Middleware Architectures in Embedded Systems Application

Gustavo André Nunes Ferreira
Polytechnic School – University of São Paulo - Brazil
gandre@usp.br

Lucas Antonio Moscato
Polytechnic School – University of São Paulo - Brazil
lamoscat@usp.br

Abstract. *Embedded systems, in special robotics systems, present a multiplicity of devices in its structure, often based on different technologies of the hardware and software, which in turn are connected by different network technologies and protocols, resulting in a considerably heterogeneous and distributed architecture.To solve such problem, the concept of middleware emerged in the area of distributed systems. A middleware is a layer of software integration developed to assist in the solution of the inherent complexity and heterogeneity of the distributed systems.The main objective of this paper is to investigate, based on the experiences reported in specialized bibliography related to this issue, in examples of use of middleware architectures in applications of embedded robotics, particularly with devices of low availability of resources (memory, capacity of processing and communication), such as microcontrollers and smart sensors, and analyzes of different types of approach used for a convenient use of these architectures with such devices and applications.*

Keywords*: middleware, embedded systems, distributed systems, robotic*

1. Introduction

Embedded systems, particularly robotic systems, are made of many devices of different natures. These applications will contain three sets of basic elements: sensors (of dislocation, positioning, etc) for the perception of its execution environment, actuators represented typically by motors, and one or more units of information system processing (that range from the simplest microcontrollers to the most sophisticated microcomputers). Such multiplicity of devices in a robotic application, many times based on hardware and software different technologies, which in turn are connected by different technologies and protocols of networks, generates a considerable heterogeneous and distributed architecture. This contributes to turn extremely complex, tedious and susceptible to errors the task of software development for robotic applications (Utz *et al.*, 2002).

Aiming to avoid such problem, the middleware concept appeared in the area of distributed systems, a software integration layer designed to help managing the complexity and heterogeneity inherent in distributed systems (Gill and Smart, 2002). Middleware architecture consists fundamentally of a software integration layer, resident above of the operational system and the communications substratum, and offers abstractions of high level and a set of services, with the objective of facilitating distributed programming. As examples of well known middleware architectures currently can be mentioned: RPC (Remote Procedure Call), Sun's original client-server procedural programming model; DCOM (Distributed Component Object Model), Microsoft's distributed component model for Windows platforms; Java RMI, the Java community's remote method invocation model for the Java language; CORBA (Common Object Request Broker Architecture), the OMG (Object Management Group) standard for language and platform independent distributed object computing (Gill and Smart, 2002). In the present, it is already possible to find a considerable number of publications approaching the use of middleware architectures, particularly of CORBA implementations, in robotic applications. For example: Bottazzi *et al.* (2002) approaches the use of such architectures in robotic teleoperation systems; Utz *et al.* (2002) mentioned the handiness of such architectures in mobile robots applications; Schmidt (2002) and Gong (2003) make considerations about middleware architectures in real-time and embedded systems; Sanz and Alonso (2001) approaches the use of CORBA architectures in control systems, refer to applications in robotic teleoperation, risk management systems in industrial chemical processes, real-time video systems, among others.

The objective of this paper is to present an overview of the use of middleware architectures in embedded robotics applications, particularly with devices of limited availability of resources (memory, processing and communication capacity), such as microcontrollers and smart sensors, analyzing the different types of approaches used for a convenient use of these architectures with such devices and applications. In section 2, the three main types of existing approaches for the employment of middleware architectures in embedded systems are presented, together with examples for each approach in section 3. In section 4, are found the analyses and conclusions of this work, as well as suggestions for potential future research works.

2. Middleware architectures for Embedded Applications

One of the main arguments found in specialized literature against the use of middleware architectures in embedded applications is that it adds unnecessary code to the system, harming its performance. This argument becomes still stronger to applications of limited resources of memory, processing and communication. In fact, general-purpose middleware architectures implementations can have significant amount of non useful code. In contrast, only one

subgroup of such middleware architectures can offer many of the benefits of the use of these structures, without harming significantly the system performance.

Recent researches in this area have explored some forms of assuring only the essential characteristics of the middleware architecture for the target application, thus reducing the impact in the performance case to case.

In this context, the forms of main current confrontation of this question can be divided in three approaches: a first one based in the uses of delegation, a second based in the reduction of the middleware infrastructure and a third that uses component-based middleware architectures.

In following sub-sections the three approaches mentioned will be better explained.

2.1. Delegation Based Approach

This approach guesses the presence of a system with resources to integrate one or more devices through the use of a traditional middleware infrastructure. Such devices do not present resources to support such infrastructure; however it is desirable that they offer some type of service for the system as a whole.

As solution for the previously described situation, this approach makes the devices that do not support the middleware infrastructure to use some type of delegation to become accessible.

In this context, can be mentioned as examples the Jini technology on RMI (Jini Architecture Specification, 2003; Jini Network Technology, 2001) of the Sun, and the proposal Smart Transducers Interface on CORBA (Smart Transducers Interface Specification - Version 1.0, 2003) of the OMG.

2.2. Minimum Middleware Infrastructure Based Approach

In general, software libraries that implement traditional middleware architecture demand some megabytes of memory, what is beyond the capacity of the microcontrollers typically used in embedded applications.

In this approach there is an effort to reduce the infrastructure of the middleware ORB load in the embedded device, canceling non interesting functionalities for the application, but keeping the interoperability with the original middleware architecture. Such infrastructure will demand little more than a hundred of Kbytes of memory, being able to be loaded in typical microcontrollers of embedded applications.

The Minimum CORBA specification (Minimum CORBA Specification - Version 1.0, 2002) is an example of the approach based on the reduction of the middleware infrastructure.

2.3. Component Based Approach

Traditional architectures implementations are monolithic, or either, all the internal mechanism of middleware is presented as a black-box and will be always present in the system. As the application needs only 10% of that mechanism, what is true for the majority of the real applications, all the code is loaded in the application process.

Systems based on components are constructed through the combination of some software components, normally stored in archives in the local file system or a components repository accessible through the network. Programmers determine which components will be part of the system and how will be the inter-relationship. Component-based middleware architectures make possible the programmer to specify which the ORB (Object Request Broker) services will be needed and that will be loaded by middleware in initialization time.

This approach is the most flexible of the three presented in this paper, and the one that presents greater advantages in the solution of the complexities caused by the use of middleware architectures in embedded systems.

An example of component-based middleware is UIC (Universally Interoperable Core) (Kon *et al.*, 2002), developed for Ubicore LLC (http://www.ubi-core.com).

3. Examples of Middleware Approaches for Embedded Systems

In the sub-sections to follow, examples for each approach mentioned in section 2 will be briefly described.

3.1. Jini on RMI

The Jini technology appeared with the proposal of making the distributed computation an easier activity, allowing that devices create a dynamic communication network and share services in this network, forming one federation based in the RMI middleware architecture. Each device shows services that other devices of the federation can use.

For the case of devices with resources restrictions that disable them to support RMI, the Jini technology considers two types of solutions, one using a JVM (Java Virtual Machine) physically shared and another using a JVM shared by network. In first solution a group of devices uses a JVM physically shared as an intermediate layer between a device and the system or devices using the Jini technology (Jini Device Architecture Specification, 2003) (see fig. 1). Each device loads code written in Java programming language in the virtual machine, allowing interaction with the device and then the delegation for the virtual machine of the solicitations of interaction of the system.

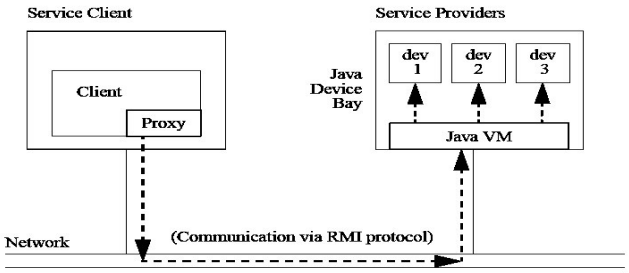

Figure 1. Jini technology on RMI: Shared JVM (Physical Option).

In second solution, JVM shared by network, a proxy of the virtual machine used for some devices will be available. They can discover such proxy existence and register with it when added to the network. The register can include the Java code necessary for a client of the device, as well as the code necessary for the proxy to communicate with the device (Jini Device Architecture Specification, 2003) (see fig. 2).

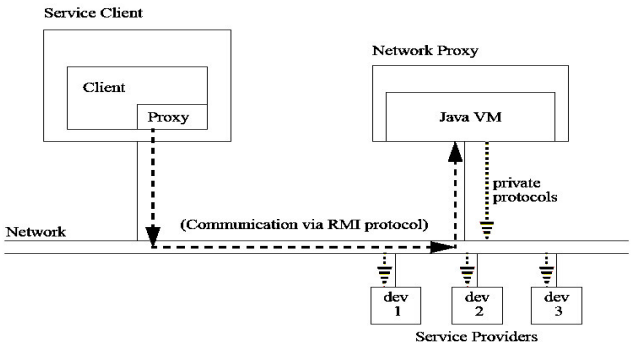

Figure 2. Jini technology on RMI: Shared JVM (Network Option).

An example of Jini technology use in applications of embedded robotics is cited in Long (2005), which present the Distributed Field Robot Architecture (DFRA), a distributed and object-oriented implementation of managerial architecture that provides consistent access to the capabilities of a team of robots in the form of Jini services. Hereby, the capabilities of each robot (such as sensors, actuators, and algorithms) can be dynamically discovered and used by the robots themselves or by human operators.

3.2. Smart Transducers Interface on CORBA

Differently of the Jini on RMI, the proposal of standardization Smart Transducers Interface of the OMG does not restrict the programming language used, offering additional transparency to the programmer in relation to the Jini on RMI. This specification allows that devices be addressed as if they were inside the systems that support full ORB CORBA (Gill and Smart, 2002). For this, a local application of a smart transducer must map the functionalities of the device in an interface file system (IFS), that is, a set of small elements of memory that typically are placed in the device local memory. The IFS is hierarchically structured into files and records enabling a unique addressing scheme. Such files and records are easily translated to some protocols, thus allowing the interoperability among diverse systems (see fig. 3). Typical smart transducers will fit in a microcontroller RISC of 8 bits with 128 bytes of RAM and 4 Kbytes of ROM (Elmenreich and Pitzek, 2003).

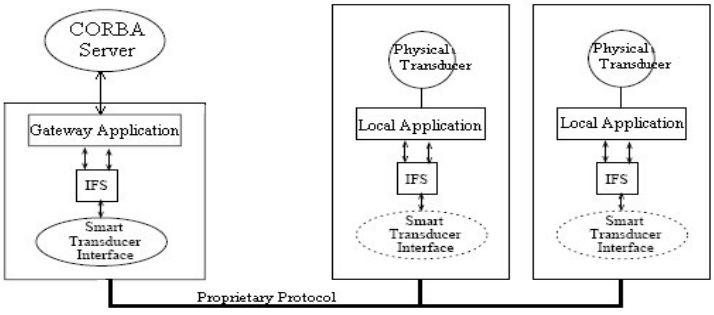

Figure 3. Application example using Smart Transducers Interface.

Elmenreich (2002) presents the implementation of two case studies of the smart transducers interface in robotics systems. The first case comprises a demonstrator with a robot arm that is instrumented by a smart transducer network partitioned into two clusters on which are integrated sensors, actuators and command units. The second case is an autonomous mobile robot, which shows the integration of new nodes and efficient communication despite of static communication schedules demonstrating the possibility of complex features like plug & play or reconfiguration.

3.3. Minimum CORBA

The Minimum CORBA specification is a subset of the specification CORBA for systems with limited resources. Clearly, such subset neglect some characteristics of the original specification, that still can be implemented by the application if necessary (see fig. 4).

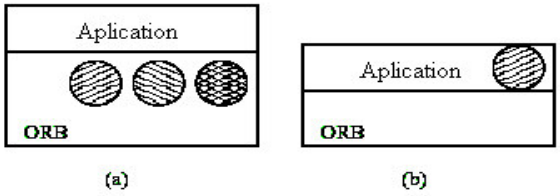

Figure 4. Characteristics (represented by patterned circles) omitted in minimum CORBA:
(a) CORBA; (b) Minimum CORBA.

Among the omitted characteristics, the dynamic aspects of architecture CORBA are noticeable (DII - Dynamic Invocation Interface and DSI - Dynamic Skeleton Interface), together with most of the Interfaces Repository and the support the interceptors. Moreover, many non essential functionalities are omitted of the interfaces of the ORB and POA (Portable Object Adapter). However, despite of all simplifications, Minimum CORBA preserves the interoperability with CORBA that implies in the support to the complete specification of IDL (Interface Definition Language) CORBA.

Figs. 5 and 6 present diagrams representing the components of ORB CORBA and the ORB Minimum CORBA, respectively.

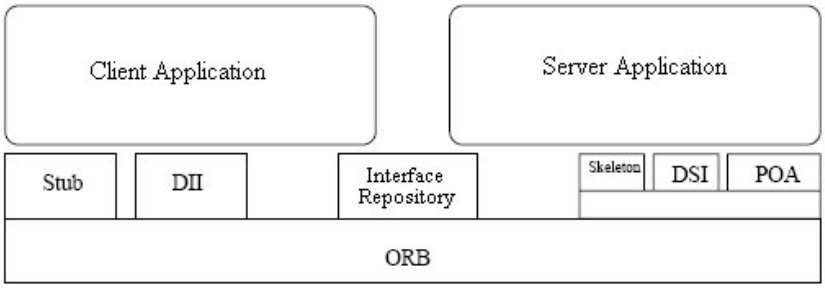

Figure 5. CORBA ORB components.

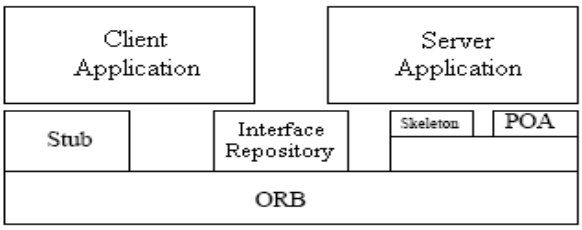

Figure 6. Minimum CORBA ORB components.

Schmidt (2002) cites the ACE ORB (TAO), an open-source CORBA C++ implementation that also includes a minimum CORBA compliant implementation. Furthermore, TAO is extensive used in the Miro architecture, a framework structured in architectural layers that, according to Utz *et al.* (2002), is successfully used in several projects striving for autonomous mobile robot control, like mapping of indoor environments, world modeling, autonomous self-localization, path planning, reactive execution, and others.

3.4. Universally Interoperable Core

The UIC defines a skeleton based on abstract components that encapsulate the functionalities found in main existing ORBs. Concrete implementations of these abstract components are loaded dynamically of form to meet the specific applications requirements. UIC defines different personalities (particular instances of a UIC obtained for specialization) that meet the specifications of different middleware standards. Fig. 7 presents a schematic diagram representing the abstract UIC (*UIC*), its specializations (*UIC-CORBA* and *UIC-RMI*) and the servers that export their functionalities (*CORBA Server* and *RMI Server*). Up to now, the *UIC-CORBA* is the only personality in advanced stage of development, which is capable to interact with CORBA ORBs using much less resources than the traditional ORBs.

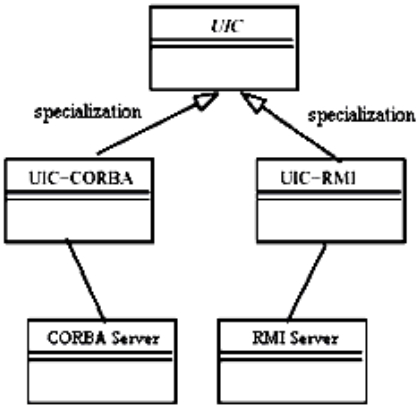

Figure 7. UIC schematic diagram.

The advantage of component-based ORBs for embedded systems can be illustrated by the code size of the libraries implementing a minimum client in UIC-CORBA. The size of middleware for a minimum client capable to send simple solicitations for a standard CORBA server is of 29 Kbytes in Windows CE, 72 Kbytes in Windows 2000 and 18 Kbytes in the PalmOS (Costa and Kon, 2002).

Due to the fact that this approach is the newest comparatively of the approaches mention in this paper, so far there are no reports of the robotics applications using component-based ORBs in the its implementations. The focus of the developments in component-based ORBs, including the UIC developments, has been targeted principally PDA's applications.

4. Conclusions

This paper is the result of the bibliographical research regarding the same types of approaches for the confrontation of the embedded systems complexities and heterogeneity by means of middleware architectures. Particularly, systems had been focused with limited availability of resources (memory, processing and communication capacity), such as the robotic applications.

On the basis of the examples of use of middleware architectures in robotic applications, was observed a trend for the choice of CORBA implementations. This fact is justified, as mentioned for Gill and Smart (2002), by two main factors in relation to the CORBA: CORBA constitutes an open standard for use and extension and closed for modifications without complete revision under the process of OMG standardization; to be both platform and programming language independent.

The first middleware approach, in section 2, for the confrontation of the embedded systems complexities and heterogeneity was to the delegation based. This approach appears as an option that brings advantages in specific cases, particularly where it desires to offer functionalities of a set of devices to a system that supports a complete middleware infrastructure.

The approach of the middleware infrastructure reduction based was presented as an example of the Minimum CORBA specification proposal for the OMG. Such specification is based on the omission of a series of characteristics of the original CORBA, in special the dynamic characteristics. Nevertheless, there is a comment here: recent researchers, among them Kristensen *et al.* (2003), Kon *et al.* (2002), Costa and Kon (2002), Román *et al.* (2001), and Eliassen *et al.* (1999), have shown to the successful field for the use of reflexives middleware architectures in embedded systems, especially in computation ubiquity. Such architectures are essentially dynamic, what would depreciate the Minimum CORBA initiative.

In this context, the most promising approach, both for the development of traditional embedded applications and for the development of computation ubiquity applications, is the component-based middleware architectures.

As future research works can be mentioned: an extended study regarding the component-based middleware architectures and of the reflexives middleware architectures; a study on the impact of the use of middleware

architectures in the real embedded systems performance; a study regarding the real gains (in terms of the development time, reusability, scalability, etc) of the use of middleware architectures in the embedded systems development.

6. References

Bottazzi, S., Caselli, S., Reggiani, M. and Amoretti, M. (2002). "A Software Framework based on Real-Time CORBA for Telerobotic Systems", Proceedings of IEEE International Conference on Intelligent Robots and Systems, IROS'02, Lausanne, Switzerland.

Costa, F. M. and Kon, F. (2002). 20o Simpósio Brasileiro de Redes de Computadores, SBC, Búzios, RJ, Brasil, chapter Novas Tecnologias de Middleware: Rumo Flexibilização e ao Dinamismo, pp. 1-61.

Eliassen, F., Goebel, V., Kristensen, T., Plagemann, T., Andersen, A., Rafaelsen, H. O., Yu, W., Blair, G., Costa, F., Coulson, G., Saikoski, K. B. and vind Hansen (1999). Next generation middleware: Requirements, architecture, and prototypes, Proceedings of The Seventh IEEE Workshop on Future Trends of Distributed Computing Systems, IEEE Computer Society, IEEE, Tunisia, South Africa, pp. 60-65.

Elmenreich, W. and Obermaisser, R. (2002). A standardized smart transducer interface, Proceedings of the 2002 IEEE International Symposium on Industrial Electronics, IEEE Industrial Electronics Society, IEEE, L'Aquila, Italy, pp. 164-169.

Elmenreich, W. and Pitzek, S. (2003). Smart transducers - principles, communications, and configuration, Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems (INES), IEEE, Luxor, Egypt, pp. 510-515.

Gill, C. D. and Smart, W. D. (2002). Middleware for robots? Proceedings of the AAAI Spring Symposium on Intelligent Distributed and Embedded Systems, Stanford, CA.

Gong, Y. (2003). CORBA application in real-time distributed embedded systems, Technical Report ECE8990, Mississippi State University, Mississippi, MS, USA.

Jini Architecture Specification (2003). Technical Report Version 2.0, Sun MicroSystems, Santa Clara, CA, USA. http://www.sun.com/jini/.

Jini Device Architecture Specification (2003). Technical Report Version 2.0, Sun MicroSystems, Santa Clara, CA, USA. http://www.sun.com/jini/.

Jini Network Technology (2001). Datasheet, Sun MicroSystems, Palo Alto, CA, USA. http://www.sun.com/jini/.

Kon, F., Costa, F., Blair, G. and Campbell, R. H. (2002). The case for reflective middleware, Communications of the ACM 45(6): 33-38.

Kristensen, T., Arnesen, L. P. S., Valen, E. and Plagemann, T. (2003). Interactive Multimedia on Next Generation Networks, Vol. 2899 of Lecture Notes in Computer Science, Springer-Verlag Heidelberg, chapter Evaluation of middleware for distributed objects on handheld devices, pp. 270-281.

Long, M., Gage, A., Murphy, R. and Valavanis, K. (2005). Application of the Distributed Field Robot Architecture to a Simulated Demining Task. Proceedings of the IEEE International Conference on Robotics and Automation, IEEE, Barcelona, Spain.

Minimum CORBA Specification - Version 1.0 (2002). Technical Report formal/02-08-01, Object Management Group, Needham, USA.

Román, M., Kon, F., and Campbell, R. (2001). Reflective middleware: From your desk to your hand, IEEE Distributed Systems Online (Special Issue on Reflective Middleware) 2(5). http://dsonline.computer.org.

Sanz, R. and Alonso, M. (2001). CORBA for control systems, Annual Reviews in Control 25(0): 169-181.

Schmidt, D. C. (2002). Middleware for real-time and embedded systems, Communications of the ACM 45(6): 43-48.

Smart Transducers Interface Specification – Version 1.0 (2003). Technical Report formal/03-01-01, Object Management Group, Needham, USA.

Utz, H., Sablatng, S., Enderle, S. and Kraetzschmar, G. (2002). Miro - middleware for mobile robot applications, IEEE Transactions on Robotics and Automation - Special Issue on Object-Oriented Distributed Control Architectures 18(4): 493-497.

5. Responsibility notice

The authors are the only responsible for the printed material included in this paper.