IST-2001-37652

Hard Real-time CORBA

Title

D3.4 Robot Control Testbed

Soft real-time implementation

Authors

Klas Nilsson (klas@cs.lth.se)
Mathias Haage (mathias@cs.lth.se)

Reference	IST37652/075 Deliverable 3.4
Date	2003-10-23
Release	1.0
Status	Final
Clearance	Consortium

Partners

Universidad Politécnica de Madrid
Lunds Tekniska Högskola
Technische Universität Wien
SCILabs Ingenieros

Summary Sheet

IST Project 2001-37652
HRTC
Hard Real-time CORBA

D 3.4 Robot Control Testbed

Soft real-time implementation

Abstract:

Based on the design of the Robot Control Testbed (RCT) for Hard Real-Time CORBA (HRT CORBA), the soft/non RT part of the testbed has been implemented. The primary test case is to use stereo vision to control a robot motions to catch a thrown ball.

One part of the implementation is the virtual testbed, which uses OpenGL-based rendering with synchronized updates of the virtual world to obtain virtual camera images that are consistent with the manipulator motions. Graphics and dynamics was implemented in C/C++, whereas the simplified control code was written in Java to support simple prototyping of the IDLs.

The physical implementation used the same IDLs but object/servant implementations were made in C for the actual hardware, but still on the TCP/IP (IIOP) level. The main purpose is to prepare for the hard real-time RCT, which is required to actually run the physical servo control of the robot joints. Thus RT-CORBA is not sufficient for controlling industrial robot.

Copyright

This is an unpublished document produced by the HRTC Consortium. The copyright of this work rests in the companies and bodies listed below. All rights reserved. The information contained herein is the property of the identified companies and bodies, and is supplied without liability for errors or omissions. No part may be reproduced, used or transmitted to third parties in any form or by any means except as authorised by contract or other written permission. The copyright and the foregoing restriction on reproduction, use and transmission extend to all media in which this information may be embodied.

HRTC Partners:

Universidad Politécnica de Madrid
Lunds Tekniska Högskola
Technische Universität Wien
SCILabs Ingenieros.

Release Sheet (1)

Release:	**0.1 Draft**
Date:	2003-09-27
Scope	Initial version
Sheets	All

Release:	**1.0 Final**
Date:	2003-10-23
Scope	Final version
Sheets	All

Table of Contents

1 Introduction

The Robot Control Testbed (RCT) implementation was planned to be carried out in two stages: one early stage for the non- or soft real-time support, and a later stage for the hard real-time part. In practice as found during the specifications, and as reported in the quarterly reports and in the design and specification documents, the open issues for the hard real-time support needed to be clarified prior to procurement and soft real-time implementation. That in turn implied that the implementations had to start earlier (to explore potential difficulties), and the soft real-time part had to be made with the hard real-time demands on the system in mind (not to end up with two completely different systems). Therefore, this document describes the testbed hardware and structure from both soft and hard real-time points of view, while the description of the implementation for the hard real-time documents the further development of the software and communication to meet the requirements of the hard real-time CORBA experiments.

Two robot systems were available for the RCT implementation:

1. The ABB Irb-2000 robot with external VME-based control computers as developed @control.LTH.se replacing the original ABB S3 control computers.
2. The ABB Irb-2400 robot with internal PCI-based control computers, with additional PCI/PMC-based control computers extending (but not replacing) the original system.

The aim and plan was to make use of the item 1 system, since that system provides access to the innermost control (communication) loop of the motion control. There are, however, arguments for implementing the RCT based on the item 2 system:

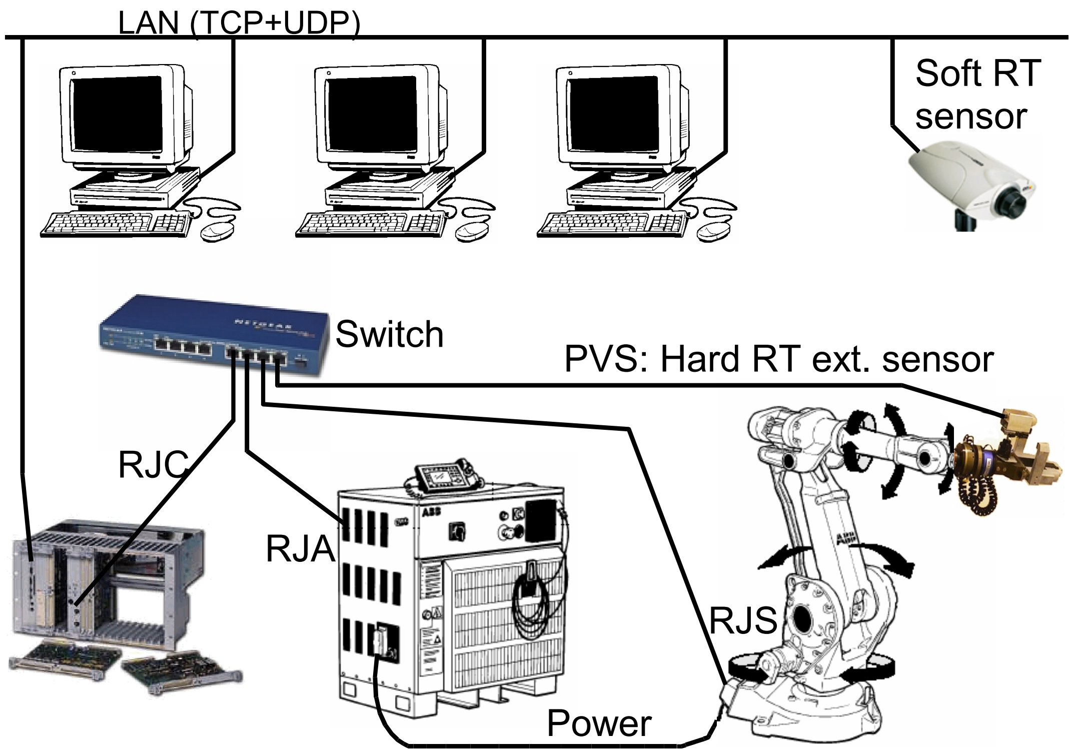
- The hardware in system 2 is entirely based on commercially available components, so there are external support available for fault analysis and repair. In particular, this includes the measurement system of the robot.
- The added control computer for external sensor based control is a (as described in the design and in the procurement) Motorola PrPMC800 PPC PMC board, which is a quite modern and PCI-based computer well suited for running (PowerPC) Linux, as needed for the ORBs. On the VME boards, on the other hand, there are on-board PCI logic connected to the back-plane VME bus, and it was not clear what efforts that was needed to get Linux installable and runnable on this hardware.

With these items in mind for the purpose of reducing the risk of an RCT implementation failure, and while waiting for the other parts of the project to approach testing, the implementation was carried out for both systems mentioned. We will refer to the first system by calling it VME-based, while the second will be called the PCI-based controller.

During final implementation it became clear that an RCT based on item 1 above was possible, and best for demanding test cases, so that is the system covered in the hard real-time implementation. In the following, both systems are covered since even if there is no full RCT implementation based on the Irb-2400 and its brand new ABB controller, the implementation done is important to mention for possible future developments; with less than 4 months of (estimated) work a HRTC a hard real-time CORBA robot control kit could be accomplished for use in practically any lab with a new ABB robot. Actually, a configuration for the next generation of ABB controllers is also possible. Therefore that item 2 platform is documented here, even if the possibilities are outside the scope of this project (recall that the motivation for the work here was risk management).

2 Implementation outline according to the generic design

As described in the RCT design, the system consists of a number of units



connected via switched fast Ethernet, as depicted in the following figure.

A minimum implementation is to have an ORB and CORBA objects encapsulating the RJC, and thereby also the total built-in control. The aim of providing HRT-CORBA enabled units for external sensing, in this case the external cameras, was found to be possible (but not within the timeframe of the project): The next generation of cameras from Axis Communication (www.axis.com) includes the same type of ETRAX processors and network interface as our VME-based RCT included for joint sensing and actuation.[1] For the current RCT, however, Sony FireWire cameras are used since the delivery of the new cameras from Axis was delayed. Therefore, hard real-time communication is only used for control of the robot joints, and currently only for the VME-based system. Due to the fact that the drivers for the Sony/UniBrain FireWire interface is only available for Windows, the RCT with visual feedback includes at least one Windows computer, of course not for the hard real-time part. On the other hand, the presence of a heterogeneous system with different operating systems (Linux/Solaris/Windows) very well illustrates the benefit of the CORBA platform independence.

3 The PCI-based RCT implementation

The basic real-time control functionality of the PCI-based Irb-2400 system was demonstrated to the reviewers during the final review visit in Lund. The robot performed compliant force control by reading an external force/torque sensor via an interface board on the PCI bus, and by adjusting the position control references of the motion controller via shared memory. The control cabinet is depicted in Figure 1.

[1] In other words, an extra outcome of this project is that it will be possible to run the HRTC communication and ORB-techniques in future low-cost cameras. We intend to make use of this in future robot projects, for low-cost 3D vision.

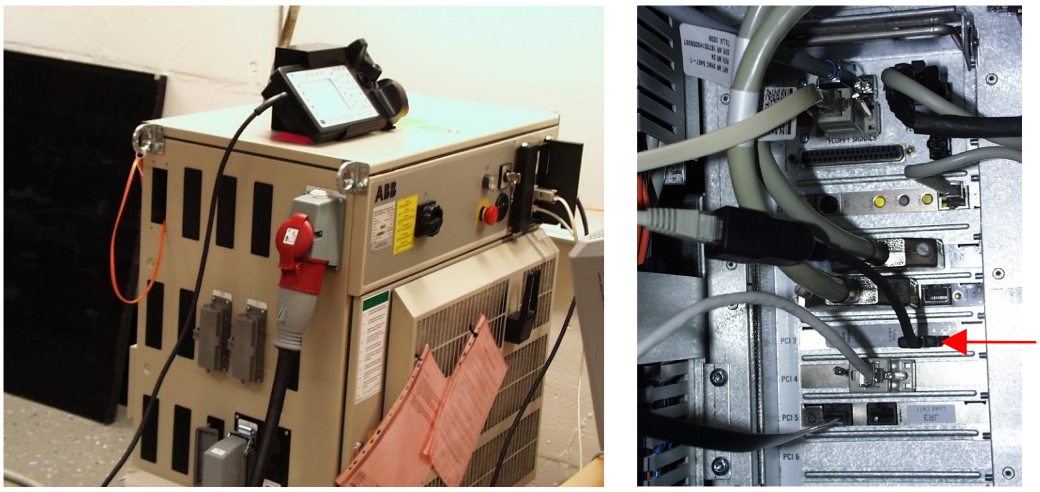


Figure 1 The robot control cabinet (left) that internally (right) is extended by the PPC PrPMC800 in the PCI3 slot where the arrow points at the 100Mbit/s Ethernet that provides the new network interface added to the original system. The force/torque sensor is connected to the lower PCI5 slot, which is used by the PPC processor to accomplish force control.

The organization of the hardware on the PCI bus is shown in Figure 2. The difficulties encountered during the implementation included changes of the Linux kernel for installation and execution on the PPC board. Another difficulty was the very complex booting and shared memory allocation, since we could not change the BIOS of the ABB Main computer running VxWorks and the reset of the PPC processor prevented it from allocating its PCI resources in time. The solution is based on remapping the shared memory to an area that the PCI/PMC carrier board (see Figure 2) gets allocated during system boot.

© HRTC Consortium / Clearance: Consortium

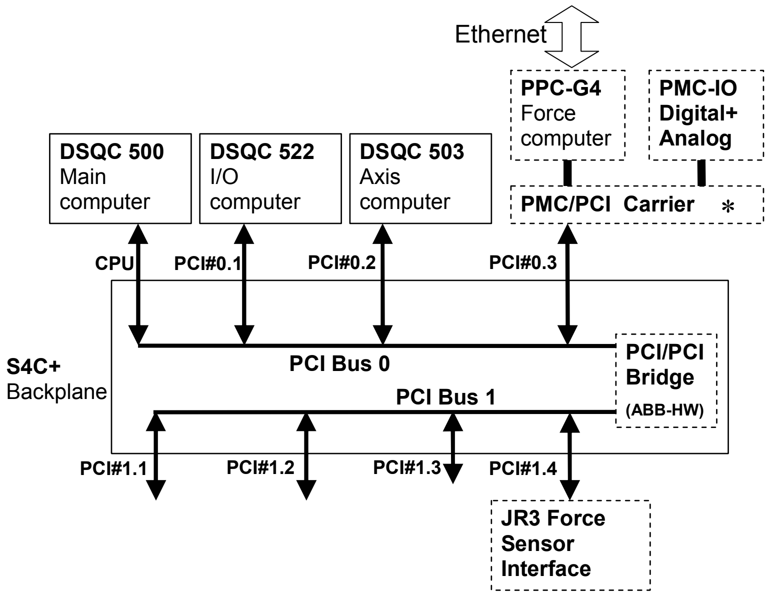


Figure 2 The PCI bus and connected boards. There are Ethernet connections (not shown) also to the Main computer and to the IO computer but with standard TCP/IP as defined by ABB, while the PPC Ethernet connection (marked as Ethernet) forms the real-time Ethernet for adding HRT CORBA to the system. The ABB communication works with 10Mbit/s, which provides throttling so it does not disturb the real-time traffic too much.

When implementing the actual force controller (or some other application specific control as the visual servoing below), it is convenient to work graphically and define the controller by a so called block scheme. For the force control of the demo the definition of the controller is shown in Figure 3 (actually, the specific algorithms are internal to the force ctrl block).

The interface blocks (that after code generation provides the shared memory interface) would be good candidates to form CORBA interfaces. However, the installation procedure (installing modules in the Linux kernel during runtime) would mean object migration of hard real-time objects. That is outside the scope of the project, but it is interesting to see the need and the experimental possibilities.

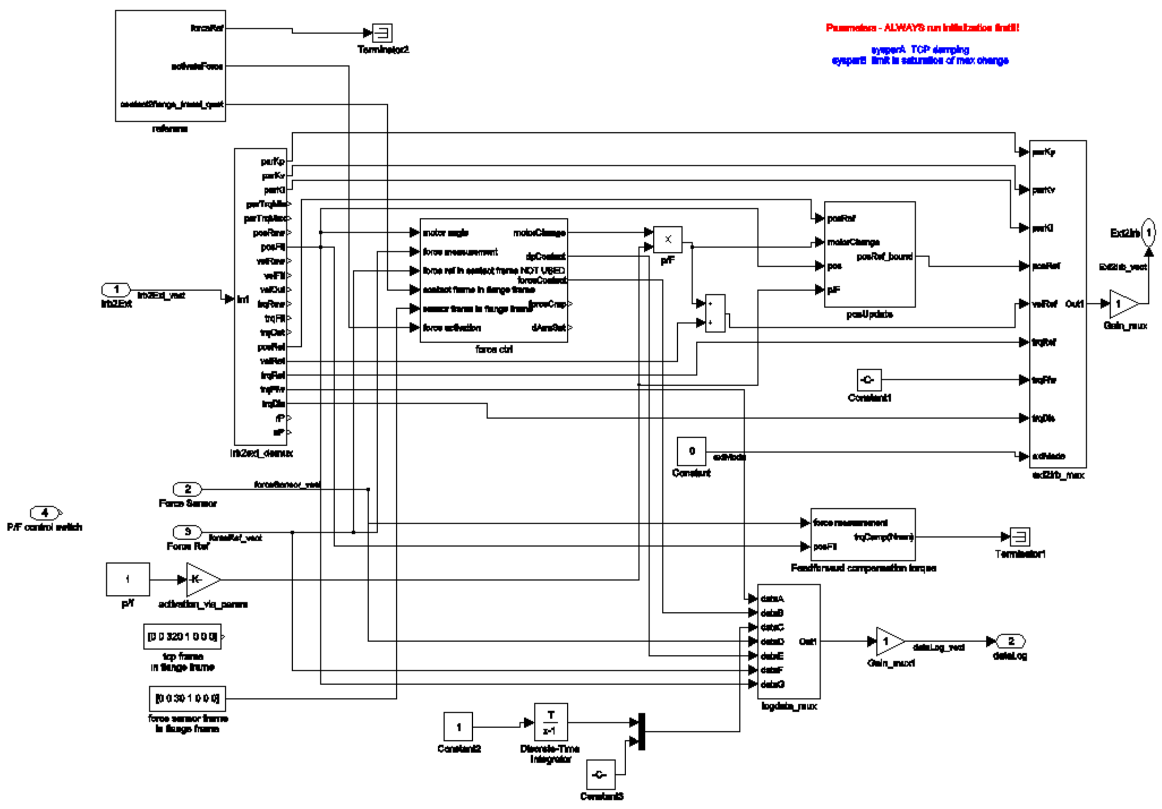


Figure 3 The compliant force controller component, defined graphically with interfaces in terms of the blocks marked.

4 The VME-based RCT implementation for hard real-time testing

The VME-based control computer replaces (due to earlier reconfigurations done at ULund) the original 4xCPU computer board of the Irb-2000 robot. The new external (located outside the control cabinet) computer boards are of type Motorola MVME2400 and MVME2600 with PowerPC (PPC) processors, except for a legacy M68030 board that performs some supervision tasks (like checking motor temperatures and checking the timing of periodic control threads in other CPUs). Other parts of the system consist of the distributed ETRAX processors and the host PC computers. The implementations of these three different parts of the system are commented in one section each below.

4.1 PPC control computers

Prior to the HRTC work, the in-house Stork real-time kernel was used, including our own communication stacks and IO device handlers. File systems and typical operating system type of calls were not available, since they were not really needed for the control of the robot. However, to run even a standard non-real-time ORB, the platform in terms of operating system and device drivers has to resemble typical OSs; otherwise it would be necessary to rewrite many parts of the ORB to tailor it to the specific platform at hand. That would be neither desirable nor possible within the project. The only reasonable option is to run Linux on the VME boards. During the planning and initial work we considered the porting/making of the Linux kernel for this type of hardware to be risky from a project time point of view, which was also the reason for the alternative platform according to the pervious section. For example, there is an internal PCI/PMC connector on the board for interfacing local peripherals and mapping of global VME-bus memory had to be done in two stages via the PCI logic. Hardware interfacing was on the other hand expected to be straight forward since we had device handlers working for the Stork kernel.

It turned out to be the opposite: the Linux system was better prepared for this type of hardware than expected, but adopting the hardware interfaces for IO and networking was harder than expected. In total, without hard real-time communication, the platform work proceeded according to the plans. To get the ORB and the OCI interface to run was, however, harder. First the needed PPC version of the ORB was decided to be developed after the TTP communication was supported. Then using the OCI support of the ICa ORB, due to our lack of such experience and some bugs, resulted in substantial development efforts. Parts of this was due to the fact that hard real-time support (using our ThrottleNet protocol) was considered already during the work with the soft/non real-time implementation. Details of the implementation can be found in the hard real-time implementation document.

The VME board of primary interest is the so called slave computer, which takes care of the low-level control and the communication with the ETRAX-based sensing and actuation of the robot joints. The old motion control code was rewritten in C (and also a Java version was made in an adjacent master thesis project but never used in the physical tests) for Linux/RTAI, but making a separate user-space implementation was not considered to be meaningful or worthwhile since such an implementation was likely not to work due to the fault detections that are built into our hardware. Instead, an instrumented real-time version was developed to permit introduction of the timing of non-real-time communication. Then the system can be booted using jitter-free communication, and then the instrumentation permits the activation of the timing of the non-RT communication.

In practice, for the ETRAX-PPC-ETRAX communication of the joint control, such a communication network was set up with dummy hardware control of the robot but with exactly the same type of processors. Then the control code and ORBs using TCP/IP and IIOP communication was used and the timing was measured and logged. The measured timing was then introduced as delays in the real-time implementation. An extra benefit of this procedure was that

© HRTC Consortium / Clearance: Consortium

fractions of the full delay could be used and changed during run-time. Refer to the documentation of the tests for further details and a video showing the physical results.

4.2 Distributed ETRAX computers

We had Linux running on the ETRAX computers already before start of this project. However, it was with Ethernet support coded by Axis into the kernel (not in a separate module), without RTAI to prepare for the hard real-time part of the work, without TrottleNet, and without knowing if an ORB actually would be possible to run on this amount of memory.

Due to extensive technical difficulties with the development of a robustly working real-time communication, this work was divided into two approaches in order to be able to complete the project (almost) in time. That is, the non rela-time communication was possible to test as described in previous section, but preparing for the hard real-time part was done in two ways. First, the full ThrottleNet compatible implementation of the networking (including extraction of the Ethernet driver from the kernel to a separate Linux kernel module) was done on separate ETRAX computers for testing of the communication without the physical robot. Secondly, the two ETRAX boards connected to the robot hardware were used with modified integrated Ethernet drivers to accomplish ThrottleNet compatible traffic (but hard-coded without separate kernel modules).

The installation of the ICa ORB on the ETRAX went well. Out of the available memory of 3 Mbytes, 30% was used whit the CORBA support. Clearly it would be desirable to better determine which parts of RT CORBA that really are necessary in order to further decrease the memory footprint. Our impression is that minimumCORBA [minCORB] with extended (for configuration of the hard real-time communication) OCI support would be appropriate.

4.3 Host computers

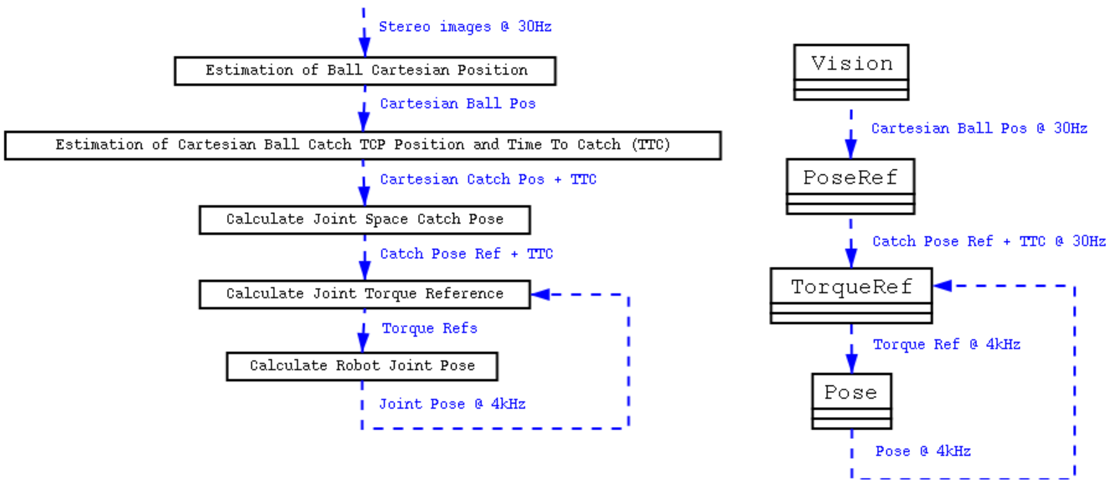
The vision system is running in soft real-time following standard video rate (30Hz). Three computers are involved in the current system; two Sony FireWire digital cameras are interfaced with a PC (running Windows) that is responsible for image acquisition and image processing (digital green filter and segmentation). A Matlab Linux PC is responsible for calculating a catch point by upgrading recognized image feature points to Cartesian space. Finally, a Solaris Sun workstation runs a Java application for converting Matlab communication to appropriate CORBA calls into the robot controller.

5 Virtual testbed

The simulation/analysis toolbox TrueTime mentioned in the RCT design is documented in a separate deliverable. Here the virtual testbed refers to the simulation of the RCT with appropriate visualisation to illustrate the testbed experiments. Since true real-time performance cannot be assumed to work on such computers, time driven software may need to be simulated in an event driven manner. The CORBA interfaces should of course be the same, and the dynamic behaviour should be close to that of the physical platform.

As mentioned the RCT design, visualization will either be based to dedicated OpenGL-based graphical models, and/or implemented in Java3D in the case that the virtual testbed is Java-based. Initial test with both principles have been carried out, but decision about the best technique was postponed until implementation phase. It was found that an OpenGL-based visualisation was preferable since it permitted appropriate 3D reconstruction of moving objects using stereo cameras in the virtual world. The difficulty was that the cameras and other objects need to be synchronised in time, not only for the simulation of motions but also for the rendering of the image since the image forms the feedback data for the visual servoing. The specific algorithms and are outside the scope of this document, and the source code listing is omitted for brevity but can be obtained by e-mail request from the consortium.

The data flows comprising the two RCT control loops are simulated in the virtual RCT as follows. Images are obtained from the virtual world (the native part of the virtual RCT) at a (simulated) 30 Hz frequency (left part of the picture below). Based on the images the target joint positions are computed and passed as references to the joint servo control, which runs at a rate of 4 to 8 kHz (right part of the picture below, where the rate is stated to be 4 kHz).


Stereo images @ 30Hz
Estimation of Ball Cartesian Position
Cartesian Ball Pos
Estimation of Cartesian Ball Catch TCP Position and Time To Catch (TTC)
Cartesian Catch Pos + TTC
Calculate Joint Space Catch Pose
Catch Pose Ref + TTC
Calculate Joint Torque Reference
Torque Refs
Calculate Robot Joint Pose
Joint Pose @ 4kHz
Vision
Cartesian Ball Pos @ 30Hz
PoseRef
Catch Pose Ref + TTC @ 30Hz
TorqueRef
Torque Ref @ 4kHz
Pose
Pose @ 4kHz

© HRTC Consortium / Clearance: Consortium

The interconnection of these two data flows and the involved IDLs are:

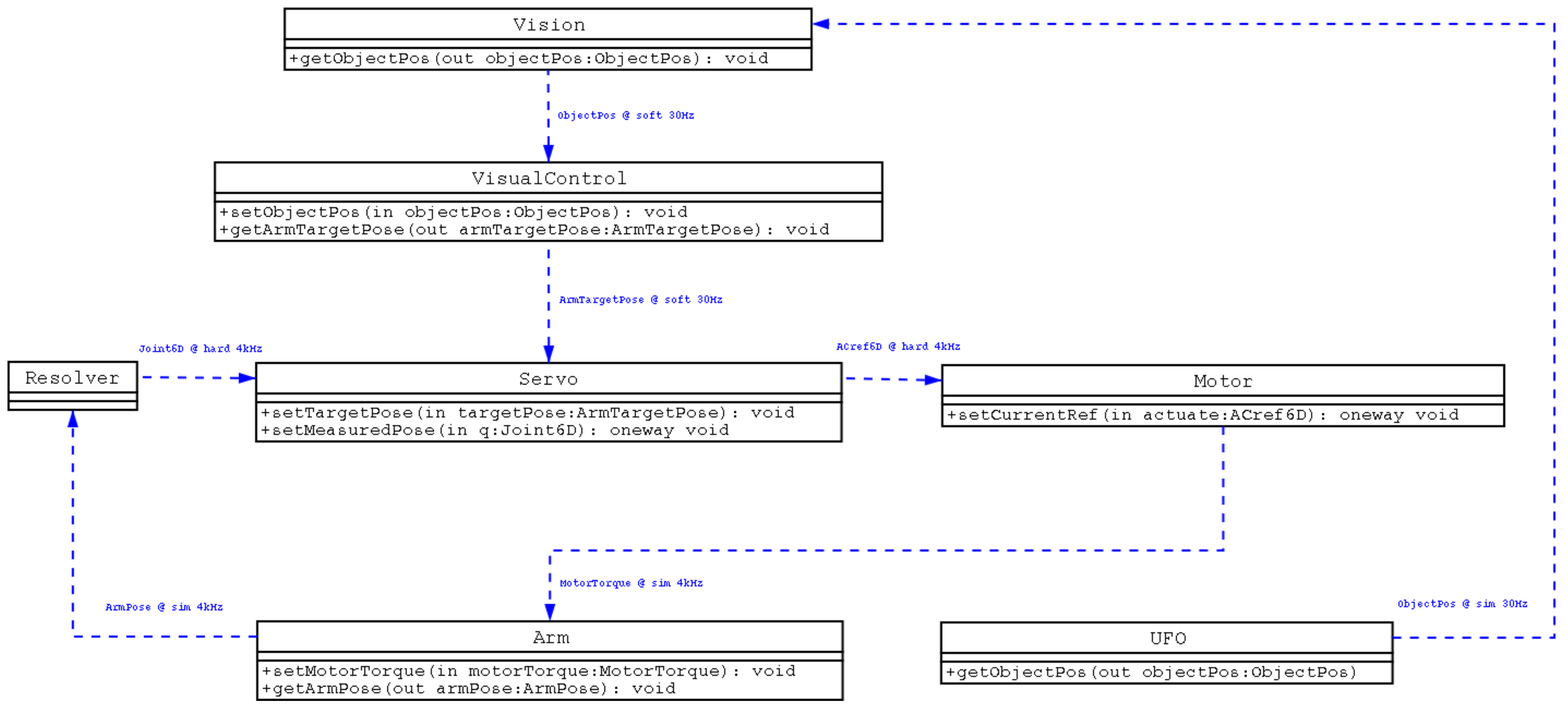

Vision
+getObjectPos(out objectPos:ObjectPos): void
ObjectPos @ soft 30Hz
VisualControl
+setObjectPos(in objectPos:ObjectPos): void
+getArmTargetPose(out armTargetPose:ArmTargetPose): void
ArmTargetPose @ soft 30Hz
Joint6D @ hard 4kHz
ACref6D @ hard 4kHz
Resolver
Servo
+setTargetPose(in targetPose:ArmTargetPose): void
+setMeasuredPose(in q:Joint6D): oneway void
Motor
+setCurrentRef(in actuate:ACref6D): oneway void
MotorTorque @ sim 4kHz
ArmPose @ sim 4kHz
ObjectPos @ sim 30Hz
Arm
+setMotorTorque(in motorTorque:MotorTorque): void
+getArmPose(out armPose:ArmPose): void
UFO
+getObjectPos(out objectPos:ObjectPos)

Figure 4 The distributed objects/IDLs that form the virtual RCT.

www.hardrealtimecorba.org

Both the Arm and its simulated dynamics and the other objects, including the Unidentified Flying Object (UFO) to catch, are part of the virtual world that is natively implemented based on OpenGL by a set of 33 C++ classes. The rest of the implementation was done in Java. The main part or the implementation is the classes implementing the IDLs above. The IDLs are the same as later used in the physical testbed implementation, but we found it more convenient to work with CORBA in Java (that is, hard real-time Java for hard real-time CORBA would be preferable, but outside the scope of this project).

In principle, neglecting timing and dependability issues, the control computations within each node could be carried out by a main application calling the methods of the various distributed objects (that have access to the IO). For real-time, and in particular for reliable hard real time, distributed active objects that periodically perform control computations and actions are needed, as shown by results from other partners of the HRTC consortium (TUWien). In the RCT, we accomplish that in a CORBA-compatible way by having active objects implementing an Activity interface providing a start method. Calling start, as in Java, means creating an active object with its own thread of execution. Here, on the other hand, instead of actually creating a new thread, we occupy a servant thread that performs according to arguments to start (details not worked out).

Starting an active object can also involve scheduling the traffic within the local network, influencing scheduled ThrottleNet or TTP in a manner that becomes rather independent of the actual protocol used. We found the virtual RCT to be valuable for exploring these and other possibilities.

6 References

[CORB] Common Object Request Broker Architecture (CORBA/IIOP) Specification 3.01, Object Management Group, Needham, MA, U.S.A., 2002, http://www.omg.org

[RTCORB] OMG: Real-Time CORBA 1.0 Specification, http://www.omg.org

[minCORB] OMG: Minimum CORBA Specification, version 1.0, http://www.omg.org/docs/formal/02-08-01.pdf

[PCI] PCI Special Interest Group: “PCI Local Bus Specification”, 1998, http://www.pcisig.com

[PrPMC] VITA Standards Organization (VSO): “Processor PMC Standard For Processor PCI Mezzanine Cards”, 1999, http://www.vita.com

[ETRAX] Axis Developer Board LX, Axis Communications, 2002, http://developer.axis.com/products/devboard/index.html