Clock

ORTS uses a global variable, CURRENT_TIME, which is incremented by an interrupt handler for defining time. The interrupt handler is invoked by the on-board clock hardware. The CURRENT_TIME variable shows the number of ticks since the system has been started. CURRENT_TIME is a multiple of the constant value TIME_QUANTUM which has a granularity equal to $1E^{-4}$ seconds. It is not possible to have smaller granularities with the current hardware due to the execution time overhead introduced by the ORTS scheduler. Faster clock rates overutilize the DSP processor and result in less computational time for the processes. Access to time is provided by the GetTime() function which returns the time in seconds since the start of the system.

3.1.3.2 Delaying Processes in ORTS

In addition to having access to a clock, two functions are defined for delaying a process either for a relative duration or until the end of the period of a periodic process.
**Relative Delay:** The relative delay function is used to suspend a process for a given duration rather than checking time using the `GetTime()` function and busy waiting. The relative delay function in ORTS is `Sleep()`. The delay period is relative to the time when the `Sleep()` function is called. The calculation of the wake up time of a process is shown in the following equation.

\[
WakeUpTime = CURRENT\_TIME + SleepInterval
\]

**Delay Until the End of Period:** This function is necessary for providing absolute delays. In periodic processes it is necessary that the process be scheduled after an exact amount of time since the previously scheduled time. ORTS uses the `WaitPeriodicWakeUp()` function to ensure that the process does not wake up before the required period has elapsed. The `WaitPeriodicWakeUp()` function is accurate only in the lower bound. In other words the process is not released before the desired time is reached and it is released sometime after that time. When the `WaitPeriodicWakeUp()` function is used the time that the process has to sleep is calculated before the process is moved to the sleeping processes list. Since the calculation of the next delay time is based on the first wake up time, the delay until function does not result in cumulative drift. As can be seen from equation below, the next wake up time for a process is calculated using the previous wake up time and the wake up period. The current time information is not used for the calculation of the next wake up time.

\[
WakeUpTime(n + 1) = WakeUpTime(n) + WakeUpPeriod
\]
Corrected Relative Delay: The use of Sleep() function in the periodic tasks results in timing drift equal to the SleepInterval. An extra variable LastWakeUpTime is added to the ORTS Kernel to keep the last schedule time of the periodic process. The calculation of WakeUpTime is updated for the delay until the end of period function as follows:

\[
WakeUpTime = LastWakeUpTime(n) + WakeUpPeriod
\]

\[
LastWakeUpTime(n + 1) = LastWakeUpTime(n) + WakeUpPeriod
\]

3.1.3.3 Designing Tasks with Different Timing Requirements in ORTS

The timing constraints in real-time systems can be generalized as periodic and aperiodic. The tasks assigned for these timing constraints are called periodic and aperiodic tasks respectively. Periodic tasks are invoked at periodic intervals and are used for sampling data or executing control loops that have to meet predefined deadlines. Aperiodic tasks are activated in response to aperiodically generated events. Aperiodic events are usually received as interrupts to the system. In ORTS, it is possible to define both periodic and aperiodic tasks using the timing facilities. A sample periodic process can be defined as follows in ORTS.
void PeriodicTask(void)
while(1)
{
    TaskAction();
    WaitPeriodicWakeUp();
}

Once the TaskAction() is finished, the WaitPeriodicWakeUp() function has to be called to calculate the next execution time for the task. The resolution of the intervals has a lower bound which is equal to the TIME_QUANTUM variable.

In ORTS, aperiodic tasks can be implemented as follows.

void AperiodicTask(void)
while(1)
{
    TaskAction();
    suspend();
}

void InterruptHandler(void)
{
    resume(AperiodicTask);
}

Aperiodic tasks are triggered by interrupts. The interrupt handler for the aperiodic event resumes the user assigned aperiodic task. Once the execution of the aperiodic task is finished, the task suspends itself. Any priority level other than hardware priority can be assigned to aperiodic tasks.

The relative delay function Sleep() can be used in both aperiodic and periodic tasks.
3.1.4 ORTS DSP Real-Time Operating System and Host System

Communication

The communication between the host system and the real-time kernel is established via the dual port memory which is on the DSP board. The dual port memory is also referred to as shared memory since it is shared between the two systems. These two terms will be used interchangeably in the rest of the thesis.

The shared memory is partitioned into buffers which are used as communication links between the host system and the real-time kernel. These links are also unidirectional similar to the interprocess links of the real-time kernel. Access to the buffers is synchronized. Therefore, if a task in the real-time kernel is required to write data to a full buffer, it is blocked until the corresponding task on the host system consumes data from the buffer and vice versa. The synchronization is provided using the head and tail pointers of buffers. These pointers are also kept in the shared memory as part of the links.

3.2 Discussion of ORTS from Real-Time Perspective

After thoroughly examining the real-time features of ORTS, the following suggestions are made for improving the real-time characteristics of the system.

- The current hardware that ORTS is implemented on does not allow the applications to run at faster clock rates than 1KHz. Otherwise the scheduler overutilizes the processor and there is no computational time is left for the tasks. The average sampling frequency of applications such as intelligent machining (sensor data acquisi-
tion, adaptive control, fast fourier transform, tool wear monitoring, tool breakage detection, chatter avoidance) is 100 $\mu s$. However the granularity of ORTS scheduler is 10 $\mu s$ which is not sufficient for such applications. A simple cyclic executive based scheduler which is introduced in the following section was implemented to overcome the stated drawbacks of ORTS.

- A real-time system is defined to be hard real-time if the value of computing a result is zero after the deadline is missed. In soft real-time, systems the value of computing a result decreases gradually after the deadline is missed. ORTS supports soft real-time applications on its current hardware platform. However, it is capable of supporting hard real-time applications on more powerful execution platforms. A new function that warns the user if a deadline is missed was added to ORTS as part of this research work. The application programmer can use the warning messages from this facility to modify the priority assignments or the code to optimize the computational time of the processes in order to ensure that the deadlines are met.

- ORTS uses priority inheritance for the synchronization protocol. Priority inheritance solves unbounded priority inversion; however, chained blocking is still an issue. Other protocols such as priority ceiling protocol, and priority ceiling emulation prevent chained blocking, mutual deadlocks, and unbounded blocking. ORTS should use one of these synchronization protocols to avoid mutual deadlocks, and chained blocking.

- ORTS has 8 priority levels. In real-time applications it is desirable to have a differ-
ent priority level for each task in order to increase the schedulability of a system.

Having tasks with equal priorities also causes priority inversion. The total number of priority levels in ORTS has to be large enough to accommodate the total number of tasks that are used in an application.

- Timing drift occurs if a task is not released at the beginning of its wake up time. In the case of a local drift the consecutive periods are not affected in a periodic task. However, cumulative drift results in time shifting of all the consecutive wake up times of a periodic task. The previous implementation of Sleep() function was causing cumulative drift in periodic applications which is a serious timing problem. This was solved by updating the Sleep() function to store the wake up time of the task in a separate variable for the calculation of the next wake up time.

- Preventing local drift is important in tasks such as sampling. ORTS prevents local drifts in tasks by moving the time-critical functionality to the device interrupt handlers. For example in the case of sampling the values of the encoder registers, or the A/D converter registers are stored in the interrupt handler when the conversion is finished, and the necessary calculations are carried out in the associated periodic task. Interrupts cause priority inversion because they run at hardware priority, higher than any software task priority. It is the application programmer's responsibility to keep the interrupt task small to minimize its effects on schedulability.

- ORTS uses a preemptive, prioritized, round robin scheduler. Round robin schedulers are suitable for interactive applications where the average response time is
low. Additionally, they are highly non-deterministic and preemption cannot be controlled. Therefore, round robin schedulers are not suitable for real-time systems. A preemptive, prioritized FIFO scheduler should be used instead.

3.3 Fast Cyclic Executive

ORTS can support hard real-time systems. However, due to the execution time overhead related to the kernel functions it cannot be used for tasks that have short deadline. It is suitable for control applications in the areas of robot control and low frequency machine tool monitoring which have a frequency range of 10 Hz to 100 Hz. For intelligent machining monitoring and control applications that are mentioned in the previous section, faster execution ranges of 10KHz - 100KHz are required. Additionally this second group of applications is composed of a set of predefined tasks that run sequentially and do not need synchronization among each other. Therefore, a faster, and simpler scheduler is required. For that purpose a cyclic executive based real-time system called the FCExec was implemented on the TI C31 33 MHz DSP processor based PC-C31 board and WinNT. The cyclic executive was also ported to TI C32 60 MHz DSP processor based INDY board.

3.3.1 Design Structure of FCExec

FCExec is composed of a real-time cyclic scheduler that runs on TI-C3X processor based DSP boards and a WinNT based host application, as shown in Figure 3.3. The host application uses WinNT threads and it is an interface between the user and the real-time scheduler.
The host system and the real-time scheduler communicate with each other through the dual port shared memory which is on the DSP board.

The dual port shared memory is partitioned into two main areas as shown in Figure 3.4. The first partition is used for sending commands such as start, stop, continue, and check period via the command location, sending user requested parameters via the cyclic period and number of channels locations, synchronizing the double buffers via buffer0 full flag and buffer1 full flag locations, and reporting deadline misses to the host system via period overrun flag location. The second partition is a double buffer which is used for sending the acquired data from the real-time system to the host system.

**Cyclic Scheduler:** The cyclic scheduler is basically an infinite loop that executes a list of tasks at a given frequency and order. The tasks run to completion at every cycle. The user defined frequency is provided to the real-time scheduler via the shared memory location which is named period. The list of tasks that are to be executed at every cycle are defined by the user. At every cycle, the scheduler also checks whether the period deadline of a task is missed or not. If the deadline is missed, it is reported to the host system via...
synchronization flag
command
period overrun flag
buffer0 full flag
buffer1 full flag
number of channels

Figure 3.4 FCExec shared memory distribution for DSP-PC communication

Data Logging From Real-Time Scheduler to the Host System: The partition of the shared memory which is used for logging data from real-time tasks to the host system is divided into two buffers. The two buffers are used for logging data with double buffer mechanism. After the first buffer is filled, data is not written to the second buffer until the second buffer is totally consumed by the host system. Similarly, the host system does not start reading data from a buffer unless it is totally full. This method decreases the time spent for synchronization of the host and the real-time cyclic scheduler.
These buffers are suitable for transferring large amounts of data from the real-time system to the host system. The real-time system does not need to synchronize with the host system unless both of the buffers are full. Similarly, the host system does not need to synchronize with the real-time system unless both of the buffers are empty. The synchronization is done through the buffer0 full flag and buffer1 full flag flags which are also defined on the shared memory.

**Real-Time Clock:** Similar to ORTS, FCExec uses a global variable `CURRENT_TIME` to keep the total time elapsed since the initialization of the system. The `CURRENT_TIME` variable is incremented each time the onboard timer ticks. The resolution of the timer is set to $1 \times 10^{-5}$s which is 10 times faster than the timer of ORTS. Therefore, FCExec has a 10 times finer timer granularity than ORTS.

**Sending Commands from a Host Application to the Real-Time Scheduler:** The shared memory location `command` is used for sending commands from the host system to the real-time scheduler. There is a command manager in the real-time system that processes the commands received from the host system. By sending commands to the real-time system, the host system can start and stop the cyclic scheduler, set the frequency for the cyclic scheduler, reset the data logging buffers, and synchronize itself with the real-time system after both have booted.
3.4 FCExec in Comparison with ORTS

Two sample applications will be presented in the following subsections for a better understanding of the usage of ORTS and FCExec. After the sample applications are shown, the advantages and disadvantages of the two systems will be compared. In the first example, an adaptive control application was implemented using the FCExec system and was run on a milling machine. In the second example, an axis control application was implemented using the ORTS system and was run on a x/y table.

3.4.1 Adaptive Control Application

Adaptive control is used to ensure that the peak resultant force during a milling operation is always equal to a constant reference force value to avoid tool breakage, chipping, etc. by manipulating the feed rate of the milling machine. The input to the system is the maximum allowable cutting force value. The cutting forces are sampled at a high frequency to correspond to 5 degrees or less rotation of the spindle by using sensors such as a dynamometer or strain gages and input to the computer control system via an analog to digital converter. After the measured forces are scaled and low pass filtered, the maximum resultant cutting force for each spindle revolution is calculated. The adaptive control algorithm computes the new feed rate value, which is scaled and sent to the milling machine, using digital to analog converters.

The implementation of adaptive control is illustrated in Figure 3.5. The dotted lines show the connection to the controlled system. The other blocks show the functions that are
called cyclically at every period. The blocks up to the adaptive control block are executed at a high frequency such as 2.5KHz when the spindle speed is 1500 rpm which corresponds to 3.6 degrees of spindle revolution per sample. The function blocks starting from adaptive control are executed at 12.5 Hz which is 200% of the spindle period.

This application is an example of a series of functions that have to be called one after the other without any synchronization among them. Therefore, a simple cyclic scheduler is sufficient.

3.4.2 Axis Control Application

The computer numerical controller units of CNC systems are used for decoding the internationally recognized standard NC programs into physical control, computation, and PLC units of the controlled system [5].

In the application shown in Figure 3.6, the CNC unit produces time stamped target position values from the given NC code blocks. The CNC unit is a stand-alone task that runs independent of the axis position control loop. The trajectory information produced by
the CNC task, and the current position information input received from the encoders are fed to the axis controller. To prevent sudden changes in the position which may result in jerks on the drives, the discrete position commands are resampled in order to smoothen the trajectory. The control law is applied and the new control signal is fed to the controlled system via a digital to analog converter.

This application has two tasks running simultaneously. The first task is the CNC executive which is not a periodic task. The second task is the axis control loop and is run at 1KHz. A multitasking RTOS such as ORTS is suitable for such applications.

3.4.3 Advantages and Disadvantages of ORTS and FCExec
As mentioned before, ORTS is a RTOS which provides the necessary environment for design and implementation of low frequency applications such as robot control, and low
frequency machine tool monitoring. However, for high frequency applications such as intelligent machining and data acquisition where tasks run sequentially and among which no synchronization is required, ORTS is rather slow and has more features than required. For that purpose, the simpler cyclic executive scheduling framework based FCExec system was implemented.

The real-time part of FCExec is a simple infinite loop where user defined tasks are executed in an order defined by the user. Therefore, there's no overhead due to task switching or task synchronization. ORTS involves a scheduler that requires task switching and task synchronization. This extra processing results in less computational time for the tasks.

FCExec is not as flexible as ORTS in terms of application design. The design of the main loop of the cyclic executive can be complex if the computational time of a task is more than the period of the main loop or if the period of a task is not harmonic with the period of the main loop. In both of these cases, FCExec is not an easily reconfigurable system. On the other hand, tasks can be released at any time in ORTS and can be executed at any frequency because it uses a preemptive, prioritized scheduler.

### 3.5 Conclusion

The ORTS OAC offers a development and prototyping environment for real-time control applications. The ORTS timing functions were improved for real-time compatibility as part of this research. On the TI C3X processor, ORTS is suitable for hard-real time applications.
that have low deadlines in the range of milliseconds. However, it is not possible to use it for faster response time applications with the current hardware configuration. It is also observed that applications such as intelligent machining which have short deadlines do not require complex real-time facilities. A simple cyclic executive based scheduler, FCExec was implemented to compensate for the drawbacks of ORTS. The FCExec system runs on two different DSP Board configurations, namely PCC31 and INDY. It is currently used for a research project which includes the application of sliding mode control to the piezo actuator for turning operations in the MAL at UBC.
Chapter 4

Open Architecture Data Interfaces for ORTS

In contrast to the number of projects for standardizing OACs there has been no research carried out for interfacing these OACs to each other. Another goal of this research has been to develop and implement a gateway between OSACA and ORTS. The first gateway implemented between OSACA and ORTS is called ORTS OSACA Interface (OOI). OOI enables OSACA applications to be started and run from an ORTS host application environment. Therefore ORTS users can benefit from OSACA applications without learning the OSACA API. Despite the fact that OSACA users have to learn about the ORTS structure, it can be also used as an access to ORTS applications for OSACA users.

The second gateway, Process Control and Monitoring Interface (PCM), was implemented for communication between the ISG NC Kernel and ORTS. This enables process control and monitoring applications which are implemented with ORTS API, to be used for enhancing CNC systems. Since the OSACA API was already implemented on the ISG NC Kernel, OOI might have been an alternative approach for ORTS and the ISG NC Kernel to exchange data. However, due to the unpredictable timing characteristics of OSACA which are discussed in Chapter 3, and the non real-time data communication between the host system and the real-time kernel of ORTS, the implemented interface was non-deterministic
and non-real-time. In contrast, the PCM is lightweight and permits relatively real-time properties compared to the OOI.

In the following section, the data interfaces of implemented OACs are compared. The next section illustrates the way a simple control application would be implemented using ORTS and OSACA. This example is used to show the differences and similarities between the two systems which were considered for preparing the OOI. After the OOI is defined, the way the example control application would be implemented using the interface is shown. Next, the PCM is presented. The chapter concludes with an adaptive control experiment which utilizes the PCM to show the applicability of the gateway.

4.1 The Data Interfaces of Other Open Architecture Controllers

ORTS is an open architecture RTOS that runs on DSP boards and uses WinNT as the host system. The data interfaces between ORTS applications [2, 3, 4, 5] are links which provide unidirectional data transfer. The user can define links between any ORTS application, either on the host system, or on the real-time system.

OSACA [6, 7, 8, 9, 10, 11, 12] aims to define a system architecture for open control systems which is independent of any specific manufacturer. The data interface of OSACA, which is referred to as the communication system, is based on the ISO/OSI Reference Model [22]. The data to be shared by an OSACA application are interfaced to other OSACA applications as objects. The functionality of OSACA control applications are grouped into a predefined set of modules (i.e. numerical controls, robot controls, programmable logic
controls etc.) according to their main functionality in the control. A standard API is defined to support portability and extendibility of OSACA applications. The API functions for exchanging data are generic and application independent, therefore the system functionality can be easily extended.

OSEC [15, 16, 17] is the Japanese initiative for standardizing open architecture control systems. A reference model which is composed of layers, each representing a different functional block of manufacturing systems (i.e. motion control function block, servo control function block, PLC function block, human machine interface function block, etc.), is defined. The API's of each layer is defined as part of the reference model. However there is no restriction on the underlying data exchange structure between layers. The interface functions defined for the layers are the methods of exchanging data between the layers. A different set of functions is defined for each layer. Therefore addition of a new layer would result in another set of functions for exchanging data.

The TEAM API requirements were derived from the Open Modular Architecture Controller (OMAC) [13, 14] requirements. A client-server based communication mechanism is used between the modules. Proxy agents are used for communication among distributed applications.

The HP [18] OAC software is composed of modules which include an API and a message based communication system. The API defines a message passing protocol among the modules and also specifies the message format within the protocol.
4.2 OOI - ORTS OSACA Data Interface

The OOI was designed and implemented as a data interface gateway between OSACA [7] and ORTS [2] as shown in Figure 4.1. OOI enables OSACA applications to be started and run from ORTS host application environment using ORTS scripts. The interface was designed by comparing the API's of the two systems.

4.2.1 Comparison of OSACA and ORTS Application Structures

The motion control example shown in Figure 4.2 is used to highlight the similarities and differences of application structures of OSACA and ORTS. Every control block represents an integral functional unit within the overall control system. Control blocks accept inputs from the previous block and produce output for the following block. The dotted lines represent input and output to and from the controlled system. The position information is received by the analog to digital converter which is shown by the A/D conversion block. The control algorithm is applied to the digital position information. The digital control sig-
Figure 4.2  Example control application for comparing OSACA and ORTS

The information obtained from the controlled system is sent to the digital to analog converter which is shown by the D/A conversion block.

For analysis and verification purposes, the current position and the next position values, the output signals obtained from the A/D converter, and the PID control are logged by the data acquisition system. The applications are explained using Unified Modeling Language (UML) collaboration diagrams [27]. Collaboration diagrams describe interactions among objects. Objects are shown with rectangles. A message from one object to another is shown with a straight line, text and an arrow next to it. Arrows represent the message flow direction. The numbers next to the arrow indicate the nested procedural calling sequence of the messages. For example within nesting level 1, message 1.1 precedes message 1.2. Numbers at the same level of nesting level represent threads that execute concurrently. For example messages 2.1.1 and 2.1.2 are concurrent.
4.2.1.1 General Structure of ORTS Applications

Integral parts of control applications such as position controllers, data acquisition units, or signal processing units are implemented as processes in ORTS. Processes can be executed either on the PC or on the DSP. Real-time processes are executed on the DSP where the real-time kernel of the ORTS system is placed. Process execution on the ORTS PC side is non-deterministic. Therefore, only non-time-critical tasks which provide the user interface, and enable the user to log data, or enter input are run on the PC side. The processes exchange data with each other using unidirectional links. Links can be established between any two ORTS processes, either on the PC or the DSP. The access to links is through LinkWrite and LinkRead system calls. The example control system shown in Figure 4.2 can be implemented in ORTS as shown in Figure 4.3.
The real-time parts of the control application are implemented as DSP processes named *DSPProcess_ADC*, *DSPProcess_PID*, *DSPProcess_DAC*. These real-time processes correspond to the A/D conversion, PID control, and D/A conversion units of the closed loop control sequence of the example control application shown in Figure 4.2. The non-critical data logging parts are implemented as PC processes named *StoreAdcOutput*, and *StorePidOutput*. The data acquired by the *DSPProcess_ADC* is transferred to *DSPProcess_PID* via *Link_Adc2Pid* for processing. It is also sent to the PC to be logged on the hard drive via *Link_Adc2Pc*. Similarly the data processed and calculated by the *DSPProcess_PID* is transferred to the *DSPProcess_DAC*, and to the PC via the *Link_Pid2Adc*, and *Link_Pid2Pc*, respectively. The PC processes *StoreAdcOutput* and *StorePidOutput* correspond to the data acquisition system of the same example control application.

### 4.2.1.2 General Structure of OSACA Applications

Integral parts of the numerical controls for machine tools such as man machine control, motion control, logic control, axis control, or process control are implemented as architecture objects in OSACA. Architecture objects are realized on a client-server basis. An architecture object can be a client, a server, or both.

All information that will be accessible among OSACA applications is mapped to a communication object (CO). COs provide two basic types of interfaces: data interface, and process interface. The data interface of architecture objects is represented by variable communication objects (COVar). Each internal variable (i.e. data physically allocated in
the application) which should be accessible by other architecture objects has to be mapped to an instance of the variable CO class.

The functional purpose of the client-server structure is characterized by the relationship of the data producer and the data consumer. An architecture object is called a producer if it generates data (e.g. as a result of internal calculation). A consumer architecture object relies on externally generated data as the basic input for its own functionality (i.e. to perform calculations or display data). The producer architecture object acts as a server and offers its internally generated data to other client architecture objects which act as consumers.

Normally, server applications in OSACA use a state machine to control the program flow. Access to the state machine is realized through process objects. The states can be changed either by the server itself or by the client using the corresponding client process object.

The sample control application could be implemented as one single architecture object or every control function can be implemented as a separate architecture object as shown in Figure 4.4. The latter approach provides flexibility and can be easily reconfigured. For example, the PID controller can be replaced with pole placement controller without altering any other architecture objects.

The control units of the application are defined as three separate architecture objects which are named AO_ADC, AO_PID, AO_DAC. These three objects correspond to the A/D conversion, PID control, and D/A conversion units of the example control appli-
Figure 4.4 Sample implementation of the control application using OSACA.

As stated before, server architecture objects in OSACA have a state machine interface for controlling the sequence of actions within the object. The state machine interface is provided by the process objects. The client architecture object can control the defined
2. change state

2.1.1.1. get variable name

AO Client CceVariable
: CceVariable

AO Server ScoVariable
: ScoVariable

2.1. change state

AO Client CceProcess
: CceProcess

AO Client ScoProcess
: ScoProcess

AO Server BAOC

AO Server BAOC

Figure 4.5 OSACA Application Data Exchange Detail

state machine using the corresponding client process object. If the client architecture object is required to update the value of a variable object, first it updates the corresponding client variable object. Then, it changes the state of the server architecture object via the corresponding client process object causing the server to calculate the new value of the variable object and update the server variable object accordingly.

4.2.1.3 Similarities and Differences between OSACA and ORTS

Applications

The implementation of a sample control application using OSACA and ORTS was described in the previous subsections. Here, a comparison is given to help to achieve an understanding of the APIs of ORTS and OSACA. This will provide insight into the design and implementation of the OOI. The similarities and differences of the two systems can be summarized as follows:
The primary part of an application is referred to as a process in ORTS and as an architecture object in OSACA. Communication between these parts is realized by links in ORTS, and by server and client variable CO pairs in OSACA.

As for differences, the variable object pairs which are the communications means in OSACA are bidirectional. In other words, both the client and the server can read and write the value of these objects. However ORTS links are unidirectional. An ORTS process can either read from or write to a link. Therefore, bidirectional communication requires two links to be defined.

In addition, every server architecture object in OSACA is designed as a state machine, and this state machine is controlled by a client using state transition functions. On the other hand, ORTS processes are independent from the control of a client and each one is associated with a separate process of its own.

4.2.2 Details of ORTS OSACA Interface

In order to access OSACA applications from ORTS an OSACA ORTS data exchange interface was designed and implemented. New script commands were defined for the ORTS API to utilize the OOI functionality. Using the new ORTS script commands, communication between OSACA applications and ORTS applications can be established during run-time and OSACA applications can be run from an ORTS environment.

OSACA variable objects constitute the data interface of the OSACA communication system. Due to the bidirectional data exchange structure of variable object pairs in
OSACA and unidirectional links in ORTS, a new variable object class which allows data transfer in only one direction was defined in the OOI. The two new classes, \texttt{OOIScoVariableWriteOnly} and \texttt{OOIScoVariableReadOnly} are derived from the server variable object class of OSACA. \texttt{Set}, and \texttt{get} services of variable objects are redefined within this class. The OSACA communication variable objects that have to be accessed from ORTS must be instances of these new classes. These objects, when instantiated, create a corresponding OSACA variable object and ORTS communication link. \texttt{OOIScoVariableWriteOnly} is used for accessing the data calculated by ORTS processes, and \texttt{OOIScoVariableReadOnly} is used for sending data to ORTS processes for further calculation.

The \texttt{set} service of the \texttt{OOIScoVariableWriteOnly} object reads the new value of the variable object from the corresponding ORTS link and updates the corresponding OSACA variable object. The \texttt{get} service of the \texttt{OOIScoVariableWriteOnly} object returns an error message since the variable object is write only. \texttt{OOIScoVariableReadOnly} is the dual of the \texttt{OOIScoVariableWriteOnly}. Similarly, the \texttt{get} service of the \texttt{OOIScoVariableReadOnly} gets the current value of the corresponding OSACA variable object and sends it to the related ORTS process via the corresponding ORTS link.

Using the OOI, the example application described in the previous sections can be implemented as shown in Figure 4.6. In this example, \texttt{DSPProcess_ADC}, \texttt{DSPProcess_PID}, \texttt{DSPProcess_DAC}, which constitute the real-time units of the system, are implemented as ORTS processes. Non time critical units, which are namely \texttt{AO_StoreAdc} and \texttt{AO_StorePid} are implemented as OSACA architecture objects. The communication
between the ORTS processes are established via the Link_Adc2Pid and Link_Pid2Dac, similar to the ORTS example. The variable object AdcResult, which is an instance of the OOI variable object class is used to transfer the data acquired by the DSPProcess_ADC to the AO_StoreAdc. Similarly, the variable object PidResult is used to transfer the data calculated by the DSPProcess_Pid to the AO_StorePid.

The details of the new OOI variable object class can be seen in Figure 4.7. When the OSACA architecture object requires the contents of the variable object to be updated, the set service of the variable object is called. The set service reads the data from the corresponding ORTS link and updates the contents of the corresponding OSACA variable object.
A new script command \textit{OSACA} is added to the ORTS scripting language to simplify the usage of the new interface. Using this new command, ORTS users can run an architecture object and define the variable objects that are to communicate with the ORTS applications from within the ORTS environment. The architecture objects that are going to be run using the script language have to be registered in the architecture object list, which is defined in ORTS.

\subsection*{Advantages and Disadvantages of OOI}

The proposed and implemented OSACA ORTS interface is a simple solution. It can be used by ORTS users without detailed knowledge of OSACA by simple script commands. ORTS applications can exchange data with OSACA applications without any change to the ORTS applications. OSACA applications can exchange data with ORTS applications after simple modifications.
The main disadvantage of OOI is that OSACA application programmers have to use the new CO variable class if the variable object has to be accessed from ORTS. The OSACA architecture object that is going to be used by the OOI has to be compiled with the ORTS host software, otherwise it cannot be accessed from the host system.

4.3 PCM - Process Control and Monitoring Interface

An OSACA communication stack has been already defined for the ISG NC Kernel by ISG of Germany. Therefore, ORTS applications and ISG NC Kernel applications can exchange data using the OOI. However, OSACA is not suitable for real-time systems as discussed in Chapter 2. Additionally, the communication between the Windows based host system and the real-time kernel of ORTS is not real-time due to the non real-time structure of Windows applications. These observations motivated the development of a shared memory based data interface between the ISG NC Kernel and ORTS. The interface, called the Process Control and Monitoring Interface (PCM), enables ISG NC Kernel to utilize ORTS process control applications. The reference object architecture defined within the OSACA standard supports easily portable and interchangeable manufacturing system applications. The PCM uses similar reference object architecture as OSACA so that applications developed for the interface can be easily ported to the OSACA environment.

The hardware and software structure of PCM is shown in Figure 4.8. ORTS runs on a TI based DSP board which is plugged into a Pentium based PC. The PCM defines a data exchange mechanism, which uses the dual port shared memory present on the DSP board.
The ISG NC Kernel runs on the VxWorks RTOS. VxWorks and WinNT run on the same processor and use the PC memory for context saving, and exchanging data with each other. Hardware that switches between VxWorks and WinNT on the same processor via global PC interrupts is plugged into the PC.

The PCM uses buffers defined on the DSP board shared memory for exchanging data. Both ISG NC Kernel and ORTS act as client and server. A client and server pair is defined for each system to send requests and process requests, respectively. The requests are sent and received via the serial buffers. The second type of buffers, parallel buffers, are used for exchanging data between the applications of both systems. New script commands were added to the ORTS API to provide access to the functionality defined by the PCM.
The details of the buffers and the associated request processing protocol are explained in the following sections.

### 4.3.1 Serial Buffers

The two serial buffers are designed as bidirectional data exchange structures. There is a client task and a server task associated with each serial buffer. One direction of the bidirectional serial buffers is used for sending requests by the client task and the other direction is used for receiving replies coming from the server task of the other system. Each system uses the client task to send command requests to the other system. The server associated with the same buffer receives a command request, processes it and returns the reply using the same buffer. A client-server command-reply protocol was implemented and it is shown in Figure 4.9 on a UML sequence diagram.
The message, which is actually a command or a reply consists of the command/reply code and the command/reply parameter. A command exchange protocol was implemented for synchronizing the two systems and creating parallel buffers during runtime. Also, a command protocol for manipulating tasks, starting, stopping, and resetting the systems was implemented. The command exchange protocol for creating parallel buffers is shown in Figure 4.10 as an example. To create parallel buffers using the defined protocol, a list of the required parallel buffers has to be provided to the two systems. First the client requests that the server creates a parallel buffer using the \texttt{CT\_CreateParallelBuffer} command. Then the server sends the successful accomplishment message, \texttt{RT\_CreateParallelBuffer}. This is repeated for each entry in the parallel buffer list which is known a priori by both of the systems. After all the buffers in the list are created, the client sends the \texttt{CT\_ParallelBuffersCreated} command. Then the server confirms it with the corresponding reply, \texttt{RT\_ParallelBuffersCreated}.

### 4.3.2 Parallel Buffers

The parallel buffers, which provide unidirectional and synchronous data exchange, are used for interprocess communication between the two systems. They allow data of arbitrary size and structure to be transferred one at a time.

Parallel buffers are created in the shared memory during run-time. The parallel buffers that are required for the application are provided as a list by each system. ORTS requests that the ISG NC Kernel creates the parallel buffers that it needs. Similarly the ISG NC Kernel requests that ORTS creates the parallel buffers that it needs.
The parallel buffers can be configured easily using the parallel buffer lists for every application. The users can access the parallel buffers from their applications using the generic parallel buffer read and parallel buffer write operations.

4.4 Sample Application for Testing the PCM

The adaptive control application [24], which was introduced in Chapter 3 was implemented using ORTS and tested by the author at the Ex-Cell-O High Speed Machining Center in Germany. The system was capable of keeping the peak cutting forces at a desired level. However, when PCM was implemented, there was no machine tool working under ISG.
Kernel at the time. Therefore, to test the PCM, the simulation of a machining process which was implemented in MAL at UBC was used, and the cutting forces were predicted as a function of the feed rate. A time varying discrete transfer function of the process was simulated at each adaptive control interval. The real-time ISG interface with ORTS based adaptive control was tested in this simulation mode. The test showed that PCM demonstrates real-time capabilities for data exchange between the two systems. The real-time machining has the identical software structure, except that the simulated cutting forces are replaced by the measured forces at the same frequency used in the simulations. The results and setups of both tests are explained in the following subsections.

4.4.1 Adaptive Control in Real-Time

The adaptive control program was run on an industrial milling machine to improve the machining process. The feed rate of the machine was set to 1680 mm/min. The adaptive control was set to keep the upper limit of the resultant forces to 350N. It can be seen in Figure 4.11 that as a result of the adaptive control the force is kept constant around 350N. The second plot shows the changes in the feed rate according to the applied feed rate override percentage which can be seen on the third plot. Adaptive control calculates the feed rate override value in order to keep the resultant peak force applied to the cutter at the 350N limit which is specified by the user.
Figure 4.11 Test results of adaptive control during a real-time experiment

4.4.2 Adaptive Control During Simulation

The adaptive control task was coupled with a cutting process simulation task to simulate the forces received from the sensors during machining. Both of these processes run on ORTS. The feed rate override is calculated as a percentage and sent to the NC Kernel using serial buffers. The NC Kernel reads the feed rate override percentage and updates the feed rate accordingly. Simultaneously, the NC Kernel is running an NC code which is machining a half cylinder. The feed rate value is updated according to the received feed rate override.
percentage. Both the feed rate and the feed rate override percentage are logged by the NC Kernel. The feed rate value defined by the NC Code is 3000 mm/min. In Figure 4.12 the first graph illustrates the feed rate override percentage varying roughly from 50% to 100%. It can be seen from the second graph that the feed rate is varying according to the percentage override value.
4.5 Conclusion

Two data interfaces, which comprise the fundamental results of this thesis work have been designed for ORTS. The first data interface, OOI, enables communication between OS-ACA and ORTS applications via a TCP/IP protocol based communication stack. New script commands were added to the ORTS user interface so that ORTS users can run OS-ACA applications from an ORTS environment and define the data that will be exchanged between the two systems. Also, the interface enables ORTS applications to access OS-ACA applications running on different environments such as VxWorks, VxWin, SunOS, SOLARIS 2.4, and OS/2 via the PC side of ORTS.

The second data interface, PCM, enables the ISG NC Kernel to exchange data with the ORTS RTOS over a shared memory based communication structure. This interface can be used to change machining process control parameters of the ISG NC Kernel to improve productivity, and the quality of machining process. PCM is light weight and deterministic compared to the OOI. PCM uses similar conventions to OSACA for defining the data to be exchanged. Therefore, the applications prepared for the PCM can be ported to OSACA environment with only minor changes.

In Figure 4.13, the abstract architecture of the designed interfaces is shown with the platforms that are being used. ORTS runs on a TI based DSP board which is plugged into a Pentium based PC. ISG NC Kernel runs on VxWorks. VxWorks and WinNT run on the same PC processor. Global interrupt hardware is used for switching between VxWorks and WinNT on the same processor. The data exchange platforms are summarized as follows:
Figure 4.13 System Design

- OOI uses the TCP/IP based OSACA communication stack for communication between OSACA and ORTS host applications.
- ORTS host system and ORTS real-time kernel communicate via the shared memory on the DSP board.
- VxWorks uses the PC memory to exchange information with WinNT applications.
- ISG NC Kernel and ORTS exchange data using the PCM via the shared memory on the DSP board.

ISG NC Kernel and ORTS applications can exchange information either via the OOI or the PCM. However, this feature has not been tested yet.
The proposed interfaces are the results investigating the exchange of data between existing OACs. Having such interfaces enables users of different OACs to utilize the applications developed on other systems without learning their API. For example, in case of the PCM, ISG NC Kernel users can access ORTS machining process control applications to improve productivity and quality.
Chapter 5

Conclusions and Future Work

This chapter summarizes the results of this research work. It also provides a list of suggestions for the continuation of the research.

5.1 Summary

There is a trend towards OACs in the manufacturing industries to ensure portability, extendibility, interoperability, scaleability, interchangeability of applications. The first part of this research is the analysis of two of the OACs, namely OSACA and ORTS, from a real-time perspective by comparing their API specifications with real-time systems.

OSACA is the first OAC evaluated in this research. A list of suggestions have been made for OSACA which can be taken as a guideline for the preparation of a real-time scaled down version of OSACA. It can be concluded that OSACA interface simplifies the implementation of manufacturing applications by its open architecture structure and well-documented and defined API. However, the OSACA platform specifications do not include sufficient real-time constraints and does not provide real-time support unless the underlying operating and transport systems are real-time compliant. Therefore, to use OSACA with the current API specifications for real-time applications, it has to run on a real-time environment such as VxWorks.
The second OAC evaluated in this research is ORTS. ORTS is suitable for hard real-time applications with low deadlines in the range of milliseconds. A new warning system was integrated with ORTS so that the application programmer is informed of deadline misses. This warning message can be used as a guideline for testing the schedulability of applications. The cumulative drift introduced to the periodic tasks in ORTS when the user calls the sleep() system call was eliminated with a new implementation of the system call.

The evaluation of ORTS also showed that ORTS has unnecessary overhead for control applications such as intelligent machining and monitoring which do not require sophisticated synchronization or shared memory access functionality. For this reason FCExec, which is a cyclic executive based real-time system, was implemented for a TI C31 33MHz DSP processor based PC-C31 DSP board. FCExec is suitable for high frequency applications where tasks run sequentially and among which no synchronization is required. FCExec enables control execution frequencies up to 100 KHz. In ORTS the control execution frequencies are limited to 1 KHz. FCExec was also ported to another TI C32 60MHz DSP processor based DSP board, namely the INDY. A warning system for alerting the application programmer about deadline misses was also included in this system. FCExec is used as a development and test environment for controlling turning machines in MAL at UBC.

Another achievement of this research is bridging ORTS and OSACA to exchange data between applications developed for the two systems. Two bridges, called OOI and PCM, were implemented. The first bridge OOI, which bridges between OSACA and ORTS,
exploits the open architecture advantages of OSACA, and therefore enables ORTS applications to exchange data with OSACA applications running on any platform. It allows access from ORTS to OSACA without any modification in the ORTS applications and with simple modifications in the OSACA applications. Simple commands were added to the ORTS scripting language to execute OSACA applications from an ORTS environment and to define the data to be exchanged between the two systems.

As a result of the non real-time characteristics of OSACA and non-deterministic data transfer between the host system and real-time kernel of ORTS, the data interface does not guarantee real-time requirements. As a consequence, the second data bridge PCM, is implemented between ISG NC Kernel and ORTS. PCM is light weight and has relatively better real-time characteristics compared to the OOI. In contrast to OOI, PCM is less open because of its hardware dependent structure. The data interface is based on a client-server communication model. A data exchange protocol for client-server communication was also defined as part of the research. New script commands for utilizing the implemented interface was prepared for ORTS. An adaptive control simulation was run on the implemented data interface to show the applicability of the system.

5.2 Future Work

Although the work presented in this research enhances the capabilities of the real-time OAC’s, the following tasks are suggested for further enhancements.

• Suggestions for a real-time OSACA can be used as a guideline and a scaled down
real-time version of OSACA can be prepared. Currently ISG NC Kernel and ORTS applications exchange data using the PCM. If real-time OSACA is implemented ISG NC Kernel applications and ORTS applications can exchange data using the OOI in real-time.

- OOI can be modified so that OSACA applications can directly run on ORTS without any modifications. Similarly ORTS applications can run on OSACA environment directly.

- API of FCExec can be enhanced in order to enable ORTS applications to be used in FCExec directly. It would also simplify application development for FCExec.

- FCExec system can be integrated with ORTS system so that same API and GUI as ORTS can be used for developing and testing applications.

- Currently ISG NC Kernel and ORTS applications can exchange data in real-time using the PCM. If real-time OSACA is implemented the two systems can exchange data via the OOI in real-time.
References


"OSEC Project Description Paper", Global Activity on Open Architecture Control Sys-

Position Paper OSEC: Open System Environment for Controllers —OSEC Architecture
and OSEC-API—”, Global Activity on Open Architecture Control Systems, ITIA Series,


[20] “Documentation BF KOMMU (communication)”, Software Version 2.4.4, ISG Internal

Arbeit, Universitaet Stuttgart Institut fuer Steuerungstechnik der Werkzeugmaschinen
und Industrieroboter, Univeristy of British Columbia Manufacturing Automation Labora-

[22] ISO 7498: “Information Processing Systems - Open Systems Interconnection-Basic Refer-


[26] “Rate Monotonic Scheduling Theory: Practical Applications”, Software Engineering Insti-
tute, Carnegie Mellon University, Notes, 1990.
