Robotics and Computer-Integrated Manufacturing ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

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

Experiments with service-oriented architectures for industrial robotic cells programming

G. Veiga [a], J.N. Pires [a,*], K. Nilsson [b]

[a] Mechanical Engineering Department, University of Coimbra, Polo II Campus, 3030-788 Coimbra, Portugal
[b] Computer Science Department, Lund University, Sweden

ARTICLE INFO

Article history:
Received 17 February 2008
Received in revised form
18 July 2008
Accepted 16 September 2008

Keywords:
Industrial robotic cell programming
Service-oriented architectures
Robotic systems adapted to SMEs

ABSTRACT

Integration of equipment in industrial robot cells is to an increasing part involved with interfacing modern Ethernet technologies and low-cost mass produced devices, such as vision systems, laser cameras, force–torque sensors, soft-PLCs, digital pens, pocket-PCs, etc. This scenario enables integrators to offer powerful and smarter solutions, more adapted to small and medium enterprises (SMEs), capable of integrating process knowledge and interface better with humans. Nevertheless, programming all these devices efficiently requires too much specific knowledge about the devices, their hardware architectures and specific programming languages, details about system communication low-level protocols, and other tricky details at the system level. To address these issues, this paper describes and analyses two of the most interesting service-oriented architectures (SOA) available, which exhibit characteristics that are well adapted to industrial robotics cells. To compare, discuss and evaluate their programming features and applicability a test bed was specially designed, and the two SOA are fully implemented to program the test bed. Special focus is given to the way services are specified and to the orchestration tools used to manage system logic. The obtained results show clearly that using integrations schemes based on SOA reduces system integration time and are more adapted to industrial robotic cell system integrators.

1. Introduction

The use of industrial robots in typical manufacturing systems requires integration of several devices in the task of interfacing with other systems and human operators. The challenge is to respond to increasing demands in terms of flexibility and agility, like it is common in 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 several types of input–output devices like vision systems, user interface devices, force–torque sensors, etc.; developing human–machine interface solutions that can take more advantage from the human operators although hiding from them the tricky details about how to have things done; developing programming facilities to allow system integrators to focus on system functionality avoiding the specific languages, device-dependent hardware and software details, communication specific information, etc.

The flexibility obtained today was basically achieved through specialization within known major application areas, which today are supported by highly dedicated languages and environments to handle the devices functionality and the system integration. Since system programmers are not necessarily general programming language experts, programming a manufacturing cell is therefore a task for a skilled engineering team, as done in large enterprises.

For SMEs, there are several ad-hoc ways to approach the problem. Nevertheless the trend is to have a client–server software environment that not only enables system architects to distribute functionality, but also to coordinate actions from a central client commanding application. This assumes availability of a remote procedure calling mechanism, or several, and a method of packaging functionality hiding from the user the complexity inherent to the process of communicating and handling information. Supporting a human ("natural") way of commanding a task goes beyond simple sequencing of commands, so we are not talking about software components that simply hold a collection of methods and data structures. Instead, software blocks need to handle complex tasks that are parameterized in terms of the production at hand.

With the advent of internet, service-oriented architectures (SOA) emerged to increase the degree of decoupling between software elements. A SOA relies on highly autonomous but

* Corresponding author. Tel.: +351 239 790700; fax: +351 239 790701.
E-mail address: norberto@robotics.dem.uc.pt (J.N. Pires).

0736-5845/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.rcim.2008.09.001

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 business (or process) related and less technological related. SOA gave software engineers time to think more on the business logic and less on the interconnection details. At the device level service-oriented architectures are emerging as the way to deal with the increasing amount of embedded devices present in our homes and offices.

In manufacturing, the required (but to be hidden for the user) complexity together with the presence of many high-performance processing devices makes the concept of service-oriented architectures particularly suitable to use. In fact, it leads to the idea that each programming block (i.e., not only physical devices) should be considered as a potential device (SOA device style) that offers programming or setup services. Those programming services should also be advertised, like with a normal device, inside the SOA environment, and any device offering execution services should signalize availability to the programming module that *activates* and uses its functionality. All the *activated* programming modules could then be used to constitute a working program. Existing programs, from user databases or from actually connected robots/devices, will be ready for execution if all the contained programming modules are activated. It can still be acceptable to have an activated program that can lead to a blocking situation. In that case the user should provide the required event handling. These programming modules/services as can be seen has the main foundations of a high level programming (HLP) framework for industrial robots.

In the following, to confront the general SOA idea with actual manufacturing needs, two of the most promising SOA platforms are analyzed and experimentally evaluated. Special attention is given to the definition of services and to the development/analysis of high-level orchestration programs. The focus in this last point is justified by the need to cope with the ease of use of the programming languages/environments of the cell subsystems. By evaluating different architecture styles and analyzing the applicability of the resulting systems, the aim is to contribute to the development of future plug-and-produce solutions for SME manufacturing.

2. Preliminaries

Since industrial cell components are getting more and more autonomous the automation technology should integrate modern networking architectures with event-driven and publish–subscribe communication. In the following, a few alternatives will be analyzed and briefly discussed.

2.1. Components and interconnectivity

Considering a *holonic* cell structure [1,2], with *holons* composed by automation devices, like an industrial robot or a vision system, one can classify as uncommon the need to have real-time in the communication framework. Although in some special but important cases, such as visual servoing and conveyor tracking involving feedback loops via several interconnected devices, systems need to be able to accomplish shop–floor deterministic traffic, it can be claimed that those cases constitute a smaller part of the integration problem. The majority of the component connections can instead be described in terms of coarse-grained services, with synchronous calls for setup and asynchronous events for operation. Additionally, there are techniques for real-time communication that operate along the same principles, using user datagram protocol (UDP) for the real-time events [3]. Hence, real-time extensions appear to be possible, but are outside the scope of this paper.

2.2. Safety and predictability

Safety is getting increasingly important for robotic work-cells, with the intense utilization of 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 simplify the scenario: safety sensors and controllers are configured separately, based on special hardware and certification procedures; safe robot work-cells are not mission critical (as an airplane control system). Therefore, occasional failure can be acceptable if it can be detected and handled (by stopping or performing another task). Thereby, we can simplify the (hopefully fast) development of flexible manufacturing systems, avoiding some of the issues of X-by-wire and similar systems for vehicle technologies [4].

2.3. Architecture

The SIRENA project (Service Infrastructure for Real-time Embedded Networked Applications) [5] pointed out the advantages of using SOA in industrial automation (Table 1). Both Ahn et al. [6] and Nielsen et al. [7] proposed using a SOA for the device level as robot middleware of an industrial mobile platform.

Industrial applicability also calls for support of dumb (wired IO for example) and legacy devices. This subject was addressed with success by James et al. [8,9] using specially designed gateway devices, and is now featured by the Microsoft Robotics Studio (MSRS) [10].

There are many SOA proposed for the device level and fairly nice revisions have been written that resume their basic features [11,12]. For the actual implementation, a SOA is in practice based on a middleware platform, which can be considered a lower level architecture. Such platform should include suitable mechanisms to support the SOA main characteristics for the device level: addressing, discovery, description, control and event handling (eventing).

3. Approach

In this work four of the most relevant and available approaches of SOA are considered: Jini [13], universal plug-n-play (UPnP) [14], decentralized service structure protocol (DSSP) [7] and device profile for web services (DPWS) [15].

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

UPnP and DPWS rely extensively on standard network protocols such as transport communication protocol (TCP/IP),

Table 1
Effects on the use of service oriented architectures in automation [1]

	Today	Near future
System	Centralised, large, intelligent controllers, dumb devices	Decentralised intelligent and autonomous devices
Communications	Polling client–server point-to-point	Event-driven, publish–subscribe, peer-to-peer
Setup	Long and difficult, manual programming tedious debugging	No programming, plug and play, context-aware configuration

UDP, hypertext transfer protocol (HTTP), simple object access protocol (SOAP), extendable markup language (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. Their style is close to the one defined in the business world with generic the pair composed by webservices description language (WSDL) and SOAP.

Although similar in many aspects, the UPnP and DPWS architectures use different languages for device description and different protocols for discovery and event notification. A proposal has been made to the UPnP foundation [15] for a convergence between the two approaches 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* [16].

DSSP is a simple SOAP-based protocol that defines a light-weight, representational state transfer (REST)-style service model [7] that also relies extensively on web technology. Paired with concurrency and coordination runtime (CCR) it constitutes the major parts of the MSRS platform.

3.1. *Selection of platform*

From the above discussion it is clear that UPnP and DSSP should be implemented and evaluated. UPnP and DPWS are very similar technologies, which mean that concepts and design styles can be easily ported between each other. On the other hand, for the UPnP case there are more development tools available and the past work with UPnP industrial test bed [17] is a valuable head-start.

Even considering that all these architectures are service-oriented, substantial differences between them may exist. This is the case of DSSP in comparison with the UPnP/DPWS pair referred above. The first is a *RESTful* architecture which means that it relies on a limited amount of verbs (limited number of actions) and unlimited number of nouns. The second follows the XML–remote procedure calls (RPC) SOAP general guideline and resembles many of the WS-* technology guidelines. Hence, the DSSP and UPnP architectural styles are quite different.

3.1.1. *UPnP*

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 (variable with events associated) (Table 2).

Addressing occurs when a device or a control point obtains a valid IP address. Normally the dynamic host configuration protocol (DHCP) is used; otherwise the device or control point uses de auto-IP mechanism.

Table 2
Basic steps that compose UPnP networking

Steps	Today
Addressing	Control point and device get an address
Discovery	Control point finds device of interest
Description	Control point obtains service descriptions from devices
Control	Control point invokes actions on devices
Eventing	Services announce changes in their state
Presentation	Control point monitor and control device status using a HTML user interface

Discovery takes place once devices are attached to the network and are properly addressed. 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 necessary 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 (modeled in their state variables) changes, the service publishes updates by sending messages over 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.

3.1.2. *DSSP*

The application model defined by DSSP results from the REST model by exposing services through their state and a uniform set of operations over that state.

DSSP services are fine-grained entities that can be created, manipulated and destroyed repeatedly with DSSP operations, and their orchestration forms a DSSP application. A DSSP service consists of:

- Identity—global unique reference of the service.
- Behaviour—definition of the service functionality.
- Service state—current state of the service.
- Service context—relationships the service has to other services.

The state of a service is a representation of the service at any given point in time. Representing a motor can may consist of rpm, temperature and fuel consumption. The behaviour of a service (the contract) is the combination of the content model describing the state and the message exchanges that a service defines for communicating with other services. The behaviour of a service is identified by a globally unique URI known as the contract identifier.

DSSP provides a uniform model for creating, deleting, manipulating, subscribing and orchestrating services independent of the semantics of those services. DSSP defines a set of state-oriented message operations that provide support for structured data retrieval, manipulation and event notification. DSSP operations are an extension of the HTTP application model, which provides support for structured data manipulation, event notification and partner management.

An application is a composition of loosely coupled services that through orchestration can be harnessed to achieve a desired task.

DSSP achieves this by separating state from behaviour, allowing services to expose their state and hide their behaviour. The behaviour of the services is described by contracts (that have a specific URI), and determines how it can compose with other services. Such composition is called partnering.

To overcome some limitations of the traditional web-based architecture (REST) when applied to the device level, the HTTP/1.1 application model was extended with structured data manipulation, event notification and partner management between services.

4. Experiments

Given the selected architectures/platforms the following objectives are persued in the rest of this paper:

1. Validate SOA as a general solution for programming industrial robotic cells.
2. Test concepts that support the service definition.
3. Develop general software to program industrial robotic cells.
4. Evaluate different SOA styles for industrial robotic cell programming.
5. Test strategies for the automatic generation of UPnP devices.

4.1. Test-bed description

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

Basically, the conveyor transports sample pieces over the machine vision system (Fig. 1), which calculates the number pieces and the correspondent position. The results are sent to the robot controller for the pick-and-place operation. A detailed description of this setup is available in [22], where an alternative solution based on a general TCP/IP client–server application was used.

Since only the robot controller has built-in support for sockets communications, earlier work [22] used several personal computer (PC)-based applications to distribute services over the network. Two different clients were developed to operate the cell: a PC-based graphical human–machine interface (GHMI) and a personal digital assistant (PDA) interface. There was also a speech recognition interface.

4.2. UPnP

The architecture proposed in this paper replaces the client–server 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.

Since both the industrial components of the system (PLC and robot) do not have native UPnP support, it was necessary to develop an extra software layer to integrate these devices, advertising in this way the discovered devices and services over the UPnP network.

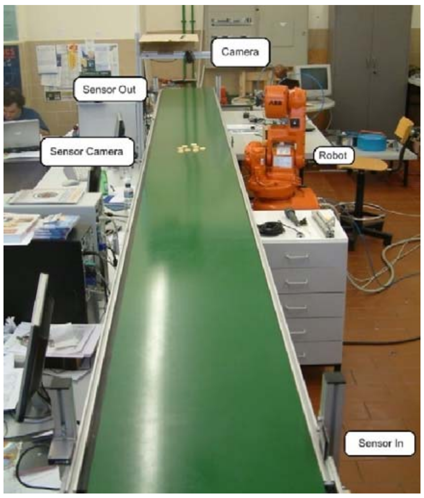


Fig. 1. Experimental setup at the laboratory.

The *RobotIRC5 UPnP* device was 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 [19] as an independent task, similar to the one presented in [22]. 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 [23]. Two services are also available (Fig. 4): the *HighLevel Prog* 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.

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 [20].

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.

Speech recognition systems (SRS) evolved significantly in the last couple of years. Actual SRS can even be used in industrial environment (see [21]). 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 Application Protocol Interface MSAPI 5.1 [18]. This system includes an automatic speech recognition (ASR) engine and a text-to-speech (TTS) engine.

To create an UPnP device that can publish events corresponding 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–document object model (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⟩⟨ ⟩ (List) are listed, and a Boolean state variable is created for each one of them. The name of the state variable is the recognized 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.

Grammars are used to define what the ASR should recognize. Each time a sequence defined in the grammar is recognized an

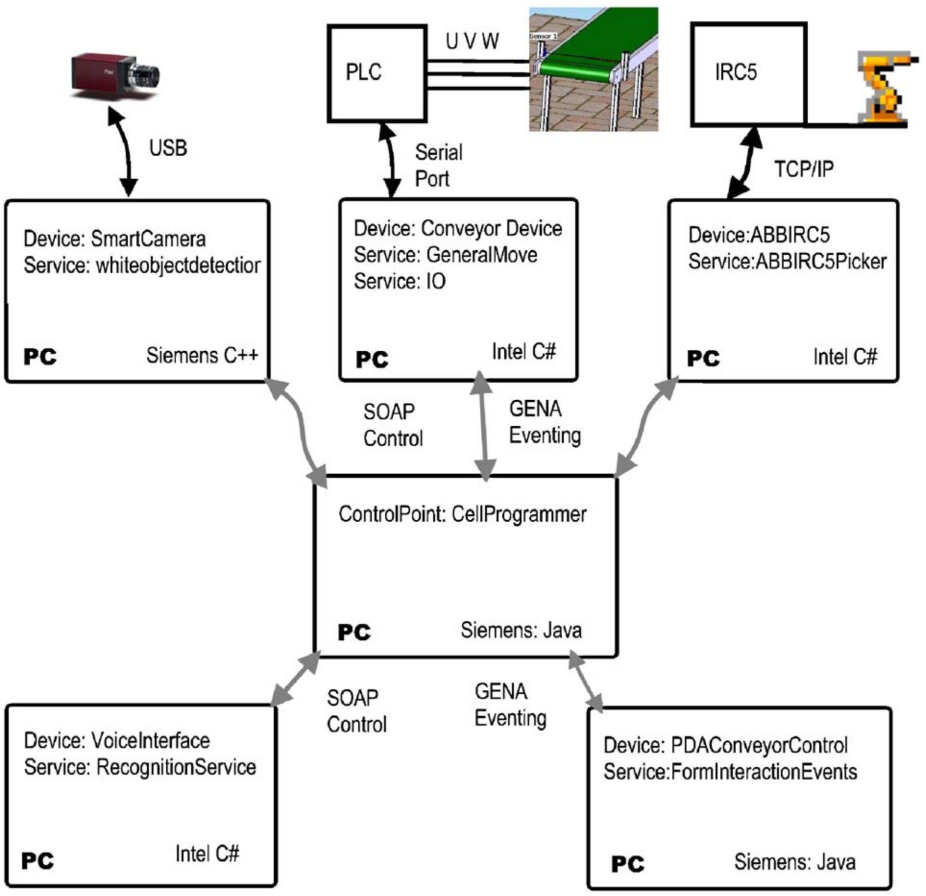


Fig. 2. UPnP devices and designed interconnections.

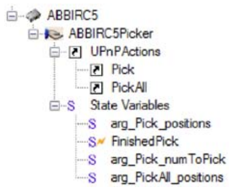


Fig. 3. UPnP device developed for the industrial robot.

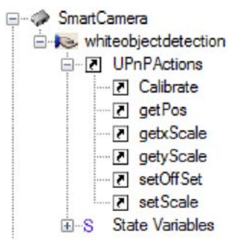


Fig. 5. UPnP device developed for the smart camera.

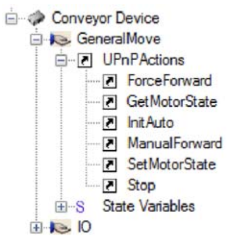


Fig. 4. UPnP device developed for the conveyor.

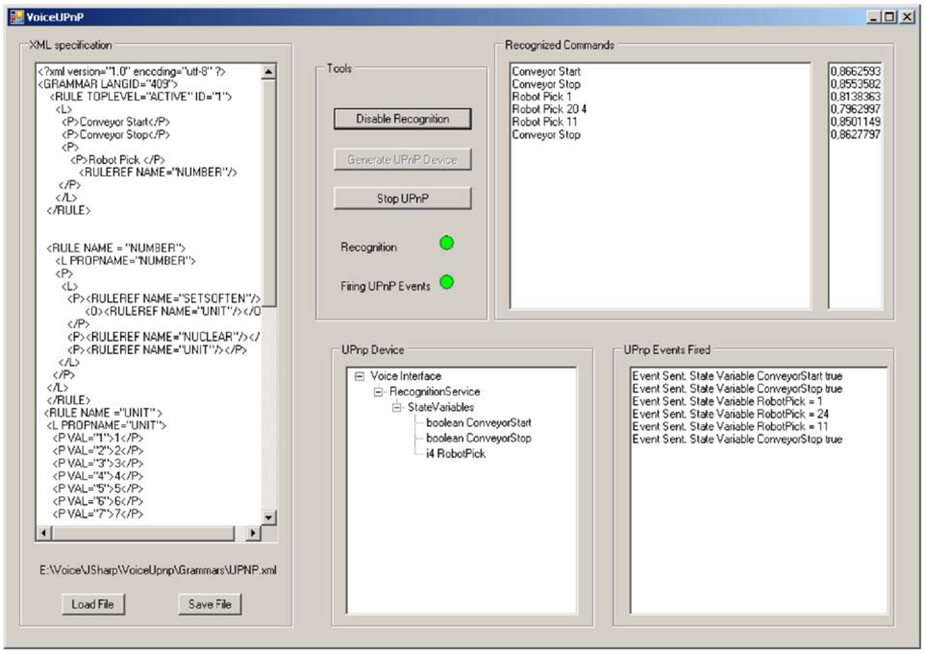
event is fired by the SRS. The Microsoft SAPI allows three different ways for specifying grammars: included in the code (programmatic grammars), using XML files, or using CFG files. Since XML is a well accepted standard it was used to specify speech recognition grammars.

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).

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 on 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 is 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 found on the network (notice the presence of a “stranger” gateway device). Clicking over them it is possible



```
<?xml version="1.0" encoding="utf-8" ?>
<GRAMMAR LANGID="409">
  <RULE TOPLEVEL="ACTIVE" ID="1">
   <L>
    <P>Conveyor Start</P>
    <P>Conveyor Stop</P>
    <P>
     <P>Robot Pick </P>
       <RULEREF NAME="NUMBER"/>
    </P>
   </L>
  </RULE>


  <RULE NAME = "NUMBER">
   <L PROPNAME="NUMBER">
   <P>
    <L>
     <P><RULEREF NAME="SETSOFTEN"/>
         <O><RULEREF NAME="UNIT"/></O
     </P>
     <P><RULEREF NAME="NUCLEAR"/></
     <P><RULEREF NAME="UNIT"/></P>
    </L>
   </P>
  </L>
  </RULE>
 <RULE NAME ="UNIT">
  <L PROPNAME="UNIT">
   <P VAL="1">1</P>
   <P VAL="2">2</P>
   <P VAL="3">3</P>
   <P VAL="4">4</P>
   <P VAL="5">5</P>
   <P VAL="6">6</P>
   <P VAL="7">7</P>
```

E:\Voice\JSharp\VoiceUpnp\Grammars\UPNP.xml

Conveyor Start	0,8662593
Conveyor Stop	0,8553582
Robot Pick 1	0,8138363
Robot Pick 20 4	0,7962997
Robot Pick 11	0,8501149
Conveyor Stop	0,8627797

```
⊟ Voice Interface
   ⊟ RecognitionService
      ⊟ StateVariables
         boolean ConveyorStart
         boolean ConveyorStop
         i4 RobotPick
```

```
Event Sent. State Variable ConveyorStart true
Event Sent. State Variable ConveyorStop true
Event Sent. State Variable RobotPick = 1
Event Sent. State Variable RobotPick = 24
Event Sent. State Variable RobotPick = 11
Event Sent. State Variable ConveyorStop true
```

Fig. 6. Voice recognition interface.

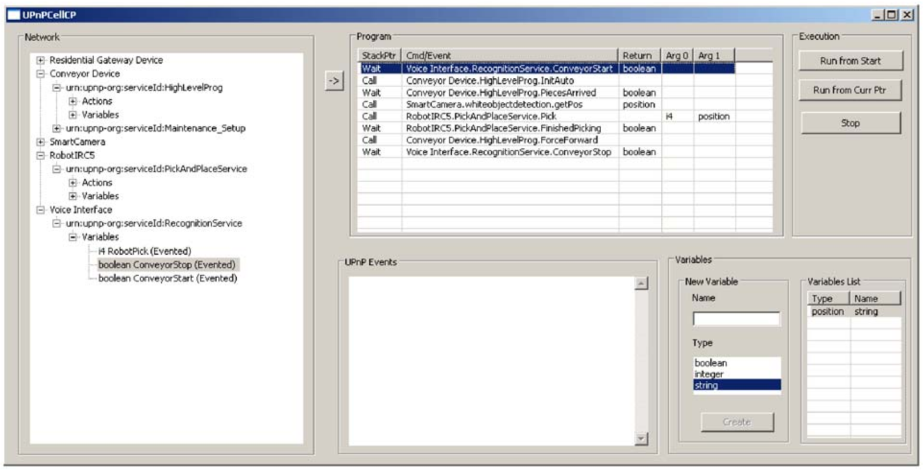
```
recContext.Recognition+=new _ISpeechRecoContextEvents_RecognitionEventHandler(this.handleRecognition);

 •••
 public void handleRecognition(int StreamNumber, System.Object StreamPosition,
     SpeechRecognitionType RecognitionType,
     ISpeechRecoResult Result)
 {
     SpeechDisplayAttributes a = Result.PhraseInfo.GetDisplayAttributes(0, 1, false);
     SpeechEngineConfidence confidence= Result.PhraseInfo.Rule.Confidence;
     confidence.ToString();

     int num=0;
     string cmd = "";

     if (confidence != SpeechEngineConfidence.SECLowConfidence)
     {
         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)
                 {
                     if (p.Name == "CMD")
                     {
                         cmd = (string) p.Value;
                     }
                     if (p.Name == "UNIT"
                         || p.Name == "SETSOFTEN"
                         || p.Name == "NUCLEAR")
                     {
                         num += (int)p.Value;
                     }
                 }
   •••
```

Fig. 7. Recognition handling delegate: retrieving semantic properties.

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.

The obtained information is used to pick-and-place the pieces calling the robot pick service.



```
⊞ Residential Gateway Device
⊟ Conveyor Device
    ⊟ urn:upnp-org:serviceId:HighLevelProg
        ⊞ Actions
        ⊞ Variables
    ⊞ urn:upnp-org:serviceId:Maintenance_Setup
⊞ SmartCamera
⊟ RobotIRC5
    ⊟ urn:upnp-org:serviceId:PickAndPlaceService
        ⊞ Actions
        ⊞ Variables
⊟ Voice Interface
    ⊟ urn:upnp-org:serviceId:RecognitionService
        ⊟ Variables
            i4 RobotPick (Evented)
            boolean ConveyorStop (Evented)
            boolean ConveyorStart (Evented)
```

StackPtr	Cmd/Event	Return	Arg 0	Arg 1
Wait	Voice Interface.RecognitionService.ConveyorStart	boolean		
Call	Conveyor Device.HighLevelProg.InitAuto			
Wait	Conveyor Device.HighLevelProg.PiecesArrived	boolean		
Call	SmartCamera.whiteobjectdetection.getPos	position		
Call	RobotIRC5.PickAndPlaceService.Pick		i4	position
Wait	RobotIRC5.PickAndPlaceService.FinishedPicking	boolean		
Call	Conveyor Device.HighLevelProg.ForceForward			
Wait	Voice Interface.RecognitionService.ConveyorStop	boolean		

Type: boolean, integer, [✓ string]

Type	Name
position	string

Fig. 8. UPnP control point: CellProgrammer interface.

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.

4.3. DSSP

The implementation of DSSP was made with the objective of allowing a precise comparison with UPnP counterpart: the MSRS package is used. Consequently, 3 services were developed which resemble the UPnP services: *ConveyorMRSR*, *Abbirc5* and *SmartCam*. All these services were developed using the C# programming language from the scratch not using any of the MSRS supplied services. This enables a more precise comparison, since it is done using similar services (built in the same way). Nevertheless, one exception has been allowed for the *VoiceUPnP* service. The speech recognition was developed using the MSRS provided speech recognition service and the recognition logic programmed using the Microsoft Visual Language Programming (MVLP) [10]. This allowed also evaluating MVPL.

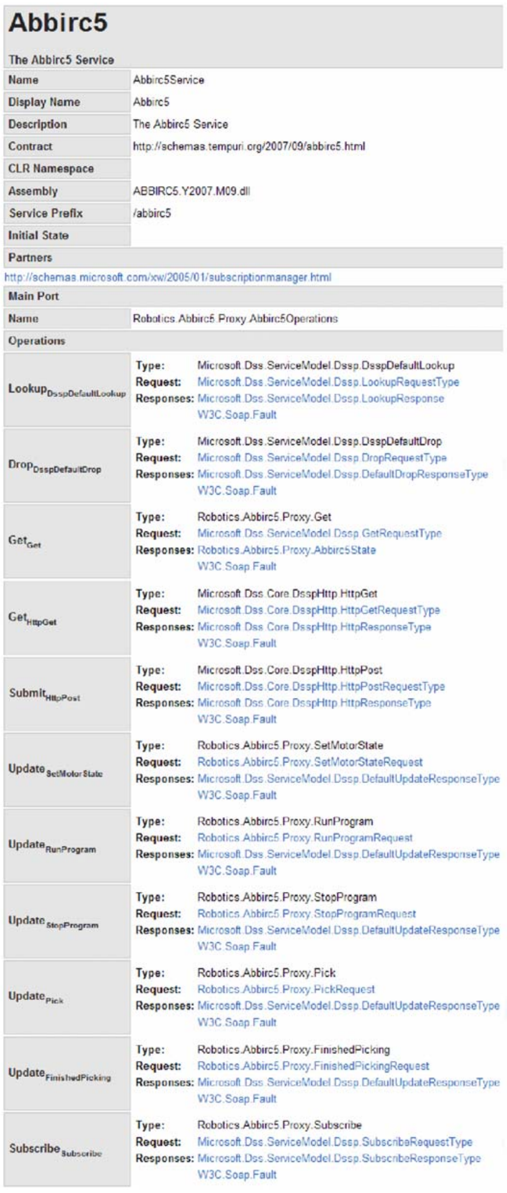
MSRS services are specified by contracts. These contracts specify which are the message ports available and which type of messages are available. Like any RESTfull approach the DSSP operations involves exchanging message of specific types which in this case are: CREATE, DELETE, DROP, GET, INSERT, LOOKUP, QUERY, REPLACE, SUBSCRIBE, SUBMIT, UPDATE and UPSERT. For the robot the following contract is available (Fig. 9).

Comparing this service with the one presented in the UPnP implementation, it is obvious to conclude that the *UpdatePick* operation is not a real Update operation but a way to accomplish to emulate the *Pick* method. On the other hand the *UpdateMotorState* is a nice substitute for the UPnP *MotorOn( )* and *MotorOff( )* methods.

The contract information is available via HTTP since the service is running and the state can be retrieved via the HTTP get operation (Fig. 10).

To improve the interaction with the services an extensible stylesheet language transformations (XSLT) transformation was developed. This transformation gives the html look to the extensible markup language (XML) state and provides a user interface to update the service state (Fig. 11).

The orchestration in the DSSP has been implemented using the MVPL (Fig. 12).



Abbirc5

The Abbirc5 Service

Name	Abbirc5Service
Display Name	Abbirc5
Description	The Abbirc5 Service
Contract	http://schemas.tempuri.org/2007/09/abbirc5.html
CLR Namespace	
Assembly	ABBIRC5.Y2007.M09.dll
Service Prefix	/abbirc5
Initial State	
Partners	

http://schemas.microsoft.com/xw/2005/01/subscriptionmanager.html

Main Port

Name	Robotics.Abbirc5.Proxy.Abbirc5Operations

Operations

$\text{Lookup}_{\text{DsspDefaultLookup}}$	**Type:** Microsoft.Dss.ServiceModel.Dssp.DsspDefaultLookup **Request:** Microsoft.Dss.ServiceModel.Dssp.LookupRequestType **Responses:** Microsoft.Dss.ServiceModel.Dssp.LookupResponse W3C.Soap.Fault
$\text{Drop}_{\text{DsspDefaultDrop}}$	**Type:** Microsoft.Dss.ServiceModel.Dssp.DsspDefaultDrop **Request:** Microsoft.Dss.ServiceModel.Dssp.DropRequestType **Responses:** Microsoft.Dss.ServiceModel.Dssp.DefaultDropResponseType W3C.Soap.Fault
Get_{Get}	**Type:** Robotics.Abbirc5.Proxy.Get **Request:** Microsoft.Dss.ServiceModel.Dssp.GetRequestType **Responses:** Robotics.Abbirc5.Proxy.Abbirc5State W3C.Soap.Fault
$\text{Get}_{\text{HttpGet}}$	**Type:** Microsoft.Dss.Core.DsspHttp.HttpGet **Request:** Microsoft.Dss.Core.DsspHttp.HttpGetRequestType **Responses:** Microsoft.Dss.Core.DsspHttp.HttpResponseType W3C.Soap.Fault
$\text{Submit}_{\text{HttpPost}}$	**Type:** Microsoft.Dss.Core.DsspHttp.HttpPost **Request:** Microsoft.Dss.Core.DsspHttp.HttpPostRequestType **Responses:** Microsoft.Dss.Core.DsspHttp.HttpResponseType W3C.Soap.Fault
$\text{Update}_{\text{SetMotorState}}$	**Type:** Robotics.Abbirc5.Proxy.SetMotorState **Request:** Robotics.Abbirc5.Proxy.SetMotorStateRequest **Responses:** Microsoft.Dss.ServiceModel.Dssp.DefaultUpdateResponseType W3C.Soap.Fault
$\text{Update}_{\text{RunProgram}}$	**Type:** Robotics.Abbirc5.Proxy.RunProgram **Request:** Robotics.Abbirc5.Proxy.RunProgramRequest **Responses:** Microsoft.Dss.ServiceModel.Dssp.DefaultUpdateResponseType W3C.Soap.Fault
$\text{Update}_{\text{StopProgram}}$	**Type:** Robotics.Abbirc5.Proxy.StopProgram **Request:** Robotics.Abbirc5.Proxy.StopProgramRequest **Responses:** Microsoft.Dss.ServiceModel.Dssp.DefaultUpdateResponseType W3C.Soap.Fault
$\text{Update}_{\text{Pick}}$	**Type:** Robotics.Abbirc5.Proxy.Pick **Request:** Robotics.Abbirc5.Proxy.PickRequest **Responses:** Microsoft.Dss.ServiceModel.Dssp.DefaultUpdateResponseType W3C.Soap.Fault
$\text{Update}_{\text{FinishedPicking}}$	**Type:** Robotics.Abbirc5.Proxy.FinishedPicking **Request:** Robotics.Abbirc5.Proxy.FinishedPickingRequest **Responses:** Microsoft.Dss.ServiceModel.Dssp.DefaultUpdateResponseType W3C.Soap.Fault
$\text{Subscribe}_{\text{Subscribe}}$	**Type:** Robotics.Abbirc5.Proxy.Subscribe **Request:** Microsoft.Dss.ServiceModel.Dssp.SubscribeRequestType **Responses:** Microsoft.Dss.ServiceModel.Dssp.SubscribeResponseType W3C.Soap.Fault

Fig. 9. Abbirc5 contract.

As already mentioned, the speech recognition logic is exposed here to promote the visibility of MVPL capabilities. This data/message driven language is very powerful to orchestrate complex services. As with the UPnP's CellProgrammer application, the use of MVPL as orchestration language is motivated by the need to facilitate integration of devices, rather than programming them.

4.4. *UPnP/DSSP comparison*

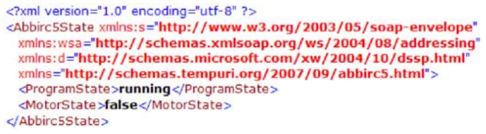
Experiments using UPnP and DSSP with the designed industrial test bed provided valuable information that can be used to select the most adequate SOA style for industrial robotic cell programming. The proposed comparison considers two main topics: architecture style and actual platform.

4.4.1. *Architecture style comparison*

The architecture styles are radically different between DSSP and UPnP. This is particularly visible in MSRS since there are no RPC-like methods in Microsoft's DSSP. In this architecture everything is a message, and everything is driven by (or driving) state changes. This is an immediate consequence of the RESTful flavor of the architecture. Of course that every RPC call can always be replaced by a specific message and most of the times by a CRUD message (Create, Retrieve, Update and Delete) without the need of creating a specific message. The questions are if this model is the best way to express what the programmer has in mind, and if the actual procedural programming model of most of the industrial robots will cope well with the REST style.

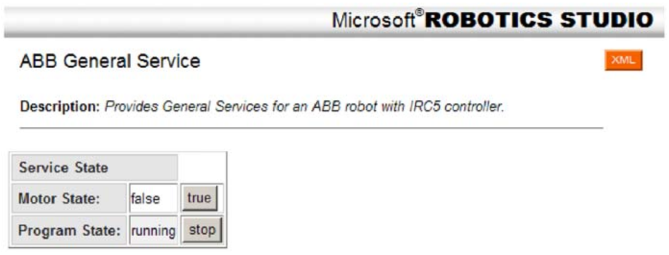
For example, consider the action "Pick" from the *AbbIrc5* from the UPnP setup. This method represents the operation of picking pieces and has as arguments the position of the pieces and the number of pieces to pick. This operation does not fit in any of the DSSP message operations and the update-message used seems an unnecessary workaround.

An important issue to consider regarding the industrial automation integration is the way services will be specified. Most industrial robots still define their tasks using a procedural language (object oriented in some cases). These languages seem very adequate to specify services to the network, using one approach similar to the one used with Web Services (SOAP/WSDL) and languages like Microsoft Visual C# or Microsoft Visual Basic.



```
<?xml version="1.0" encoding="utf-8" ?>
<Abbirc5State xmlns:s="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:d="http://schemas.microsoft.com/xw/2004/10/dssp.html"
  xmlns="http://schemas.tempuri.org/2007/09/abbirc5.html">
  <ProgramState>running</ProgramState>
  <MotorState>false</MotorState>
</Abbirc5State>
```

Fig. 10. Abbirc5 state.



Microsoft® ROBOTICS STUDIO

ABB General Service

XML

Description: *Provides General Services for an ABB robot with IRC5 controller.*

Service State		
Motor State:	false	true
Program State:	running	stop

Fig. 11. Abbirc5 XSLT/HTML interface.

4.4.2. *UPnP/DSSP platforms comparison*

The following discussion is based on the experienced gained with implementing both architectures with the presented test bed. The idea here is to point out the main differences and highlight the interesting features of each technology.

4.4.2.1. Language/platform independence. In terms of language/platform independence UPnP takes the lead. Built over common standards there are toolkits available for every major platform (Windows, Linux) and languages (C#, Java, C++). Moreover, the MSRS relies exclusively on the .NET platform.

4.4.2.2. Concurrency support. Even if we consider coarse-grained services as stated before, the availability of powerful tools (library, SDK, etc.) to help the deployment of concurrent programs is very important. Considering for example the UPnP experiment, it is

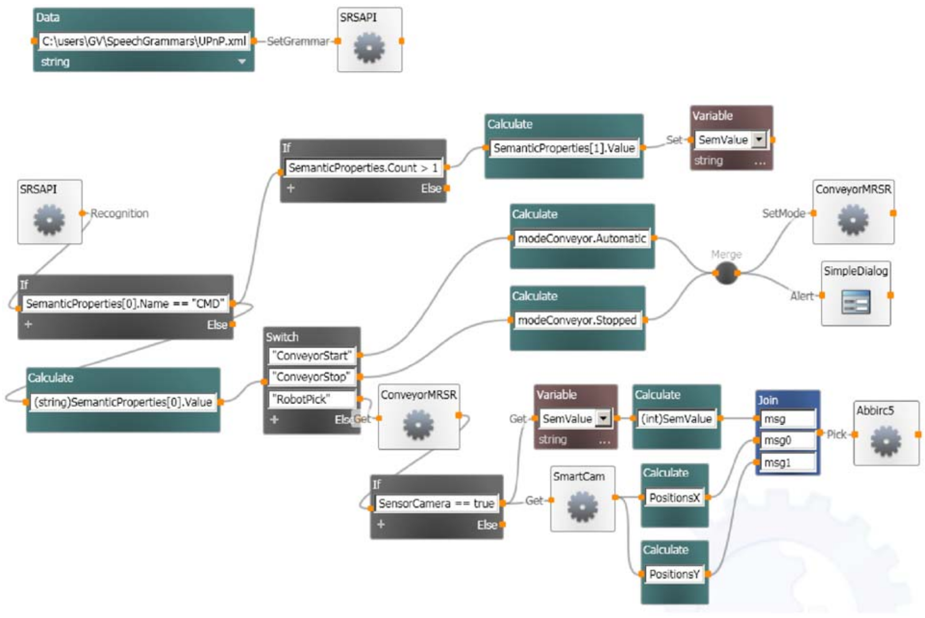


Fig. 12. Orchestration using MSRS.

considered very useful to have support for concurrent stack programs in the CellProgrammer application. This could be provided just by adding graphical support to some concurrent features (semaphores p.e.) from an existing library (Lund Java-based real-time library would be a suitable example [24]). DSSP and CCR constitute the core of the MSRS Runtime. Both technologies are tied up which means that DSS has an extensive and modern concurrency support.

4.4.2.3. High level orchestration programs. MSRS default package includes the MVPL, which is a very interesting solution for the orchestration of services. In fact, this environment can be used to perform the same function of the CellProgrammer used in the UPnP setup, with several advantages, namely related to the coordination of features using a graphical interface.

4.4.2.4. Discovery. The UPnP discovery is processed on a peer-to-peer basis. Every control point has the ability to discover devices. In the Microsoft Runtime, services can discover each other through a simple discovery service that acts as a rendezvous point between services running on a runtime and between runtimes. This can be a problem since a failure with discovery service may stop the discovery mechanism.

4.4.2.5. Security. Although many industrial networks are divided from the outside office network, it is always good to have security mechanisms in the SOA platform. UPnP does define a security mechanism [14] but it is not mandatory and add arrived far later than the first specification, which led to many proprietary security protocols, and to the absence of security in many programming stacks. The DSS runtime has a set of infrastructure services to deal with security issues. This solution is better than that presented in UPnP since once you have the runtime you have standard security.

4.4.2.6. HLP support. One of the things that a SOA should be is a suitable platform for the development of HLP features. One of the keys for HLP is environment sensibility (please see Section 4.4.2.8), which is easily reached with a "hot plug-and-play" discovery as we found in UPnP. In this scenario UPnP discovery (peer-to-peer) takes advantage. On the other hand, the mechanism that allows the composition of services provided by DSSP (called "partnering") seems a very powerful way of defining dependencies between services. Considering the HLP perspective these dependencies require a service to run in an HLP perspective that represent devices and services representing programming modules.

4.4.2.7. Maintenance services. Major advantages were pointed to the existence of maintenance services in the UPnP discussion.

4.4.2.8. Services available. Community dynamics momentum. Both technologies under test were not specifically designed to use with cell integration. UPnP is more suitable for home automation (specially the media rendering profiles) and DSSP from MSRS seems more suitable for fine-grained services (typically found inside a mobile robot: for example, controlling a motor with events from sensors), which means that it is not easy to reuse services.

MSRS intends to be an end-to-end solution for robotics and there is an enormous dynamics in developing new services with very interesting features. Services like speech recognition, or hand gesture recognition are shipped with the main installation of the runtime. These tools facilitate enormously the deployment of applications based on building blocks and many of them are useful to industrial cell programming. On the other hand UPnP does not have similar tools available.

5. Conclusions

The main objective of this paper was to review some of the recently proposed SOA technologies, and confront the most promising approaches with experimental setups reflecting real applications, i.e., using the selected SOA 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. A test bed was designed to implement two of the most promising SOA technologies: UPnP and DSSP, the SOA present in MSRS. The idea was to make a comprehensive comparison of both technologies and in the process discuss the utilization of SOA for system integration and HLP of robotic manufacturing cells. These architectures represent in the device level two major architectural styles originating from the office/enterprise software world which means that advantages and disadvantages pointed in this work can be of major importance for the definition of future domain-specific (industrial robotics) SOA platforms.

Focusing on industrial automation and specifically on industrial robotics cell programming, SOA can support automation engineers to focus on their expertise (machine vision, force control, etc.), allowing them to keep their favorite platform/language, to rely on the standard technologies, and to reduce their attention on the interconnection tricky tasks.

Programming industrial robotic cells using SOA as framework and friendly graphical interfaces for specification of system logic has proven to be less time consuming than traditional object oriented techniques applied against similar setup [22,25].

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 also 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.

The comparison effort also showed that both technologies have interesting features that suit well the industrial environment and the request for HLP approaches. This means that a merging effort is worthwhile, namely focusing on the good discovering features of UPnP and the advantages of the graphical orchestration interface of the MSRS. One way to this end could be the adoption of graphical state–machine definition schemes (like State Chart XML) for the orchestration of services [26].

References

[1] Gou L., Luh P., Kyoyax Y., Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation, IEEE/ASME international conference on advanced intelligent mechatronics '97, vol. 37, 1998. p. 213–31.

[2] El-Kebbe Salaheddine DA. Towards a manufacturing system under hard real-time constraints. In Informatik 2000: 30. Jahrestagung der gesellschaft-für Informatik, Berlin, September 2000.

[3] XML-basiertes kommunikationsprotokoll für Industrieroboter und prozessor-gestützte peripheriegeräte, Stand 17.10.2005. Information sheet downloadable from ⟨http://www.vdma.org/xirp⟩.

[4] AUTOSAR. Werner Zimmermann/Ralf Schmidgall, "Bussysteme in der fahrzeugtechnik-protokolle und standards (in German, Eng title: Bus systems in vehicles—protocols and standards), Vieweg, ISBN 978-3-8348-0235-4, Specs at ⟨http://www.autosar.org/find02_ns6.php⟩.

[5] SIRENA Project. Service infrastructure for real-time networked applications, Eureka Initiative ITEA. ⟨www.sirena-itea.org.sadasd⟩, 2005.

[6] Ahn SC, Kim JH, Lim K, Ko H, Kwon Y, Kim H. UPnP approach for robot middleware P. In: Proceedings of the 2005 IEEE international conference on robotics and automation, Barcelona, Spain, April 2005.
[7] Nielsen H, Chrysanthakopoulos G. Decentralized software services protocol—DSSP/1.0, July 2007.
[8] James F, Smit H. Service oriented paradigms for industrial automation. IEEE Trans Ind Inf 2005;1(1).
[9] James F, Smit H. Service oriented device communications using the devices profile for web services. ACM Int Conf Proc Ser 2005;115(1).
[10] Robotics Studio, Microsoft Robotics Studio 〈msdn.microsoft.com/robotics/2007〉.
[11] Rekesh J. UPnP, Jini and Salutation—a look at some popular coordination frameworks for future networked devices, California Software Labs, 1999.
[12] Bettstetter C, Christoph R. A comparison of service discovery protocols and implementation of the service location protocol. In: Proceedings of EUNICE open European summer school, Twente, Netherlands, September 13–15, 2000.
[13] Jini. The community resource for Jini technology: 〈http://www.jini.org〉, 2007.
[14] UPnP forum, available at 〈http://www.upnp.org〉, 2006.
[15] Schlimmer J, Chan S, Kaler C, Kuehnel T, Regnier R, Roe B, et al. Devices profile for web services: a proposal for UPnP 2.0 device architecture. Available at: 〈http://xml.coverpages.org/ni2004-05-04-a.html〉, 2004.
[16] PnP-X: plug and play extensions for windows specification. Available at: 〈www.microsoft.com/whdc/Rally/pnpx-spec.mspx〉, 2006.
[17] Veiga G, Pires JN, Nilsson K. On the use of SOA platforms for industrial robotic cells. Intelligent manufacturing systems proceedings IMS2007, Spain, 2007.
[18] Microsoft, speech application programming interface (API) and SDK, version 5.1, microsoft corporation, 〈http://www.microsoft.com/speech〉, 2007.
[19] Abb: ABB IRC5 documentation, ABB flexible automation, Merrit, 2005.
[20] OpenCV. Available at: 〈http://sourceforge.net/projects/opencvlibrary/〉, 2007.
[21] Pires JN. Experiments on commanding an industrial robot using human voice. Ind Rob: Int J 2005;32(6) [Emerald].
[22] Pires JN, Godinho T, Araújo R. Controlo e Monitorização de Células Robotizadas Industriais: Revista Robótica. Abril. 2006.
[23] Siemens PLC S7-200 System Manual, 2000.
[24] Robertz SG, Henriksson R, Nilsson K, Blomdell A, Tarasov I. Using real-time Java for industrial robot control. In: Proceedings of the fifth international workshop on Java technologies for real-time and embedded systems. Vienna, Austria, September 26–28, 2007.
[25] Pires JN. Industrial robots programming building applications for the factories of the future. Berlin: Springer; 2007.
[26] Veiga G, Pires JN. Plug-and-produce technologies: on the use of statecharts for the orchestration of service oriented industrial robotic cells. ICINCO 2008, International Conference on Informatics in Control, Automation & Robotics. Madeira, Portugal, 11–15 May 2008.