Advanced Engineering Informatics 24 (2010) 4–13

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

Architectural paradigms for robotics applications

Michele Amoretti [a], Monica Reggiani [b,*]

[a] Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Parma, 43100 Parma, Italy
[b] Dipartimento di Tecnica e Gestione dei Sistemi Industriali, Università degli Studi di Padova, 36100 Vicenza, Italy

ARTICLE INFO

Article history:
Received 12 June 2009
Accepted 5 August 2009
Available online 2 September 2009

Handled by Prof. I. Smith

ABSTRACT

In recent years, several technical architectural paradigms have been proposed to support the development of distributed and concurrent systems. Object-oriented, component-based, service-oriented approaches are among the most recent paradigms for the implementation of heterogeneous software products that require complex interprocess communications and event synchronization. Despite the sharing of common objectives with distributed systems research, the robotics community is still late in applying these research results in the development of its architectures, often relying only on the most basic concepts.

In this paper, we shortly illustrate these paradigms, their characteristics, and the successful stories about their application within the robotic domain. We discuss benefits and tradeoffs of the different solutions with the goal of deriving some practical principles and strategies to be exploited in robotics practice. Understanding the characteristics, features, advantages, and drawbacks of the different paradigms is, indeed, crucial for the successful design, implementation, and use of robotic architectures.

1. Introduction

The technological development of robotics research will soon lead to the marketing of robots that can play a key role in supporting people in their everyday tasks. Pursuing a specific objective while dealing with a dynamic environment and ensuring a safe interaction with human beings, requires a complex multifunctional structure for robot control, where heterogeneous hardware and software components interact in a coordinated manner. Additionally, further requirements are being introduced by an increasing number of projects [1,2] adding cognitive requirements while preserving pervasive requisites of autonomous robotics design, i.e. the capability to have a real-time interaction with the real world [3].

The robotics community has recently proposed several architectures for the development of robot control software [4–9]. This includes the avoidance of monolithic development methodologies since they are unable to deal with the problem complexity. Despite the large number of significant proposals, there is still a lack of common, suitable solutions that would allow the reuse of previous efforts. The main reason for this failure is the difficulty of clearly describing and formally defining a problem domain which is still unclear in the field of multifunctional robots: for the same problem, different research projects still produce different specifications for its domain. This also holds for cognitive robotics research where projects only share a common understanding of cognition as the ability to think or reason about embodiment worlds, but there are quite different assumptions about the representation, organization, utilization, and acquisition of knowledge. This has a huge impact on the final software architectures as it often prevents the exchange of software solutions developed by different research groups.

Even if the robotics community is still not in the stage of avoiding the recreation of incompatible solutions, a plague which is common to other software research fields, it would greatly benefit from the advances and maturity reached by distributed technology research. This research field is already converging toward a few technical architecture paradigms, and mature implementations of these ideas are freely available in the form of software middlewares supporting complex interprocess communication, event synchronization, and data distribution. A thoughtful application of these research results in the development of robotic software architectures would, at least, alleviate the cost of re-invention of core concepts and techniques for the control of distributed devices. Nevertheless, their application to robotics research is still late, often relying only on the basic concepts of the available middlewares.

In this paper, we shortly introduce three technical architecture paradigms that have been successfully exploited in the robotics field (Section 2). Their characteristics and successful stories within the robotic domain are discussed in detail in Sections 3–5. We discuss the benefits and tradeoffs of the different solutions with the goal of deriving some practical principles and strategies to be

* Corresponding author. Address: Dipartimento di Tecnica e Gestione dei Sistemi Industriali, Università degli Studi di Padova, Stradella San Nicola, 3, 36100 Vicenza, Italy. Tel.: +39 329 9860296.
E-mail addresses: michele.amoretti@unipr.it (M. Amoretti), monica.reggiani@unipd.it (M. Reggiani).

1474-0346/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.aei.2009.08.004

exploited in robotics practice. Understanding the characteristics, features, advantages and drawbacks of the different paradigms is indeed crucial for the successful design, implementation, and use in robotic architectures. Finally, we present a set of guidelines with an in-depth discussion about the influences and impacts of architecture paradigms on robotic applications to drive the design of control software architecture for robotics projects (Section 6).

This paper focuses on the analysis of technical architecture paradigms and software strategies for their use in the robotics domain. A performance comparison of the final control software architectures is outside of the paper's objectives. Interested readers can refer to other papers dealing with the comparison robotics architectures [10,11] and to the middleware activity of the Rosta project [12].

2. Architectural paradigms

The development of complex cognitive embodied systems is a challenging task as it requires a collection of behavior control abilities, including perception, manipulation, and learning. These abilities have to work concurrently and have to collaborate through the exchange of available knowledge. The design of intermodule communication and event synchronization is therefore of main importance during the development of the control software infrastructure. Several design methodologies and architectural paradigms for data communication have been proposed by the distributed computing community. This section is a short review that will introduce the evolution process that brought the development of the three technical architectural paradigms discussed in this paper.

Fig. 1 illustrates several abstraction layers, of increasing complexity, for distributed applications. At least one paradigm lays on each layer, i.e. a pattern or model defining the best design practices.

At the lowest level, *Message Passing* is the fundamental paradigm for distributed applications and provides an abstraction to encapsulate the details of the network communication and the operating system. Intermodule communications are based on *send* and *receive* primitives that allow input/output in a manner similar to the file I/O.

One abstraction layer consists of Message-Oriented Middleware (MOM) and Remote Procedure Call (RPC), which are two of the most prominent communication paradigms [13]. In the MOM paradigm, a message system serves as an intermediary among separate, independent modules. The message system acts as a switch for messages, allowing modules to exchange messages asynchronously, in a decoupled manner. Using a Point-to-Point communication model, MOM forwards a message from the sender to the receiver's message queue. Compared to the basic message-passing model, this paradigm provides the additional abstraction for asynchronous operations. Their support with message passing would have required low-level implementation through threads or child processes. Another MOM communication model is Publish/Subscribe which, at each message, associates a specific topic, task, or event. Modules interested in the occurrence of a specific event may subscribe to messages for that event. When the event occurs, the process publishes a message announcing the event or topic and the MOM message system distributes the message to all the subscribers.

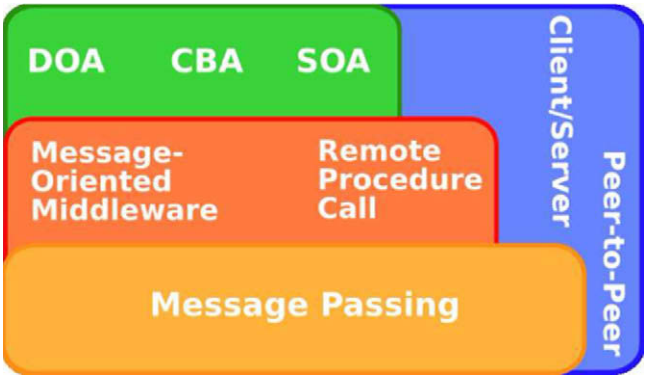

Fig. 1. Distributed architectural paradigms, at different abstraction layers.

The second paradigm of this layer, the Remote Procedure Call (RPC), allows distributed software to be programmed like conventional applications which run on a single process. A Remote Procedure Call causes a subroutine or procedure to be executed in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction. The programmer, therefore, writes the same code whether the subroutine is local or remote in respect to the executing process.

An increasing request for modularity and abstraction is what drove the development of the three architectural paradigms in the last abstraction layer. The Distributed Object Architecture (DOA) paradigm (Section 3) is based on the object oriented approach and is an improvement over the first attempts to provide platform independent solutions for interprocess communication. In particular, remote method invocation is the object-oriented equivalent of Remote Procedure Calls, where the remote object takes the role of the remote process. In this model, a process invokes the methods in a (remote) object, which may reside in a remote host. As with RPC, arguments may be passed along with the invocation. A following step introduced the concept of software components [14] created with the objective of promoting the reuse of design and implementation efforts. The final objective of the Component Based Architecture (CBA) paradigm (Section 4) is the development of components, eventually from multiple sources, that can be deployed according to customers' needs, often evolving during the project's lifetime. A recent trend for the development of modern large-scale distributed and mobile systems is calling for a new solution that will be better able to support an automated use of the available distributed resources. The idea of presenting software as a service is at the base of the Service-Oriented Architecture (SOA) paradigm (Section 5). The SOA has been recently introduced to provide loosely coupled, highly dynamic applications.

The previously described paradigms address several needs in abstraction granularity for the development of distributed applications. Another significant problem is to guarantee the reliability and efficiency of the whole distributed system by choosing the most scalable overlay scheme. The *client/server* scheme assigns asymmetric roles to the collaborating processes. One process, the server, plays the role of resource provider, passively waiting for request arrivals. The other (client) issues specific requests to the server and awaits its replies. The *peer-to-peer* (P2P) paradigm, envisions direct resource sharing among participants having close capabilities and responsibilities. Whereas the client/server paradigm is an ideal model for centralized robotic applications, such as teleoperations, the peer-to-peer paradigm is more appropriate for cooperative robotics, swarm robotics, and ambient intelligence.

In the next section, we focus on high-level solutions for the communication problem, introducing the basic characteristics of the DOA, CBA and SOA paradigms together with some of the most representative examples of their application in the robotics domain. This will lay the background that is required to motivate the choice to apply the different paradigms when a new robotic application must be developed.