Available online at www.sciencedirect.com

ScienceDirect

Robotics and Autonomous Systems 56 (2008) 5–13

Robotics and Autonomous Systems

www.elsevier.com/locate/robot

When hard realtime matters: Software for complex mechatronic systems

Berthold Bäuml*, Gerd Hirzinger

Institute of Robotics and Mechatronics, DLR – German Aerospace Center, 82234 Wessling, Germany

Available online 2 October 2007

Abstract

A still growing number of software concepts and frameworks have been proposed to meet the challenges in the development of more and more complex robotic systems, like humanoids or networked robotics. The issue of hard realtime, however, has not been the main focus of such concepts, but is essential for building and controlling mechatronic systems. Here we discuss the specific demands of complex mechatronic systems and present a software concept, the "agile Robot Development" (aRD) concept, which we developed at our institute to pragmatically address these demands. We show that the performance of current computing and communication hardware allows for a flexible component-based concept with distributed execution, even in hard realtime with rates in the kHz range.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Software concept; Hard realtime; Mechatronic system; Distributed computing; Component-based software engineering

1. Introduction

Over the last years robotic systems reached a new level of complexity. Unlike systems with a six degrees of freedom (DOF) arm with a gripper or simple mobile robots, we see now torque controlled redundant arms, articulated hands, humanoid walking robots, cooperating swarms of mobile robots or service robots communicating with sensor networks installed in their environments. To address the challenges of developing software for such systems a number of software concepts and frameworks have been proposed. This number is still growing as until now no general abstraction has been found that fits well with all the specific demands of the diverse robotic applications and hardware.

Prominent representatives are ORCA [15], MARIE [17], MIRO [30], Player [31], OROCOS [5] MCA [2], OpenHRP [23], YARP [27] and Microsoft Robotics Studio [3]. They are all based on the idea that a complex robotic system should be composed from interacting modules or components in the sense of the component-based software engineering approach [19] with all its benefits such as flexibility, code reuse or decoupling of the development flow in a team. To allow the components to be distributed on a network of heterogeneous computers all approaches also provide tools for simplifying and standardizing communication.

1.1. Demands of mechatronic systems

In our institute the specific demands for a software concept arise from building and controlling highly complex mechatronic systems, e.g. the DLR Light-Weight-Robot arms (LBR), DLR Hands [20] or the recently built upper humanoid body Justin (see Fig. 1) with 41 DOF [29] (Section 4 shows further examples). The two main demands are: first, to provide scalable computing resources in hard realtime to allow for computationally demanding control loops in the kHz range (up to 10 kHz in the near future) running over all DOF and second, to support an "agile development flow" of a small, tightly interacting team of experts in the spirit of the "Agile Software Development" methodology [9,16]. Such a flexible, iterative and rapid development flow is essential for building complex systems, especially when working on research prototypes. From our experience key points that allow for an agile process in robotics are that the software concept used should it make easy to connect new hardware components like sensors and actuators, to scale computing resources by simply adding more CPUs, to integrate software components from different developers and to flexibly reconfigure the physical as well as the functional communication structure of the system, which is

* Corresponding author.
E-mail address: berthold.baeuml@dlr.de (B. Bäuml).

0921-8890/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2007.09.017

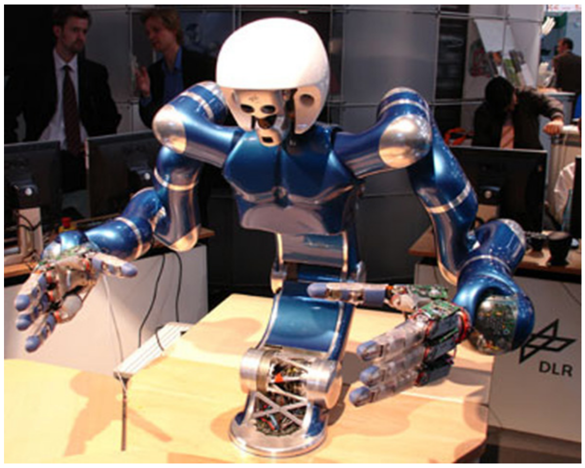

Fig. 1. An example for a complex mechatronic system: the DLR upper humanoid body Justin [29] with 41 DOF. This system is built from two DLR-LBR-III arms with 7 DOF each, two DLR-Hand-II with 12 DOF each [20] and a torso with 3 DOF.

essential for iterative rapid prototyping. Special to mechatronic systems is the high complexity of the part running in hard realtime, consisting of e.g. device drivers, sensor processing, controllers, inverse kinematics, collision avoidance, and state machines (see Fig. 2 for an overview of a complex mechatronic system).

To summarize: for complex mechatronic systems an easy to use and flexible component-based software concept allowing for distributed execution while guaranteeing hard realtime is desirable.

On the other hand, the performance of standard hardware has reached a level where such a component-based, more abstract view on the system is possible even when hard realtime has to be guaranteed. Most important for that are the fast digital buses of up to 1 Gb/s inside and to the robot components and the high computing power of commodity systems in combination with a flexible communication infrastructure built from cheap components like Gigabit-Ethernet adapters and switches.

1.2. Robotic software concepts

The software concepts mentioned before have been successfully used in different fields of robotic applications. In what follows we briefly discuss to what extent they meet the desired requirements for mechatronic systems.

ORCA [15], MARIE [17], MIRO [30] and Player [31] are used in mobile robotics, where the realtime constraints are rather soft with rates in the 100 Hz range. Also Microsoft Robotics Studio [3] is targeted on soft realtime applications so far, as the underlying operating system (OS) is Windows XP.

OROCOS [5] provides hard realtime and has been successfully used in robotic applications with control rates of more than 500 Hz and up to two independently controlled 6 DOF industrial robots. However, it is questionable if OROCOS in its current form can cope with the complexity of e.g. a humanoid robot with some ten to 100 torque-controlled DOF, where distributed computation in hard realtime is inevitable. In the current version of the "Real Time Toolkit" (RTT 1.2.1) of the OROCOS project all communication between distributed components is still based on a CORBA layer (an alternative "Distribution Library" is only planned for the future), for which realtime execution is not possible as the manual states: "Components should not call remote components during real-time execution" [6].

OpenHRP [23] is designed for the development of humanoid robotics applications. It is based on RT-Middleware [11] for inter-component communication, which uses CORBA and so allows for distributed execution. In all applications reported so far, however, the hard realtime parts running the low-level controllers with rates in the kHz range have been implemented in monolithic modules using proprietary communication to reach the desired performance.

MCA [2] is used for complex robotic systems like the humanoid robot ARMAR [13]. It allows for a hierarchical composition of e.g. controller components while providing hard realtime. But as it uses TCP/IP for network communication, the concept does not natively provide hard realtime for distributed execution.

YARP [27] is another concept used in a number of complex robots like Domo [18]. It is lightweight, allows for the configuration of the quality of service (QoS) of the inter-component communication and is portable by using ACE [22]. In all the reported robotic applications the low-level high-rate controllers run on dedicated DSP boards. But as YARP also supports the realtime OS QNX [7], it would be interesting to see how it performs in a complex mechatronic system like Justin with high control rates and distributed computing on networked PCs.

In the rest of the paper we first discuss in more detail the demands in developing software for complex mechatronic systems by taking a closer look on our humanoid upper body system Justin. Then we introduce a simple software concept, the "agile Robot Development" (aRD) concept [14], that we developed at our institute to pragmatically address this demands. The key points of the aRD concept are first to add only a thin layer above the realtime operating system to get the full hardware performance and second to have control over the quality of service (QoS) of the connection between components to meet the different hard, soft and non-realtime constraints. Finally we present some performance examples and give a overview of other complex robotic applications we have realized with the aRD concept.

2. Analysis of a complex mechatronic system

As a concrete example for a complex mechatronic system we analyze here the humanoid upper body system Justin we developed at DLR for performing experiments in the control of two-handed manipulation.

2.1. System overview

A system overview is given in (Fig. 2). Justin [29] is built from five robot components: two LBR-III arms (7 DOF each), a torso (3 actuated DOF) and two DLR-Hand-II (12