

# ON THE USE OF SERVICE ORIENTED SOFTWARE PLATFORMS FOR INDUSTRIAL ROBOTIC CELLS

Veiga G.<sup>1</sup>, Pires JN<sup>1</sup>, Nilsson K.<sup>2</sup>

*Mechanical Engineering Department, University of Coimbra, Portugal<sup>1</sup>  
Department of Computer Science, Lund University, Sweden<sup>2</sup>*

**Abstract:** The integration of different technologies reusing available production solutions, within an industrial robotic cell is a subject that requires further research and development. Technologies demanding high processing power, like machine vision or voice recognition systems, are normally easy to program but require proprietary languages and platforms, which constitutes an important problem during the integration phase, mainly related with communications and setup. This paper addresses these problems implementing a service-oriented architecture that uses Universal Plug-and-Play (UPnP) technologies. The architecture is tested using a real robotic manufacturing cell. Furthermore, the paper also presents and discusses a software application designed to program manufacturing cells whose building blocks are represented by UPnP devices. Finally, a tool for automatic generation of UPnP devices based on the information contained in speech recognition XML grammars is presented and discussed. *Copyright © 2007 IFAC*

**Keywords:** Intelligent Manufacturing Systems, Communication protocols, Computer communication networks, Concurrent Architectures, Flexible Automation.

## 1. INTRODUCTION

Due to their flexibility and programmability, industrial robots play a central role in manufacturing, tightly integrated with its surrounding equipment to accomplish the needed productivity. The evolution of industrial devices like PLC's, cameras, intelligent sensors, etc., with their special programming environments/languages/features for configurability and reuse of those devices, forms a bottleneck for system integration. Therefore, programming a manufacturing cell usually requires specific knowledge about different devices making it a work for trained specialists or multi-disciplinary teams.

To assist during system integration and to promote reuse of production solutions (that is, not only the components), the development of integration tools and architectures for setup of manufacturing cells is highly desirable for the future. In particular, this is important considering the needs of small and medium enterprises (SME). Since the majority of the time consumed during integration stages is related with communication and setup (configuration and tuning) problems, new technologies should specifically improve on these aspects.

From both a practical and a scientific point of view, in automation as in other scientific areas, it is important that proposed/published approaches/results are independently evaluated by other researchers. In

particular this should be important in industrial automation, since the actual applicability of certain technologies is confirmed or rejected far beyond the duration of ground-breaking developments.

To this end, in this paper, we will review some of the recently proposed technologies, and confront the most promising approach with experimental setups reflecting real applications. Both the analysis, the selection of technologies, the experimental setup with its results, and the conclusions concerning applicability are intended contributions. That should also contribute to the development of future plug-and-produce developments for SME manufacturing.

## 2. ANALYSIS

Superior flexibility and ease of use calls for plug-and-play components and high-level programming features. With some knowledge of so called holonic cells in mind (Gou, 1998), here referring to work-cells and their contained devices in automation, we may ask ourselves: what are the important properties to consider for work-cell components. Within each (mechatronic) device, real-time issues are of course important. Between cell components, however, there are mainly signal/event dependencies, but hard real-time requirements are not that common, as can be noticed from the use of PC-based software tools and so called soft-PLCs in many industrial automation systems today. For instance, a delayed event may

delay the production for some milliseconds, but the production process does usually not fail.

This means that in these structures all the real time control is done inside a device or software component, which provides high level services to the network.

Clearly, in special but important cases such as visual servoing and conveyor tracking involving feedback loops via several interconnected devices, we need to be able to accomplish shop-floor deterministic traffic, but such cases forms a smaller part of the integration problem. For the majority of component connections, we can instead think in terms of coarse-grained services, with synchronous calls for setup and asynchronous events for operation.

Safety is getting increasingly important for robotic work cells, with safety sensors and the trend of removing the fences around the machines. This might look like a contradiction to the use of PC-based software for cell control, but two facts simplifies the situation: First, safety sensors and controllers are configured separately, based on special hardware and certification procedures. Secondly, *safety critical* robot work-calls are not *mission critical*, like X-by-wire for vehicles and the like. Therefore, we can accept occasional failures, if it can be detected and handled (by stopping or performing another task).

Based on these insights and the aim for low-cost equipments, including the use of *de facto* standards such as TCP/IP based technologies, we look for modern automation-suitable architectures. This means networking with support for event-driven publish-subscribe communications, and the setup should support plug-and-play and high-level programming features.

### 2.1 Architecture

The presence of many high-processing power devices makes the concept of *service-oriented architecture* (SOA) particularly suitable for use in the industrial manufacturing cells. A SOA relies on highly autonomous but interoperable systems. The definition of a service is ruled by the larger context; this means that all technological details are hidden, but also that the concept that supports the service is more process related and less technologically related.

The SIRENA project (2005) pointed out advantages by the use of SOA in industrial automation (Fig. 1). Ahn et al (2005) addressed this issue in a different domain (mobile robotics), proposing a SOA for the device level as robot middleware.

It is also relevant to mention that the new Microsoft Robotics Studio, recently launched, uses decentralized system services (DSS) as SOA (Microsoft 2006a). This is a lightweight .NET framework that also relies on web technologies. Nevertheless is lack of maturity (final release in December 2006), this platform was adopted by some

major players in the robotics field, including some industrial ones, which are building solutions based on the Robotics Studio.

Table 1 Effects on the use of service oriented architectures in automation (SIRENA, 2005)

Aspect	Today	Tomorrow
System	Centralised Large, intelligent Controllers dumb devices	decentralised intelligent & autonomous devices
Commu- nications	polling client-server point-to-point	event-driven publish-subscribe, peer-to-peer
Set-Up	long and difficult manual programming tedious debugging	no programming plug-and-play context-aware configuration

Fig. 1. Effects of the use of service oriented architectures in automation (SIRENA, 2005).

Industrial applicability also calls for support of dumb (for example, wired IO) and legacy devices. These issues have already been addressed with success by James et al (2005a, 2005b) using specially designed gateway devices, as now also in the Robotics Studio.

There are many SOA proposed for the device level, and fairly nice revisions have been written that resume their basic features (Rekesh, 1999) (Bettstetter and Christoph, 2005). For the actual implementation, a SOA is in practice based on a *middleware platform*, which can be considered being a lower-level architecture. Such a platform should include suitable mechanisms for support of the SOA main characteristics: *addressing, discovery, description, control* and *eventing*.

### 2.2 Middleware platforms

In this work we consider three of the most relevant and available approaches: Jini (Jini 2006), UPnP (UPnP forum 2004) and *device profile for web services* (DPWS).

Jini is an architecture proposed by Sun Microsystems based on Java. This fact makes it platform independent but language dependent. It also carries a large memory footprint, due to the presence of a virtual machine, making it less appropriate for small devices.

UPnP and DPWS rely extensively on standard network protocols such as TCP/IP, UDP, HTTP, SOAP, XML, and web technology. This makes them platform and language independent, which is a major advantage for their adoption. XML formats are broadly used and accepted and provide modern data interchange mechanisms and communications.

Although similar in many aspects, the UPnP and DPWS architectures use different languages for device description and different protocols for discovery and event notifications. A proposal has been made to the UPnP foundation (Schlimmer 2004) for the convergence in the next major UPnP review. Nevertheless, the new Microsoft operating system, Microsoft Vista, supports both technologies under the name *plug and play extensions for Windows* (PnP-X 2006).

### 2.3 Selection and use of middleware platform

In the work reported in this paper, UPnP was selected. This choice is mainly supported by the maturity of the technology. In fact, there are considerable more development tools available, covering all the platforms, for the UPnP case. Similarity of both technologies implies that the code can easily be ported.

The basic elements of an UPnP network are: *devices*, *services* and *control points*. A *device* is a container of *services* and other *devices*. A *service* is a unit of functionality, that exposes actions and has a state defined by a group of state variables. A *control point* is a *service* requester. It can call for an action or subscribe an *evented* variable.

The basic steps that compose UPnP networking are:

Table 2 Basic steps that compose UPnP networking

Steps	Description
Addressing	Control point and device get an address.
Discovery	Control point finds interesting device
Description	Control point invokes actions on the devices
Control	Control point invokes actions on the devices
Eventing	Services announce changes in their state
Presentation	Control point monitor and control device status using a HTML User Interface

The first step, *addressing* occurs when a *device* or *control point* obtain a valid IP address. Normally the *dynamic host configuration protocol* (DHCP) is used; otherwise the *device* or *control point* uses de auto-IP mechanism.

Once *devices* are attached to the network and are properly addressed *discovery* takes place. *Devices* advertise its *services* to *control points* and *control points* search for *devices* in the network.

The *description* step allows a *control point* to obtain the necessary information about a *device*. This is done using the URL provided by the device in the *discovery* message. At this stage, the *control point*

has the information about the actions and state variables provided by a *device*. The *control* step consists on calls of actions present in a *device* made by a *control point*.

When the state of a *service* (modelled in their state variables) changes, the *service* publishes updates by sending messages through the network. This is called *Eventing*. These messages are also expressed in XML and formatted using the *general event notification architecture* (GENA).

Some *devices* may have *presentation* web pages. In this case a *control point* can retrieve a page from the specified URL, load the page into the browser and depending on the capabilities of the page allow a user to *control* the *device* and/or view the *device* status.

## 3. EXPERIMENTS

Given the selected most promising architecture and platform, the objective now is to:

- 1- Validate a service-oriented architecture has a general architecture for programming industrial robotic cells;
- 2- Test concepts that support the service definition;
- 3- Develop general software to program industrial robotic cells;
- 4- Validate different UPnP programming stacks, and their interoperability;
- 5- Test strategies for the automatic generation of UPnP devices;

### 3.1 Cell Description.

The robotic cell used in this demonstration is composed of an ABB IRB 140 robot, equipped with the new IRC5 controller, a conveyor controlled by a PLC (Siemens S7-200) and a web camera.

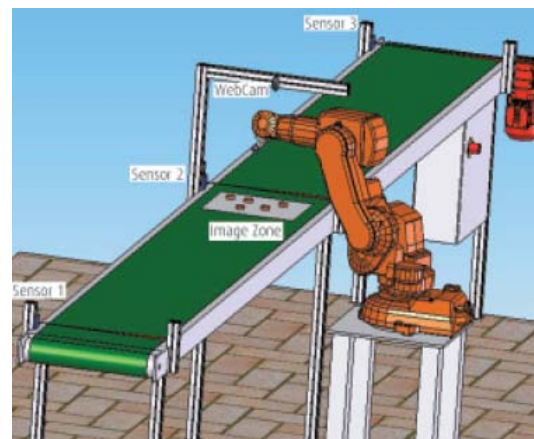


Fig. 1. General view of the cell used in this work. (Pires *et al* 2006).

Basically, the conveyor transports sample pieces over the machine vision system (Fig. 1), which calculates

the number and position of the pieces. The results are sent to the robot controller to command the robot to pick them from the conveyor and place them into a box. A detailed description of this setup is available at Pires *et al* (2006), where the author used an alternative solution based on a general client-server application developed using TCP/IP socket based communications. Since only the robot has built-in support for sockets communications, Pires *et al* (2006) used several PC based applications to distribute services over the network. They also developed two different clients to operate the cell: PC based GHMI (Graphical Human Machine Interface), and a PDA interface. There was also a speech recognition interface.

### 3.2 UPnP network and devices

The architecture proposed in this paper replaces the client-server (object-oriented) architecture with a SOA and provides some tools to support the creation of the software components necessary for the SOA. Consequently, five software applications were developed as described in Fig. 2. These applications correspond to five UPnP devices of the network.

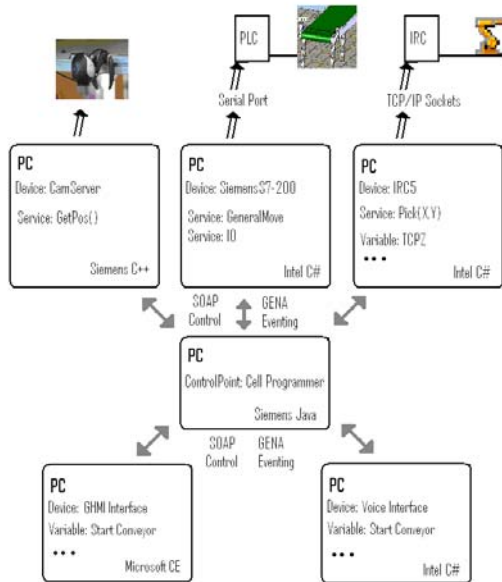


Fig. 2. UPnP network proposed in this paper.

Since both the industrial components of the system (PLC and robot) don't have native UPnP support, it was necessary to develop an extra software layer to integrate these devices into the UPnP network. The objective is to pool the correspondent equipment advertising the discovered *devices* and *services* over the UPnP network.

The *RobotIRC5* UPnP device is implemented in a software application that communicates with the robot controller via a TCP/IP based network. The robot controller runs a server application developed in RAPID (ABB 2005) on an independent task, similar to the one presented in (Pires *et al*, 2006). This UPnP device provides one *service*: *PickandPlaceService* (Fig. 3). This *service* has two different actions: one that allows picking all

identified pieces, and another that picks a single piece properly parameterized. It also includes an *evented* state variable (busy) that indicates the state of the robot, and publishes an event each time the robot finishes picking. This device uses the Intel C# UPnP stack.

The *Conveyor* UPnP device was also implemented as a software application that communicates with the conveyor commanding PLC through a serial port. Two services are also available (Fig. 4): The *HighLevelProg* service provides actions and variables with process related meanings. The *Maintenance\_Setup* service is more technology related, and is intended to be used during development or maintenance stages. This strategy of dividing process related and technology related services enhance the advantages of the SOA in the *HighLevelProg* service, without compromising some technology know-how needed for finding IO problems in the installation stage, for example. This device uses also the Intel C# UPnP stack.

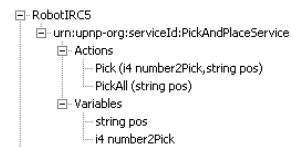


Fig. 3 UPnP device for the robot

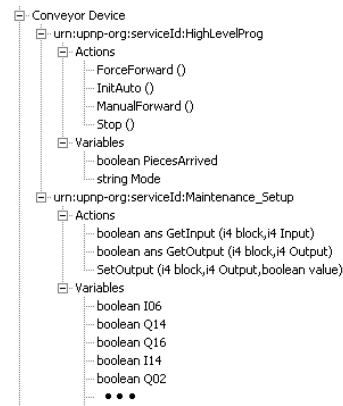


Fig. 4. Conveyor UPnP device

The *SmartCamera* UPnP device (Fig. 5) was implemented using a commercial USB web camera, and special purpose vision software developed using C++ and the Intel OpenCV vision libraries (OpenCV 2006).

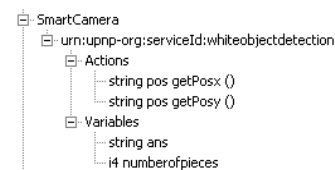


Fig. 5. UPnP device: SmartCamera.

The application determines the number and position of the pieces on the conveyor, and provides an UPnP action (getPos()) that returns these positions in a string format.

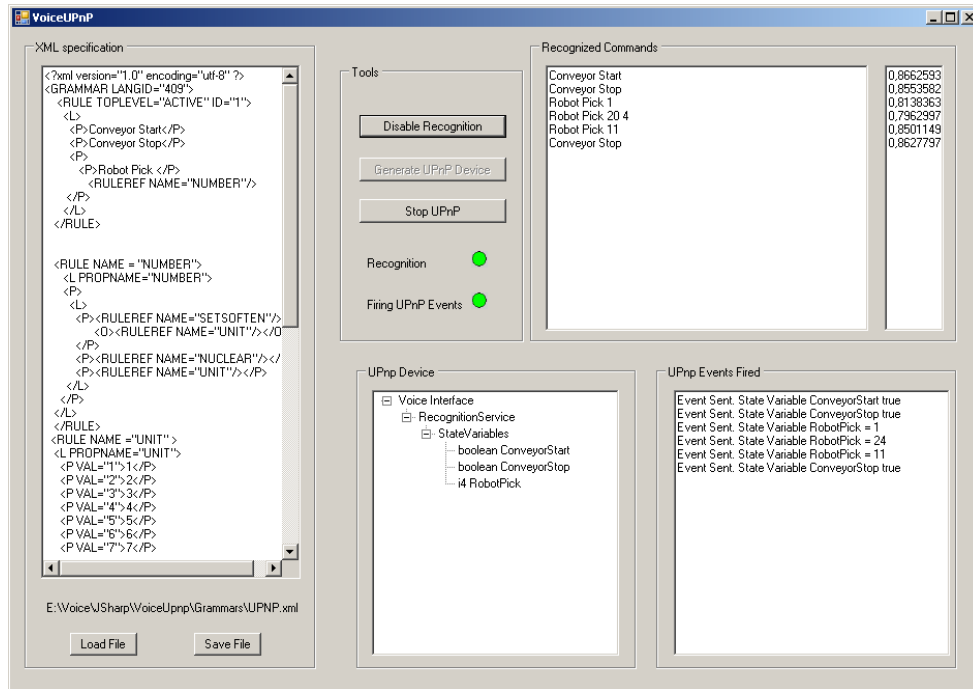


Fig. 6. UPnP device: Voice Interface.

Speech Recognition Systems (SRS) evolved significantly in the last couple of years. Actual SRS can even be used in industrial environment, see (Pires 2005). This type of technology can be seamlessly integrated in a SOA due to the fact that they are extensively event driven. To achieve this integration a software application was developed to allow the automatic pairing of voice recognition events with UPnP events (Fig. 6).

The SRS selected to use with this research was the Microsoft speech engine included with the Microsoft speech API 5.1 (Microsoft 2006b). This system includes an ASR (automatic speech recognition) engine and a TTS (text to speech) engine.

To create an UPnP device that can publish events correspondent to ASR events an application was developed (using C#, Fig. 6) that implements the following strategy:

- First the XML grammar was parsed and an XML-DOM tree document created.
- Then this tree was traversed and UPnP state variables were dynamically created and added to the *RecognitionService* of the voice interface device. All combinations implemented with grammar tags `<L></L>` (List) are listed, and a boolean state variable is created for each one of them. The name of the state variable is the recognised sentence without spaces. Nevertheless, if this traversal method goes through each rule reference, a very high number of variables would be created. To avoid these difficulties and to express the real mean of the recognized number, an integer state variable was associated with each of the recognitions that may contain a number. It is important to notice that the UPnP events are fired every time a new value is assigned to the state variable, even if the value is the same.

### 3.3 High-level services.

Grammars are used to define what the ASR should recognize. Each time a sequence defined in the grammar is recognized an event is fired by the SRS. The Microsoft SAPI allows three different ways for specifying grammars: included in the code (programmatic grammars), using XML files, or using CFG files. Since XML is a well accepted standard it was used to specify speech recognition grammars.

Grammars define a TopLevel Rule that includes all the necessary commands. From each of these commands it is possible to call other rules. In the example presented in Fig. 2, a rule (“NUMBER”) was created to support the recognition of numbers (0-99). This rule is composed by several secondary rules (UNIT, SETSOFTEN,...) that have properties associated

These properties allow the easy recovery of a value when a number is recognized, because they are sent as an argument of the delegate call when a recognition event occurs (Fig. 7).

```

recContext.Recognition+=new _ISpeechRecoContextEvents_RecognitionEventHandler(
    this.handleRecognition);
...
public void handleRecognition(int StreamNumber, System.Object StreamPosition,
    SpeechRecognitionType RecognitionType,
    ISpeechRecoResult Result)
{
    SpeechDisplayAttributes a = Result.PhraseInfo.GetDisplayAttributes(0, 1, false);
    float confidence = Result.PhraseInfo.Rule.EngineConfidence;
    int num=0;
    txtBoxRecoCmd.Text+=Result.PhraseInfo.GetText(0, -1, true);
    if (Result.PhraseInfo.Properties!=null)
    {
        if (Result.PhraseInfo.Properties.Count > 0)
        {
            foreach (ISpeechPhraseProperty p in Result.PhraseInfo.Properties)
            {
                num += (int) p.Value;
            }
        }
    }
}
...

```

Fig. 7. Recognition handling code: Voice Interface Example using delegates C#.

This application provides a very interesting approach to link the meaning of both dictated numbers and UPnP state variables. This approach could be extended to terms like *Conveyor* and *Robot* which could be associated with the respective devices, or even linked to ontology in robotics.

The *Cell Programmer Interface* (Fig. 8) is a software application developed to control the flow of high level tasks in a manufacturing cell. Basically, it's an UPnP *control point*, with some tools suitable to build a generic stack. This stack represents the control flow of process related tasks. In the left side of this interface a tree shows all UPnP devices founded on the network (notice the presence of a "stranger" gateway device). Clicking over them is possible to get additional information (access the presentation page, for example). Using the "arrow" button, actions or events are added to the stack. Furthermore, when running the resulting program and the program

counter is pointing to an event, it means that the program is "waiting" for that event to occur. Inversely, if the program counter is pointing to an action, it means that it is calling that action and waiting for the return. There is also the possibility of defining auxiliary variables to store values that can be used as arguments in later stack steps.

The simple case depicted in Fig. 8 is a pick-and-place case. The setup should wait for a speech recognition event that commands the conveyor to start. When the event occurs an action is called commanding the conveyor to enter the automatic mode. The next action is to obtain information about the number of pieces and respective position from the camera server.

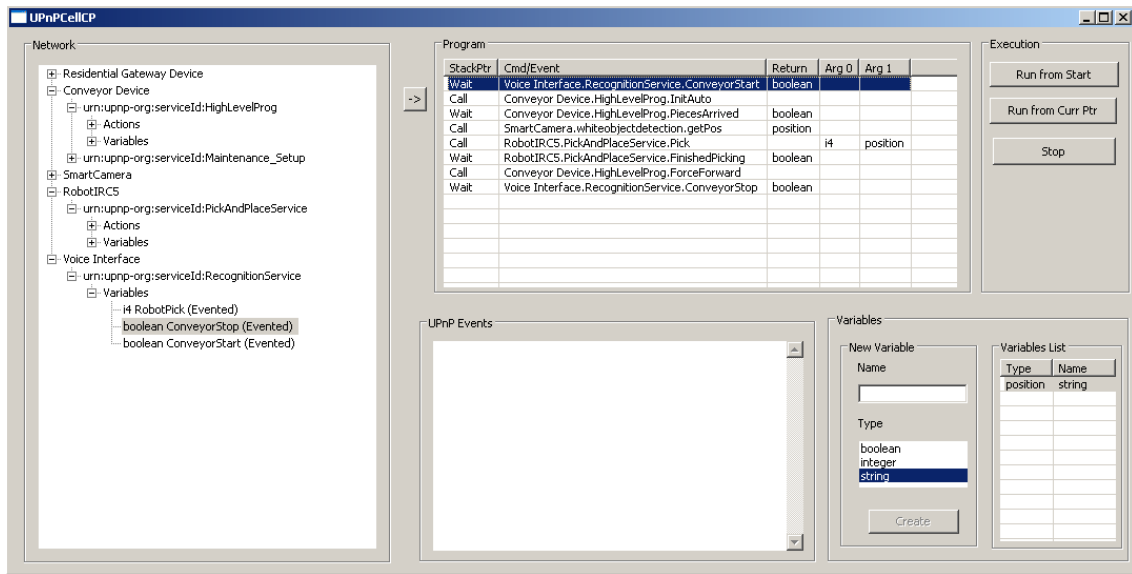


Fig. 8. UPnP control point: Cell Programmer Interface.

The obtained information is used to pick-and-place the pieces calling the robot *pick* service. When the last piece is removed the conveyor starts automatically, so in this simple example the next element of the stack is a speech recognition event that waits for the command to stop the conveyor.

## CONCLUSION

The main objective of this paper was to review some of the recently proposed technologies, and confront the most promising approach with experimental setups reflecting real applications, i.e., of using the selected SOA architecture to control a real manufacturing cell and evaluating the results. Furthermore, some novel concepts were introduced, like automatic UPnP generation from a speech XML grammar specification. This will be further developed in the near future. The proposed programming scheme based on SOA is less time consuming if compared with an object-oriented approach (Pires *et al* 2006). Planned developments include the implementation of built-in solutions for the most important cell components like robots, cameras, PLCs, intelligent sensors, etc. This approach enables to extend the plug-and-play concept, based on SOA, to plug-and-produce just by adding to the devices a set of services that correspond to particular cell functionalities. The manufacturing system must then be prepared to use the new services, with the necessary adaptations done automatically, to start or keep producing.

## ACKNOWLEDGMENTS

This work has been partially funded by the European Commission's Sixth Framework Program under grant no. 011838 as part of the Integrated Project SMERobot.

## REFERENCES

- Abb (2005): ABB IRC5 Documentation, ABB Flexible Automation, Merrit, 2005
- Ahn S. C., Kim J.H., Lim K., Ko H., Kwon Y and Kim H. (2005) UPnP Approach for Robot Middleware *P Proceedings of the 2005 IEEE International Conference on Robotics and Automation Barcelona, Spain, April 2005.*
- Bettstetter C. and R. Christoph, (2005) A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. *Proc. EUNICE Open European Summer School, Twente, Netherlands, Sept 13-15, 2000.*
- James, F. and H. Smit (2005a) Service Oriented Paradigms for Industrial Automation. In: *IEEE Transactions on Industrial Informatics, Vol. 1, no. 1 February 2005.*

- James, F. and H. Smit. (2005b) Service Oriented Device Communications using the Devices Profile for Web Services. *ACM International Conference Proceeding Series, Vol. 115, no. 1 December.*
- Jini (2006), The Community Resource for Jini technology: <http://www.jini.org>.
- Gou L., P. Luh, and Y. Kyoyax (1998). Holonic Manufacturing Scheduling: Architecture, Cooperation Mechanism, and Implementation. *Computer in Industrial 37*, 213-231,
- Microsoft (2006a), Microsoft Robotics Studio Available:<http://msdn.microsoft.com/robotics/>
- Microsoft (2006b), Speech Application Programming Interface (API) and SDK, Version 5.1, Microsoft Corporation, <http://www.microsoft.com/speech>
- OpenCV (2006). Available: <http://sourceforge.net/projects/opencvlibrary/>.
- Pires, J. N. (2005) Experiments on commanding an industrial robot using human voice. *Industrial Robot, an International Journal, Volume 32, number 6, Emerald.*
- Pires, J. N., Godinho T. and Araújo R. (2006). Controlo e Monitorização de Células Robotizadas Industriais revista *Robótica Abril 2006.*
- PnP-X (2006): Plug and Play Extensions for Windows Specification. Available: [www.microsoft.com/whdc/Rally/pnp-spec.msp](http://www.microsoft.com/whdc/Rally/pnp-spec.msp).
- Rekesh J.(1999), UPnP, Jini and Salutation A look at some popular coordination frameworks for future networked devices, California Software Labs, June 17.
- Schlimmer J., S. Chan, C. Kaler., T. Kuehnel, R. Regnier, B. Roe, D. Sather, H. Sekine, D. Walter, J. Weast, D. Whitehead, and D. Wright (2004) Devices Profile for Web Services: A Proposal for UPnP 2.0 Device Architecture. Available: <http://xml.coverpages.org/ni2004-05-04-a.html>.
- SIRENA Project (2005), Service Infrastructure for Real-time Networked applications, Eureka Initiative ITEA. Available: [www.sirena-itea.org](http://www.sirena-itea.org).
- Siemens (2000), PLC S7-200 System Manual, 2000
- UPnP forum (2004). Available: <http://www.upnp.org>.