INTECH
open science | open minds

OPEN ACCESS
ARTICLE

International Journal of Advanced Robotic Systems

Industrial Robot Programming and UPnP Services Orchestration for the Automation of Factories

Regular Paper

A. Valera*, J. Gomez-Moreno, A. Sánchez, C. Ricolfe-Viala, R. Zotovic and M. Vallés

Instituto de Automática e Informática Industrial, Universitat Politècnica de València, Valencia, Spain
* Corresponding author E-mail: giuprog@isa.upv.es

Received 8 May 2012; Accepted 7 Jun 2012

DOI: 10.5772/51373

Abstract The integration of equipment and other devices built into industrial robot cells with modern Ethernet interface technologies and low-cost mass produced devices (such as vision systems, laser scanners, force torque-sensors, PLCs and PDAs etc.) enables integrators to offer more powerful and smarter solutions. Nevertheless, the programming of all these devices efficiently requires very specific knowledge about them, such as their hardware architectures and specific programming languages as well as details about the system's low level communication protocols.

To address these issues, this paper describes and analyses the Plug-and-Play architecture. This is one of the most interesting service-oriented architectures (SOAs) available, which exhibits characteristics that are well adapted to industrial robotics cells. To validate their programming features and applicability, a test bed was specially designed. This provides a new graphical service orchestration which was implemented using Workflow Foundation 4 of .NET. The obtained results allowed us to verify that the use of integration schemes based on SOAs reduces the system integration time and is better adapted to industrial robotic cell system integrators.

Keywords industrial robotic cell programming, service-oriented architectures, robotic systems adapted to SMEs.

1. Introduction

The use of industrial robots in typical manufacturing systems is necessary. Therefore, they require a high level of integration with other devices and systems which are increasingly varied and powerful. The challenge is to respond to increasing demands in terms of flexibility and agility - e.g., it is common among small and medium enterprises (SMEs) that manufacture short batches of several types of products without stocks. Within system integration for such production, experiences and developments have resulted in a strong desire to improve efficiency and flexibility by combining the following three approaches:

- Integrating the different types of devices, such as conveyors, industrial robots, input-output devices and sensors, etc.
- Developing human-machine interface solutions that can take greater advantage of human operators while hiding from them the tricky details about how to have things done.

- Developing easy to use programming methods to enable system integrators to focus on system functionality and allow them to avoid the need of knowing about device dependant hardware and software details, such as the specific languages used and the protocols, etc.

Nevertheless, the desired level of flexibility for the integration of these systems has still not been achieved [20], [39].

The flexibility obtained today was basically achieved through specialization within known major application areas, more specifically, within specific equipment and its functionality. However, this is a limited solution since this approach aims to solve each system component's required functionality instead of dealing with the system's global functionality.

There are several ad hoc ways to approach the problem for systems that integrate these kinds of technology. Nevertheless the trend is to have a client-server software environment that enables system architects to distribute functionality as well as coordinate actions from a central client commanding application.

With the advent of the Internet, SOAs have emerged to increase the degree of decoupling between software elements [11]. A SOA relies on highly autonomous but interoperable systems. The definition of a service is ruled by a larger context; this means that all of the technological and interconnection details are hidden [25].

In the short term - as some authors and experiences point out [9] – it looks as though SOAs are here to stay [46], not only as applied to pure software development but also to other fields, such as robotics [34], information technologies [17] and even human task support [35].

At the device level - as predicted in [21] - SOAs are emerging as the main way to deal with the increasing amount of embedded devices present in our homes, offices and enterprises. The main idea behind the SOA paradigm is in providing system integrators with the possibility of offering their equipment and the services needed to operate them. This will enable the exchange of information between system elements through the Internet and/or local networks without the need for complex interconnections that are product dependant and without human intervention.

In the industrial sector, manufacturing can also be monitored remotely. This means that integration systems must be strong and safe enough in order to guarantee correct performance. In general terms, industrial robots in a production line are programmed without considering possible failures or faults. Failures are dealt with by a centralized system which synchronizes actions with robots. Nevertheless, if the system enters into a deadlock, a lot of time - and money - might be necessary in order to reset the process and synchronize the required devices so as to continue with production. With SOA-based systems, deadlocks can be detected – and, from there, avoided - as there is a continuous development of formal methods to check the deadlock freedom in SOA scenarios [26], [28] and [45].

With the aim of avoiding these problems, a service-oriented architecture can offer a very efficient solution since it will make device configuration and synchronization very simple. In addition to this, recent work lines have added real-time and fault tolerance capabilities to SOAs [10], [13], allowing them to undertake even time-critical scenarios.

The SOA paradigm has even more advantages when the production lines have many devices with a high capacity of processing. In order to foster and facilitate integration even more, so-called SOA logic devices are used. These logic devices are composed by a physical device and a group of the device features. In this way, a system can be implemented as several SOA logical devices, where only the interesting features for the process from each device are used.

All the services need to be made known in the SOA environment in order to implement it and they must also have their functions specified. In this way, all the logical devices can be used to create a program at the highest level, for instance, as with the welding tasks of an industrial robot.

This paper describes and analyses the development of industrial robotic cells based on industrial robots by means of the use of SOA. In order to do so, one of the most promising platforms will be displayed - the Universal Plug-and-Play architecture (UPnP) - focusing on the definition and orchestration of services.

For service planning and synchronization, a new orchestrator has been developed. It is a graphical orchestrator implemented using Workflow Foundation 4 of .NET. Therefore, the robot applications' configurations are very simple because the *drag-and-drop* enables to select and use the activities, primitives and flow control structures required for the application. The aim of this is to fill the gap left for a runtime application which is operator-oriented allowing for the easy re-programming of an industrial robotic cell, in addition to previous attempts at simplifying SOA development [14] that where more oriented towards back office development. As stated in [7], [16] and [44], visual programming allows

users (such as operators) with a lower programming background to be able to manage the system more intuitively.

This attempts to contribute to the objective of fully integrating the different layers of an enterprise under the same architecture [46], from business decision-making at the top, to the factory shop lower down.

The high complexity of such productive systems and the eruption of new approaches to auto-adaptive architectures [8], [29] have generated the belief that SOAs have an important role to play in the paradigm shift that will soon be arriving [9].

Developing good, service-based, operator-oriented applications will become fundamental for such high dimensional, rapidly changing systems.

2. Service-Oriented Architectures

2.1 Introduction

A great variety of mechatronic devices and components with a high capacity for calculation and autonomy can be found in industrial robotic cells nowadays, as mentioned earlier. Some examples are robots, artificial vision systems and programmable logic controllers (PLCs), etc. [12],[15]. These are centralized systems which use client-server point-to-point software based on the survey method. They are usually manually configured, and so this is a long, difficult and tedious task.

The industrial robotic cell should have modern network architectures and communication systems based on a publisher/subscriber pattern so as to avoid these problems and obtain better performance [31]. In this way, decentralized intelligent software with a nearly automatic configuration can be obtained. Some examples that prove this theory can be found in the bibliography, but this approach constitutes a new research area. At present, the research literature offers very little information about this approach. A summary including the main characteristics of SOAs and defining the kinds of device can be found in [6],[19]. For instance, the European project SIRENA [37] pointed out the advantages of using SOAs in industrial automation, while the primary objective of the SOCRADES [38] European project was to develop a design, execution and management platform for next-generation industrial automation systems, exploiting the Service-oriented architecture paradigm, both at the device level and at the application level.

In [30][42], the process of programming industrial robotic cells with SOAs is proven. Moreover, in [2],[3],[4] and [6] the use of SOAs is proposed as robot middleware.

Four of the most relevant and available approaches of SOAs are considered in this work: Jini [22],[33], Decentralized Software Services Protocol (DSSP) [27], Universal Plug-and-Play [33],[40] and Device Profile for Web Services (DPWS) [36].

Jini is an architecture proposed by Sun Microsystems based on Java. To make it independent from the platform, it also carries a large memory footprint, due to the presence of a virtual machine and extensive libraries, making memory resources necessary. Therefore, it is less appropriate for very small devices.

DSSP is a protocol which is based on the simple service object access protocol (SOAP) that defines a lightweight, REST-style service model and which also relies extensively on web technology. Paired with concurrency and coordination runtime (CCR), it constitutes the major parts of the Microsoft Robotics Studio (MSRS) platform.

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. XML formats are broadly used and accepted and provide modern data exchange mechanisms and communications.

The implementation of SOAs can be based on RPC (Remote Procedure Call), which provides an innovative approach and offers many possibilities to considerably improve their application in robots [5]. The design of middleware platforms for SOAs is very helpful in order to have interoperability and a flexible and efficient coupling, especially for applications working in a heterogeneous environment. Therefore, a SOA has to specify the functional and non-functional proprieties (scalability, flexibility and the quality of the service included). Table 1 shows the three kinds of software related to this type of architecture and the most representative examples.

The idea that lies behind the RPC is to make the distributed systems more accessible to programmers. The RPC provides developers with an interface with the communication code, which is the best way to simplify procedure calls. By using this concept, it is not necessary for the developer to write the network code. Thus, programmers do not need to be experts in developing network code in order to write complex systems distributed through any number of hosts in a network.

Levels	Examples
SOA	UPnP, DSSP
Middleware services (based on RPC)	DCOM, CORBA, RMI, SOAP
Transport protocol	TCP/IP, HTTP

Table 1. Service levels and examples

RPC	Protocol	Characteristics
CORBA	Inter-ORB GIOP	When it is obtained, CORBA enables a client-server direct communication. It allows very fast communication. It is not scalable.
DCOM (Windows)	ORPC	This requires several round-trips to activate and use the remote object. When the reference has been obtained, access to DCOM objects can be carried out directly from the client. It is not scalable.
RMI (Java)	JRMP	Works well, but only with the Java language. It is quite scalable.
SOAP (XML-RPC successor)	Any transport protocol	Nowadays, it has limited services. There is an overhead for extracting the SOAP envelope, parsing XML, creating appropriate objects and converting parameters. It is scalable.

Table 2. Remote Procedure Call variations

Many systems have tried to implement the concept of the RPC and so some of the most common varieties can be found, for instance, CORBA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model), Java RMI (Remote Method Invocation) and SOAP (Simple Object Access Protocol). Despite the fact that this software is used in different environments, they each have the same objective: to facilitate writing and maintaining the different applications. Table 2 shows the most-used procedures, the protocol name and the RPC's essential features.

CORBA is a RPC object-oriented version. CORBA – GIOP, to be precise - uses TCP/IP connections to transmit data. DCOM was the biggest CORBA competitor. Some years ago, these technologies' supporters considered them as an example of coding and service reuse on the Internet. However, if the client has a firewall or a very restrictive server which only enables HTTP connections, communication can become impossible.

Juric, et al. [24], analysed how RMI worked. Even though this technology worked better than the web services, the fitting/adjustment alternatives have limited services/features. In addition, the web services have to be used if it is not possible to open distributed application ports.

SOAP is a specification that enables communications between applications [47]. SOAP has two main objectives –simplicity and extensibility. To reach them, it omits features that are usually found in distributed systems within the messages' framework. In addition, it is an open standard specification based on open technologies, such as XML and HTTP, so it is versatile enough to enable different transport protocols. SOAP is lower than CORBA, RMI and DCOM. Nevertheless, SOAP correctly designs tunnels on HTTP in firewalls and proxies.

As can be found in [32], the integration cycle for the development of distributed applications in industrial robotic systems based on middleware services can be a long and tedious task. In this work, a robotic system that uses services based on SOA in a framework which is oriented towards distributed services was developed and analysed. In comparison with the existing work, the robotic system supports publisher/subscriber mechanisms and it can find services and devices in a precise way.

2.2 UPnP Architecture

This work notes the services and applications based on an UPnP architecture. This type of architecture has been chosen because it has been tested and validated for some time in different office environments. Therefore, there are a great variety of development tools available. In addition, [2],[3],[43] include some examples of the use of this architecture for the development of industrial robot cells.

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

The basic steps that compose UPnP networking are:

- Addressing. A device or a control point obtains a valid IP address.
- Discovery. A control point finds a device of interest.
- Description. A control point obtains service descriptions from devices.
- Control. A control point invokes actions on devices.
- Eventing. The services announce changes in their states.
- Presentation. A control point monitor and control device status using a HTML User Interface.

Usually the dynamic host configuration protocol (DHCP) is used to obtain the IP. Once devices are attached to the network and are properly addressed, the discovery step takes place. The devices advertise their services to control points and they in turn search for devices in the network. The description step enables 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 necessary information about the actions and state variables provided by a device. The control step consists on calls to 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 over the network. 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 - enable a user to control the device and/or view the device status.

3. UPnP Services Development for Applications Based on Industrial Robots

3.1 Introduction

In this section, some methods and tools used to develop UPnP services for industrial robot applications will be explained. Firstly, two tools that enable the services' devices' automatic generation are shown. Once the devices have been obtained, it is only necessary to program the services functionality that these devices have. This is done by means of the two tools that allow the programming of the industrial robots ABB and FANUC.

3.2 UPnP Service Development.

Although it is possible to find other solutions, in this work *Intel® Tools for UPnP™ Technology* is used for the development of the services [18]. Based on the Microsoft .NET Framework, these free software tools help hardware and software designers and programmers to develop, test and use UPnP services in a quick and simple way.

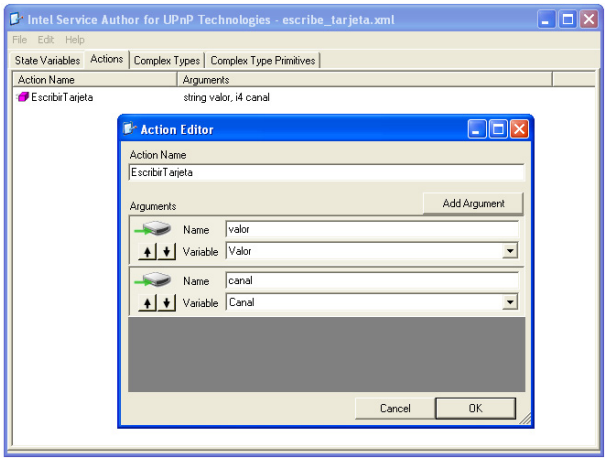

Figure 1. The Service Author tool - used for the automatic generation of the XML service description.

Although the package has 11 tools available, the application Service Author is mainly used in this work. This application generates the XML format automatically and it enables the creation and execution of the service. Therefore, state variables - which enable the input-output service arguments - can be specified. From these state variables, the actions linked to services can be defined.

Figure 1 displays the appearance of the Service Author application. This is a service that enables to digital/analogical conversion with a data acquisition card. To generate the service XML file, an action is defined (Write card). It has two input parameters: the value of the tension to be converted and the output required.

Once the service XML file is created, the only thing that remains to be done is to generate the code for the implementation of the device that offers the service. Therefore, this paper suggests a second, free software application: the Intel Device Builder for UPnP Technologies, from the library Intel® Digital Home Device Code Wizard [19]. This is an application which enables the generation of devices and control points for different platforms from XML descriptions of the services that are generated with the Service Author application.

In order to export the desired device with the services, Device Builder generates a portable C code automatically. It is only necessary to specify the desired target platform, the path where the code will be generated and the namespaces of the device.

Although codes can be generated for different platforms - such as Linux, Windows and PocketPC, etc. - in this paper, the solution is generated for the .NET Framework and C# is used. This is because of the programming environment used to develop the programming applications and to control the industrial robots. Figure 2 shows the appearance of the Device Builder tool.

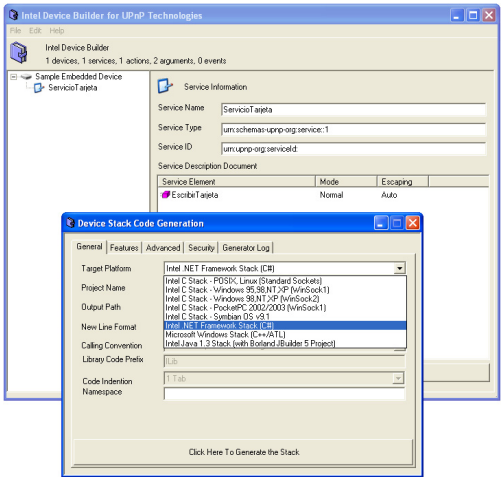

Figure 2. The Device Builder tool - used for the automatic generation of UPnP logical devices.

3.3 Industrial robots' programming environments.

After selecting the platform, Device Builder creates a .NET project with the necessary code for the service to be started or stopped or else to detect when this service is called, etc. The only thing to be done is to program the code that implements the required device functionality. It is only necessary for this to modify one of the .NET project files: namely, the DVImportedService.cs file.

Two environments are used for the development of the services' functionalities as based on industrial robots: Robot Application Builder and PCDK.

The Robot Application Builder (RAB) is a software development kit (SDK) designed for ABB robots. RAB uses the Microsoft® tool VisualStudio .NET in order to create personalized operator interfaces for both PC and FlexPendant. They enable the programmer to develop control applications for industrial robots, have access to their signals, modify the value of the variables in a program in RAPID [1] code, load programs, move the robot through RAPID programs, obtain the controller characteristics, have access to the file system of IRC5 and all the operations that can be done on an industrial robot.

RAB is based on a .NET class hierarchy, allowing for more comfortable and simpler programming (C#.Net o VB.Net), where all of these classes represent a robot entity. For example, there is a class (Mastership) that controls user access and checks their access and control rights to the robot and the IRC5 controller for actions that require a degree of security, for instance, the access to variables in a RAPID program. Thus, it can avoid the situation where two users have access to a variable at the same time.

PCDK is the second development environment used. It is a very powerful tool that enables a high degree of efficiency in the information communication and instructions between the PC and the controller of the FANUC robots, thus allowing the development of the applications in the languages of .NET, such as C# or VB.

PCDK provides robot servers so that access to the controller of the FANUC robots is allowed. This enables write and read KAREL variables, write and read numerical TPE registers, access to and the configuring of the input-output robot signals, load, saving and executing robot programs and the monitoring of their state, access to robot positions and the generating of movement paths, the monitoring alarms and use events, etc.

4. UPnP Service Orchestration

As has been explained in the previous part, the high level tasks of the robotic cell can be programmed using UPnP services. Once these services are developed, it is only necessary to plan and synchronize them.

In order to orchestrate the services, a graphical development environment was initially considered, similar to the Visual Programming Language included in the Microsoft Robotics Developer Studio (MSRS) [23].
An attempt was made to adapt control the UPnP services of industrial robots to the MSRS environment. However, this was not possible, since UPnP is not supported in MSRS nowadays.

It was also considered whether to orchestrate the services by the execution of State Chart XML (SCXML) schemes, since there are some tools that enable the converting of a Unified Modelling Language (UML) scheme into SCXML. Nevertheless, this possibility was also dismissed since it required off-line programming, which limited the UPnP features and did not have any real advantage in comparison to conventional programming.

Finally, a new orchestrator was built from a standing start. One of the new Visual Studio 2010 (VS2010) features is a powerful workflow designer that allows us to program graphically. Workflow Foundation 4 (WF) of .NET provides the relevant classes so one can customize the workflow designer for one's own needs and use it in one's own applications out of VS2010. In addition, WF provides a workflow execution engine, offering the possibility of evaluating Visual Basic expressions at runtime and enabling the creation of variables to be used in the program. Since these characteristics matched the desired requirements for the orchestrator, WF was the chosen technology to build it.

In order to develop the orchestrator, a software layer was programmed that enabled the importing of an action included in an UPnP service as an "activity" (a basic WF element). These activities include all the data processing needed for the input data to be considered as service input parameters and, in the same way, the returned values become output values of the WF activity. Figure 3 shows the orchestrator structure.

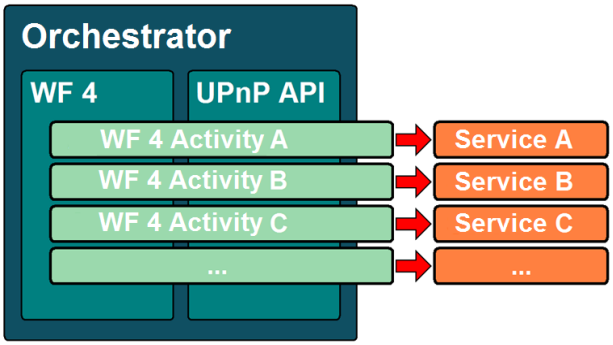

Figure 3. Orchestrator structure.

Figure 4 shows the developed graphical orchestrator. Four areas can be considered in it. In the left column, the activities available in the orchestrator are found, for instance, the workflow control primitives and UPnP services devices. In the second column, the area for the orchestration of activities can be found. Here, the program that will control the robotic cell can be composed. In addition, in the lower part of the same column, the required variables for the orchestration of services can be managed. The third column includes the properties of the selected WF activity. Here, the input-output parameters can be assigned. The right column offers the basic options for selecting files (to create a new program or to save the current one, etc). In addition, it has the "find" button, which enables the search for services on the Net. It also has the key "run", which allows the execution of the program.

Therefore, the use of the application is very simple. "Drag and Drop" enables us to place the activities and primitives that constitute the application in the working space. In addition, these primitives offer most of the flow control structures presented in any programming language ("while", "if-else", "for each"…), enabling it to encapsulate sequential or concurrent actions.

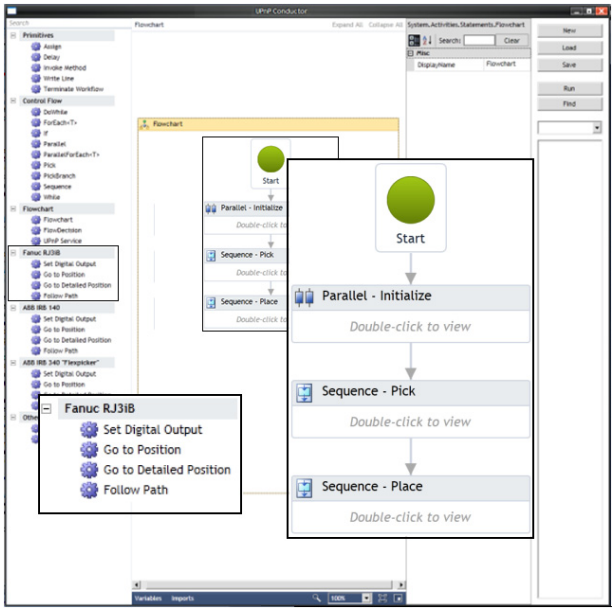

Figure 4. The developed graphical orchestrator.

Figure 4 shows an example of a simple application that has 3 stages: initiation, pick and place. At the first stage, different activities are launched in parallel to initialize the cell. Once the activities of the first stage are finished, the pick stage is run. At this stage, the robot moves until it reaches approximation, then descends to reach the picking point, closes the gripper and then rises. Figure 5 shows the encapsulate process of the second stage. Finally, in the last stage, the robot moves to the place where the piece will be left.

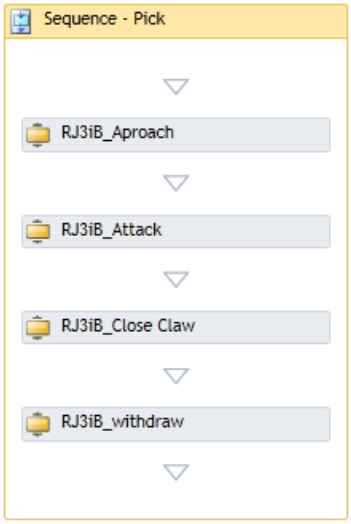

Figure 5. Activity sequence for the pick stage.

5. Development of an Experimental Test Bed for Robot Cells for the Automation of a Factory

This section presents an experimental test bed that enables the validation of the SOA as a general solution to the programming of industrial robot cells. The proposed application is an inspection and automatic pick-and-place station based on industrial robots. The demonstrator is basically composed of:

- A ABB IRB 140 robot, equipped with the IRC500 controller.
- A FANUC LR Mate 200Ib robot, equipped with the R-J3iB controller.
- Two conveyors controlled by variable frequency drives and equipped with presence sensors and encoders.
- Two web cameras located on the conveyors.

The station functionality can be summarized as follows: the first conveyor transports the pieces to be inspected. They are cylindrical pieces with similar dimensions but different colours. When the first sensor detects one piece, the web camera takes an image of it and it is analysed. If there is an object that does not pass the quality test - because of its colour - it is sent to the first robot (IRB140, equipped with a robot gripper) as a vector with the pieces of the centroids that have to be removed from the conveyor.

The correct pieces are automatically taken to the second conveyor, which has a web camera at the end of it. It is in charge of taking another image in order to detect the piece position. This information is sent to the FANUC LR Mate robot so that it takes the piece, one by one, in order to pack it.

Figure 6 shows the devices used for the development of the experimental test bed.

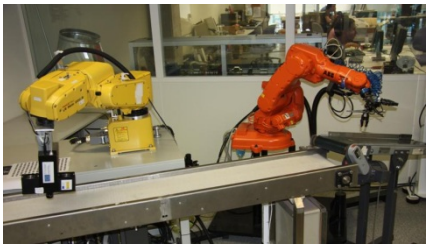

Figure 6. The experimental test bed.

It may be possible to deal with how to increase the versatility of the robot cell by incorporating features such as visual servoing, the synchronization of robots with the conveyors from the information given by their encoders, the verification of the quality control of pieces with more complex algorithms and the picking of pieces by controlling strength, etc. However, this paper focuses on the possibilities of SOA for the development of applications based on industrial robots.

For the development of the application based on SOA, five software applications, which match the five UPnP devices present (ABB_IRB_140, FANUC, Conveyor, Camera and System), have been developed. Figure 7 shows how the experimental test bed components have technological dependence.

Figure 7. Test bed technological dependences.

Since industrial robots do not support UPnP by default, it is necessary to develop an additional software layer to integrate it as standard UPnP devices and services. Therefore, ABB_IRB_140 devices were implemented in a software application that communicates with the ABB robot controller via a TCP/IP-based network. The robot controller runs a server application developed in RAPID. This UPnP device provides some services that accept one position, a configuration (position and orientation) or a positions' vector where the robot must go to, to read or write a RAPID variable content or to read a digital signal.

The FANUC UPnP device is similar to the device previously explained. It was developed to program and control the FANUC robot. For this device, services have been programmed (for instance, to send the robot a configuration or position vector where it must go to or in order to read a variable content. For this purpose, a KAREL application has also been developed. The application runs like a server, so that once the client is connected - the UPnP device - it waits to receive from the socket any information related to, for instance, the positions where it must go to.

Identically, the development of services for other manufacturers (such as KUKA and Motoman, etc.) will be possible since they support socket messaging but, for developing the test bed, ABB and FANUC were chosen because they were the only ones available.

The conveyor device was also implemented as a software application to control the conveyors. The whole of the five conveyors works by means of an alternating current engine which is activated with a frequency variator (Altivar-31, from Telemecanique). The conveyors are controlled by a PC equipped with a data acquisition card (PCI-1720, from Advantech). The conveyors can start working, stop and specify the direction and speed thanks to the digital output channels in the card. The card access programming has been done using C#.

The camera device was programmed using C# in order to control access to the commercial cameras (Logitech Webcam), which are in the inspection and packing conveyors. This device returns the pieces' numbers and positions on the conveyors.

The last device (System) is a service used to initiate and finish the robotic cell. It is also in charge of loading and booting the programs that are run in the industrial robot. In addition, it also has a service enabling the ending of the system.

The additional effort made in programming the necessary middleware for all the devices deployed for this demo consisted of: developing the UPnP Service (which is quite easy thanks to the automated tools provided by Intel); setting up the communications with the UPnP service (which is as trivial as coding socket messaging and developing the WF activity to manage the service, which only takes a few lines of code). Such effort is only necessary once per device.

Once the development process is finished, the orchestration of the final program logic was very simple and intuitive.

The main advantages of this new way of working as observed during the robot applications' development are scalability and device reusability. Since the only centralized part of the system is the system logic, adding new elements to the robotic cell is straightforward insofar as they can communicate through a TCP/IP-based network.

The development can be started by solving a small part of the problem (for example, the picking) and then connect new devices to the network to build a more complex solution. Regarding the reuse of devices, the development of a different application for this set of devices will only require changes in the service orchestration. This is particularly profitable for rapidly changing manufacturing environments.

Another finding made during this task - which turned out to be a core contribution of the graphical orchestrator - is to finally bring the business logic to the workshop. This is achieved by solving two problems:

First, drag and drop - one of the most widely used (and, therefore, expected) - input methods are used. By this, robotic cell programming is feasible for people with only minimal background knowledge in robotics and automation. It is simpler to learn to operate the orchestrator and plug devices into the network than to learn the different controllers' programming languages and the proper way to communicate between them (without even considering other difficulties such as using robots from different manufacturers).

Second, the visual programming paradigm and the workflow encapsulation capabilities of WF allow the operator to have a clear overview of the productive process.

Due to the complexity of this test bed and the SOAs' advantages, a traditional approach for this industrial robotic cell was not implemented. Therefore, data couldn't be obtained in order to accurately compare the two solutions. However, the obtained results match with the conclusions of [41]. In this work, using UPnP services, the authors obtained a reduction in programming time of 40% compared with an alternative ad hoc robotic cell solution. In addition, another evaluation was made based on the feedback of robot programmers, R&D engineers, former system integrators and engineering students. The results were satisfactory since the simplicity and the expressiveness of the SOA platform allowed their development intuitively. Moreover, various users claimed that the discovery features of the SOA allowed for a better reconfiguration experience.

6. Conclusions

The main aim of this paper was to describe one of the many SOAs recently proposed and to verify its use in real industrial workshop applications, such as the programming of robotic cells. To achieve this, an experimental test bed was designed in order to implement one of the most promising SOA technologies: UPnP

Focusing on industrial automation and - more specifically - on the programming of industrial robotic cells, UPnP can enable operators to perform high-level programming tasks. Therefore, engineers can concentrate only on low-level programming tasks, with the added advantage of being able to program them using any language. Moreover, since it is based on standard technology, it allows them to spend less time and attention on complex interconnection tasks.

A new orchestrator has also been developed in this work. It is a graphical orchestrator based on Workflow Foundation 4 technology. It is intuitive and simple to use. For these reasons, SOA allows the programming this type of industrial cell by simply specifying the logic of the system. In addition, it enables one to save time when configuring and programming industrial applications in the workshop, even for those who are not considered to be experts in the subject. In addition, developed solutions can be found and used for the most important cell components - for instance, robots, cameras, PLCs and intelligent sensors, etc.

7. Acknowledgments

The authors wish to express their gratitude to the Plan Nacional de I+D (FEDER-CICYT, Spanish Government) and to the Universitat Politècnica de Valencia (Spain) for the financing of this work, which was made under the research projects DPI2010-20814-C02-02, DPI2011-28507-C02-01 and PAID/2011/039. In addition, they also want to acknowledge the assistance of Elena Ruíz Gómez for her help in the translation of the article.

8. References

[1] ABB (2005). "ABB IRC5 Documentation, ABB Flexible Automation", Merrit.

[2] Ahn S.C., Kim J.H., Lim K.W., Ko H., Kwon Y.M. and Kim H.G. (2005). "UPnP Approach for Robot Middleware," *Proc. of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona (Spain), pp. 1959-1963.

[3] Ahn S.C., Lee J.W., Lim K.W., Ko H., Kwon Y.M. and Kim H.G. (2006). "UPnP SDK for Robot Development", *Proc. of the SICE-ICASE Int. Joint Conference*, Busan (Corea), pp. 363-368.

[4] Ahn S.C., Lee J.W., Lim K.W., Ko H., Kwon Y.M. and Kim H.G. (2006). "Requirements to UPnP for Robot Middleware", *Proc. of the 2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Beijing (China), pp. 4716-4721.

[5] Al-Jaroodi J., Mohamed N. and Aziz J. (2010). "Service Oriented Middleware: Trends and Challenges", *Information Technology: New Generations, Third Int. Conf. on, 2010 Seventh International Conference on Information Technology*, pp. 974-979.

[6] Bettstetter C. and Christoph R. (2000). "A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol", *Sixth EUNICE Open European Summer School*, Twente (Netherlands).

[7] Kelleher C. and Pausch R. (2005). "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers", ACM Computing Surveys, Vol. 37, No. 2, pp. 83-137.

[8] Candido G., Colombo A.W., Barata J. and Jammes F. (2011). "Service-Oriented Infrastructure to Support the Deployment of Evolvable Production Systems", *IEEE Transactions on Industrial Informatics*, Vol. 7, No. 4, pp. 759-767.

[9] Crnkovic I., Stafford J. and Szyperski C. (2011). "Software Components beyond Programming: From Routines to Services", *IEEE Software*, Vol. 28, No. 3, pp. 22-26.

[10] Cucinotta T., Mancina A., Anastasi G.F., Lipari G., Mangeruca L., Checcozzo R. and Rusina F. (2009). "A Real-Time Service-Oriented Architecture for Industrial Automation", *IEEE Transactions on Industrial Informatics*, Vol.5, No. 3, pp. 267-277.

[11] Delamer I.M. and Lastra J.L.M. (2006). "Service-Oriented Architecture for Distributed Publish/Subscribe Middleware in Electronics Production", *IEEE Transactions on Industrial Informatics*, Vol. 2, No. 4, pp. 281-294.

[12] El-KebbeSalaheddine D. A. (2000). "Towards a Manufacturing System under Hard Real-Time Constraints", *Informatik 2000*: 30. Jahrestagung der Gesellschaftf¨urInformatik, Berlin.

[13] Estevez-Ayres I., Basanta-Val P., Garcia-Valls M., Fisteus J.A. and Almeida L. (2009). "QoS-Aware Real-Time Composition Algorithms for Service-Based Applications", *IEEE Transactions on Industrial Informatics*, Vol. 5, No. 3, pp. 278-288.

[14] Foster H., Uchitel S., Magee J. and Kramer J. (2010). "An Integrated Workbench for Model-Based Engineering of Service Compositions", *IEEE Transactions on Services Computing*, Vol. 3, No. 2, pp. 131-144.

[15] Gou L., Luh P. and Kyoyax Y. (1997). "Holonic Manufacturing Scheduling: Architecture, Cooperation Mechanism, and Implementation", *IEEE/ASME International Conf. on Advanced Intelligent Mechatronics*, Vol. 37, pp. 213-231.

[16] Green T.R.G., Petre M. and Bellamy R.K.E. (1991). "Comprehensibility of visual and textual programs: A test of superlativism against the 'match-mismatch' conjecture", *Proceedings of the Empirical Studies of Programmers: Fourth Workshop (ESP4)*, pp. 121-146.

[17] Hong-Mei C., Kazman R. and Perry O. (2010). "From Software Architecture Analysis to Service Engineering: An Empirical Study of Methodology Development for Enterprise SOA Implementation". *IEEE Transactions on Services Computing*, Vol. 3, No. 2, pp. 145-160.

[18] Intel. http://software.intel.com/en-us/articles/intel-software-for-upnp-technology-download-tools, 2009.

[19] Intel.http://intel-r-digital-home-device-code-wizard.software.informer.com, 2010.

[20] Jaejoon L. and Kotonya G. (2010). "Combining Service-Orientation with Product Line Engineering", *IEEE Software*, Vol. 27, No. 3, pp. 35-41.

[21] Jammes F. and Smit H. (2005). "Service-oriented paradigms in industrial automation", *IEEE Transactions on Industrial Informatics*, Vol. 1, No. 1, pp. 62- 70.

[22] Jini, The Community Resource for Jini technology: http:www.jini.org, 2007.

[23] Microsoft. Microsoft Robotics Developer Studio. Available online: http://www.microsoft.com/robotics/, 2010.

[24] Juric M.B., Kezmah B., Hericko M., Rozman I. and Vezocnik I. (2004). "Java RMI, RMI tunneling and Web services comparison and performance analysis", *ACM Sigplan Notices*, Vol. 39, No. 5, pp. 58-65.

[25] Lewis G., Morris E., Simanta S. and Smith D. (2011). "Service Orientation and Systems of Systems", *IEEE Software*, Vol. 28, No. 1, pp. 58-63.

[26] Lou L., Tang F., You I., Guo M., Shen Y. and Li L. (2011). "An Effective Deadlock Prevention Mechanism for Distributed Transaction Management", Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on. pp. 120-127.

[27] Nielsen H. and Chrysanthakopoulos G. (2007). "*Decentralized Software Services protocol*" – DSSP/1.0 July.

[28] Martin M., Grounds N.G., Antonio J.K., Crawford K. and Madden J. (2010). "Banker's Deadlock Avoidance Algorithm for Distributed Service-Oriented Architectures", PDPTACSREA Press, pp. 43-50.

[29] Menasce D., Gomaa H., Malek S., Sousa J. (2011). "SASSY: A Framework for Self-Architecting Service-Oriented Systems", *IEEE Software*, Vol. 28, No. 6, pp. 78-85.

[30] Nilsson K. and Bengel M. (2008). "Plug-and-Produce technologies real-time aspects-Service Oriented Architectures for SME robots and Plug-and-Produce. Informatics in Control", *Automation and Robotics: Selected Papers from the International Conference on Informatics in Control, Automation and Robotics 2008*.

[31] Pires J.N. (2007). "Industrial Robots Programming Building Applications for the Factories of the Future", Springer.

[32] Qiang Z., Danyan C. and Xiaohong C. (2009). "An Approach to Constructing Event-Driven Virtual Enterprise Based on SOA", *Int. Forum on Computer Science-Technology and Applications*, pp. 443-446.

[33] Rekesh J. (1999). "UPnP, Jini and Salutation A look at some popular coordination frameworks for future networked devices". California Software Labs.

[34] Remy S.L. and Blake M.B. (2011). "Distributed Service-Oriented Robotics", *IEEE Internet Computing*, Vol. 15, No. 2, pp.70-74.

[35] Sasa A., Juric M.B. and Krisper M. (2008). "Service-Oriented Framework for Human Task Support and Automation", *IEEE Transactions on Industrial Informatics*, Vol .4, No .4, pp. 292-302.

[36] Schlimmer J., Chan S., Kaler C., Kuehnel T., Regnier R., Roe B., Sather D., Sekine H., Walter D., Weast J., Whitehead D. and Wright D. (2004). "Devices Profile for Web Services: A Proposal for UPnP 2.0 Device Architecture", available: http://xml.coverpages.org/ni2004-05-04-a.html.

[37] SIRENA Project (2005). "Service Infrastructure for Real-time Networked applications", *Eureka Initiative ITEA*, www.sirena-itea.org.sadasd.

[38] SOCRADES Project (2006), "Service Oriented Cross-layer Infrastructure for Distributed Smart Embedded Devices", 6th Framework Programme, European Commission (2006-2009). www.http://www.socrades.eu.

[39] Unver H.H. (2011). "System Architectures Enabling Reconfigurable Laboratory-Automation Systems", *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 41, No. 6, pp. 909-922.

[40] UPnP forum. Available online: http://www.upnp.org.

[41] Veiga G. (2009). "On the orchestration of operations in flexible manufacturing", Doctoral dissertation. University of Coimbra (Portugal).

[42] Veiga G., Pires J.N. and Nilsson K. (2009). "Experiments with Service-Oriented Architectures for Industrial Robotics Cells Programming", *Robotics and Computer-Integrated Manufacturing*, Vol. 25, No. 4-5, pp. 746-755.

[43] Veiga G., Pires J.N. and Nilsson K. (2007). "On the use of Service Oriented Software Platforms for Industrial Robotic Cells", *IFAC Int. Workshop Intelligent Manufacturing Systems*, Alicante (Spain).

[44] Whitley, K.N.(1997). "Visual Programming Languages and the Empirical Evidence For and Against", *Journal of Visual Languages and Computing*, Vol. 8, Issue 1, pp. 109-142.

[45] Wolf K., Stahl C., Ott J. and Danitz R. (2009). "Verifying Deadlock- and Livelock Freedom in an SOA Scenario", *Application of Concurrency to System Design*, 2009. ACSD '09. Ninth International Conference on. pp. 168-177.

[46] Xu L.D. (2011). "Enterprise Systems: State-of-the-Art and Future Trends", *IEEE Transactions on Industrial Informatics*, Vol. 7, No. 4, pp. 630-640.

[47] Yilmaz G. (2006). "Comparison of Soap based Technologies: .Net Remoting and Asp.Net Web Services", *Journal of Aeronautics and Space Technologies*, Vol. 2, No. 4, pp. 23-28.