Separation of concerns on the orchestration of operations in flexible manufacturing.

Germano Veiga[1], Pedro Malaca[1], J. Norberto Pires[1], Klas Nilsson[2]

[1] Mechanical Engineering Department,University of Coimbra
Pólo II, Rua Luis Reis Santos Coimbra Portugal
{gveiga, norberto}@robotics.dem.uc.pt
,pedro.malaca@dem.uc.pt

[2] Computer Science Department, Lund University,
P.O. Box 118, S-221 00 Lund, Sweden, (e-mail: klas@cs.lth.se).,

Abstract. The growing complexity of industrial robot work-cells calls for the use of advanced orchestration techniques to promote flexibility and reusability. This paper presents a solution based on service-oriented platforms that endorses the separation of concerns, coordination and execution. The execution is kept inside each individual device and their functionality is exposed via automatic generation of services. The orchestration is performed in a cell controller that uses a state-chart engine to achieve the cell behaviour. The statecharts defined for this engine are work-cell programs without implicit dependencies on the services and therefore reusable and easily maintainable. From the early evaluations made in this paper the SCXML based purposed language is more adapted to the industrial robotic cell scenario than existing alternatives. The generation of services allow the integration without knowledge of any programming language.

Keywords: Service oriented architectures, Orchestration.

1 Introduction

From a machinery point of view, robots are normally classified as flexible automation, but this classification is only valid for large companies. The flexible automation concept relies on the ability of industrial robots to be reprogrammed and reconfigured, but these tasks are still performed by specialized technicians and automation experts. Looking at Small and Medium companies (SME), it is easy to find many features missing from modern industrial robots for them to be co-workers that help humans in harder tasks. An SME shop floor is unstructured and changes occur very often in the layout, which demands robotic systems to be easier to reconfigure. An industrial robotic cell can be viewed as a distributed environment with multiple dedicated processing units. The reconfiguration of these systems is

strongly connected to the existence of industrial robotic cell programs that can act as glue between dedicated systems.

The idea of industrial robotic cell programs has been present since the introduction of the object-oriented approaches for industrial robotic cells (Lin et al., 1994). This approach has been successfully followed by several authors (J.N. Pires and Sa da Costa, 2000), evolving together with component-based solutions, leading most robot manufacturers to provide similar packages ("Yaskawa Motoman Robotics- MotoCom Software Development Kit," 2010)("ABB WebWare Server - Software Products (Robotics)," 2008). Even nowadays, research on robotic cell programming is made using object-oriented technologies (Bruccoleri, 2007). However, this approach has several drawbacks: knowledge of traditional programming languages (C++, C#, Java) is needed, which can be hard to find among robot programmers; due to the computational capabilities of this type of languages and to the characteristics of component-oriented programming it is usually difficult to promote the separation of concerns, computation and coordination, leading to coordination programs that can hardly be reused. In this work, these issues are addressed through the use of service-oriented architectures, together with the definition of a robotic work-cell programming language that promotes its use only on coordination. In Section 2 the analysis of the stated problem is made and the general architecture of the solution presented. Sections 3 and 4 describe the implementation of the system and the Section 5 draws some conclusions.

2 Analysis

Device level service-oriented platforms usually define a set of complementary technologies to the fundamental service-oriented tenets (used in the internet level). The combination of service-oriented styles with Publish/subscribe messaging patterns constitutes an interesting approach in defining a middleware to be used in an industrial environment, as postulated in (I. Delamer and Lastra, 2007). In this mixture, service-oriented principles should provide robust contract-based functionality specifications and straightforward interaction with higher layers of enterprise software. On the other hand, the Publish/subscribe messaging mechanism should provide a modern and efficient way of dealing with the event-driven environments from the shop floor.

The evaluation of device-level SOA platforms presented in (Veiga et al., 2008) has shown that their use can be very valuable to the acceptance of robots in SMEs. However, some critical issues were pointed out regarding the adoption of these network platforms, namely, who will program those services, and how can they be orchestrated (integrated) without programming in Java or C++

Another important aspect is the simplicity of the integration process. In this work, the target of the use of SOA is industrial robotics. Considering the past history of industrial automation programming languages, one can say that simplicity is very important, and special attention should be paid to the person who is going to ultimately use and maintain the system. PLC programming is a good example of how

important the match between the language and the language user is. Ladder diagrams were introduced in the 70's as a PLC programming language and its logic structure was inherited from relay circuits, in order to obtain acceptance from the engineers at that time. Its success throughout the years proves that simple dedicated programming languages can be very adequate for industrial systems. Furthermore, the ladder language has survived the coexistence with several languages (Wareham, 1988) that compose the IEC61131 standard (IEC, 2003), which purposed to address the limitations of the ladder language.

The integration of service-oriented architectures in SME robotic work-cells can be framed in different approaches. One approach is the integration within knowledge-based configurable systems working as middleware platform for plug-n-produce. The combination of semantics with service-oriented architectures is a promising research topic that has pushed forward several research efforts ("SIARAS FP6 PROGRAMME PROJECT SUMMARY," 2008)(I. Delamer and Lastra, 2006)(Malec et al., 2007) (Lastra and M. Delamer, 2006), including the EU project SOCRADES ("Home - Socrades - 2006," 2006). This is a long-term perspective that will explore the benefits provided by *device* and *service* templates present in most modern SOA middleware platforms. These profiles allow suppliers to agree on common functionality, which enables a consumer to design applications targeting generic services, regardless of their detailed implementations. In these scenarios, a robot user could p. e. plug a 2D vision system, a robot and a drilling machine into the work-cell network switch and, since all these devices comply with generic templates, a software application should be able to reason about the functionality provided by the complete set. As a result of this reasoning process, this application would propose the functionality of drilling holes in previously cross-marked places to the user, without any programming or networking knowledge. Although the results from these works are promising, the perspectives for usable platforms are only long-term. This reality is further enhanced by the fact that this approach is very dependent on standardization efforts. It is also important to note that, regardless of the evolution of the reasoning systems, these are expected to be very close to application types, like for example welding or gluing, thus exploring the synergies present in the actual industry panorama. In the short term, this fact limits the flexibility of the solution.

A different short-term approach presented in this work relies on the flexibility of automatic generation techniques, like modern compiler technologies and programming environments, to integrate devices using current state of the art technologies.

The general architecture of this solution is presented in Fig. 1.

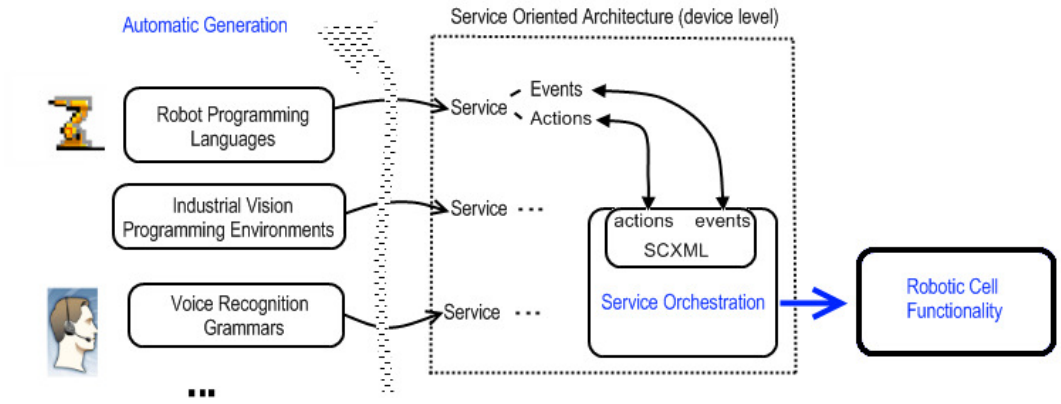

Fig. 1. General Architecture

In this approach, the simplicity of the system is a major goal.

In terms of service programming**,** this alternative presents the use of current technologies, either in terms of programming languages or programming environments, in an attempt to bridge the gap between the programmer and the technology. In this work, two examples (section 3.1 and section 3.2)are presented that demonstrate the automatic generation of services from voice recognition and robot programming grammars.

The success of domain-specific languages (DSLs) in industrial automation, together with the conclusions retrieved from the SOA evaluation carried out in (Veiga et al., 2008), point to a domain-specific graphical programming language as a powerful way of orchestration. Therefore, this solution explores this idea and proposes a visual programming language for usage in service-oriented architectures. This two-layer solution with different languages for coordination and computation (or execution) exists in other areas of computer science. In (Gelernter and Carriero, 1992) Gelernter describes a programming model with two different pieces, the coordination (composition) and the computation model, and develops a language focusing on coordination, *Linda* (Carriero and Gelernter, 1989). In (Browne et al., 1995), a similar approach is used to model parallel computing, where traditional sequential languages were used to write processing units whose concurrent behaviour was later specified with a graphical notation.

In the opposite direction of some approaches for the use of service-oriented architectures in the industrial environment (“Home - Sirena 2003 - 2005,” 2005), where devices and service are normally small granulated, this approach tries to explore all the advantages of proprietary development platforms in order to achieve easier acceptance. In this proposal, all services are therefore holons representing a high level of functionality, which explore all the features of modern systems. A robot controller, p. e., is nowadays a complex and very powerful machine, with many advanced features like multitasking, advanced human-machine interfaces, and integrated force control. The use of service-oriented architectures from the ground up

would imply full commitment from the robot manufacturer in order to provide services for all these features.

In technological terms the SOA platform chosen for this work was the UPnP (Universal Plug-n-Play). The comparison made in (Veiga et al., 2008) shown the adequacy and representativeness of this platform and the previous work made with the platform provides an important jump start.

3 Service Generation

3.1 Robot programming languages

Nowadays industrial robots are still programmed manually with extensive use of the teach pendant. Therefore, the importance of robot programming languages is high, both from the industry's point of view, as a distinctive feature, and from the robot programmers', since it affects their productivity directly. In this section, robot programming languages are proposed as a way to specify contracts.

Since programming a robot is one of the key concepts that support flexible manufacturing, it is tempting to think that a robot program could be used in a code-first approach to the specifying of services within an SOA environment.

Each industrial robot manufacturer has designed its own language to program robots. Despite many efforts to unify robot programming languages (see e.g. [109]), there is no expectation for a single language in the future. All these languages are very different and have some unique characteristics. Considering five of the major manufacturers, ABB, Kuka, Motoman, Comau and Fanuc, we can see three different concepts for their programming languages. Motoman's programming language, INFORM III (Motoman, 2006), is a high-level programming language, strictly close to robotics with no separation between data and functions (or procedures) that consume data, no support for structures, nor basic data types like boolean or string. Fanuc robots can be programmed with two different languages: Teach Pendant Programming (TPP) (*FANUC Robot series Handling Tool*, 2008), a high-level script-based programming language with the same limitations pointed out in INFORM III, and Karel (*FANUC Robotics SYSTEM R-J3iB Controller KAREL Reference Manual*, 2002), a pascal-like programming language. It is important to notice that, due to the existence of TPP, Karel is not very common in industrial installations. ABB's Rapid programming language (ABB, 2008), Kuka's Kuka Robot Language (KRL) (*KUKA Documentation V5.x - Advance Programming*, 2006) and Comau's PDL2 (Comau Robotics, 2005) are similar to Fanuc's Karel and fully support modularization, basic data types, structured/user-defined data types, and clearly separate data from functions. This features are important since functions and variables in the robot programming language can be fully associated with the basic building blocks device-level SOA services, that are actions and state variables,

In this work, we will use ABB's Rapid language as the source specification for our

services.

3.1.1 Software developed

Since the robot controller does not support the development of network comunications inside, the need arose for the use of an intermediate layer, which consists of a PC connected to the controller via a TCP/IP socket connection. The general software architecture is presented in Fig. 2. In this figure, the grey arrows represent configuration stages and the black arrows represent communication on operation.

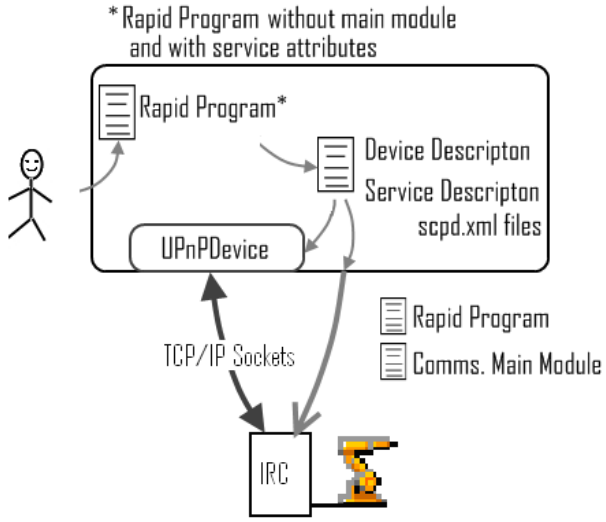

Fig. 2. Software Architecture

During configuration, the user of the system, typically the system's integrator, writes the wanted rapid program and defines which part of it should be available in the network. This Rapid program is parsed and, as a result, a device description is generated.

From the service description, two software applications are generated: a rapid program that handles the communication with the PC, and a PC application that works as a hub for all the communications to and from the robot. Both these applications are strictly related.

In order to add the necessary expression power to the robot programming language, the choice has fallen over the use of metadata annotations. Metadata annotations provide a powerful way of extending the capabilities of a programming language and its runtime. Also known as attributes, these annotations can be directives that request the runtime to perform certain additional tasks, provide extra information about an item or extend the abilities of a type. Metadata annotations are common in a number

of programming environments, including Microsoft's COM and the Linux kernel.

To match the basic elements of a service-oriented architecture in the device level, i.e., service, action and state variable, three attributes were chosen: the "![Service]" attribute can be used in a Rapid module and matches one UPnP service; the "![StateVariable]" attribute can be used in a Rapid variable and matches a UPnP StateVariable; and the "![Action]" attribute targets Rapid Procedures or Functions and matches UPnP actions. These attributes are used as Rapid comments, which allows the programmer to develop and test the program directly in the robot controller without changes. In the listing above, a Rapid program with the attributes defined in this work is presented.

The "![Service]" attribute allows the robot programmer to organize the functionality of different robot programs present in the robot. Metadata annotations are usually placed before the language element and the same applies to the specification proposed here, with the exception of the "![Service]" element. Nevertheless, the attribute being inside a comment, the robot controller complains about its presence outside modules, and this element should therefore be placed after the name of the module (Listing 1).

```
MODULE ABBIRC5Picker![Service]
  VAR bool okBool;
  ![StateVariable]
  PERS bool FinishedPick;
  ![Action]
  PROC PickAll(string positions)
     auxIndex := StrFind(positions
     ...
     MoveL Offs(camAux,0,0,-46), v150, fine, tool0;
     Set DO06;
     ...
     FinishedPick:=true;
  ENDPROC
```

Listing 1. Excerpt of sample robot program

The ![Action] attribute specifies that the annotated Rapid routine will have a UPnP counterpart. There is a restriction in routine parameters to the use of Rapid atomic types: bool, num and string, that will be matched to the UPnP types boolean, real (r8) and string, respectively. Only the normal type of Rapid parameters (see section 5.2.2 for the definition of Rapid normal type of parameters) is allowed. The other kinds of data, INOUT, PERS and VAR, imply an update of the value whenever it is changed inside the routine. The tempting solution of using UPnP out arguments to tunnel the value update all the way until the UPnP layer would have led to a situation where two different values (a Rapid variable and an UPnP variable) would have to be refreshed and thereafter always synchronized. This would be achievable through a mandatory requirement that each bidirectional parameter should also be an UPnP state variable and that only those could be used as parameters. This solution would have been hard to explain to a user, and clumsy to implement, and the real added value is limited.

Therefore, the option for a restriction to the normal type of parameters was made.

The "![StateVariable]" metadata annotation is used in every Rapid variable that is supposed to be available in the network. The restriction to atomic types is also applicable here, and only persistent data is allowed. This limitation is due to the technical implementation of the rapid TCP/IP socket server, that uses trap routines (only available for persistent variables) to handle changes in variables.

Rapid Service Generator (Fig. 1) is the software application that materializes the concepts presented before. The three main tasks of this software application are: parsing the Rapid program and building the device/service descriptions; hosting an UPnP device (based on the UPnP Intel Stack (Intel, 2008)) that works like a layer to expose the robot application in the service-oriented environment; and finally, generating the dedicated TCP/IP socket server that will be hosted on the robot and will communicate with the Rapid Service Generator.

The graphical human-machine interface allows the user to make some simple editing of the Rapid program, check if the program complies with the defined syntax, test UPnP actions with direct calls, download the program to the robot (both original and the dedicated TCP/IP socket server) and start/stop UPnP service hosting.

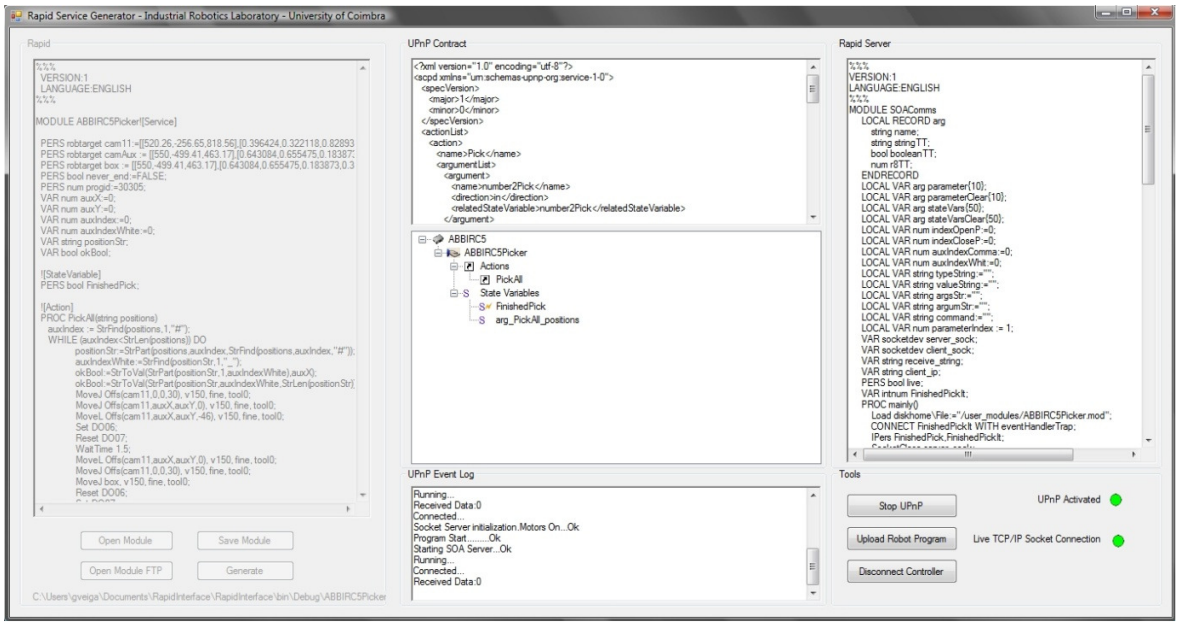

Fig. 3. Graphical interface of the Rapid service generator

The *Rapid* TCP/IP socket server is composed by two major parts. The main procedure is the one that will be running in the controller and is composed by: loading of the programmed module, creating socket connections, connecting state variables

with *Trap* event handlers and a *while* structure that is waiting for call connections from the PC application.

The use of Rapid *Trap* handlers to monitor the value of state variables is the reason for the limitation to the use of this kind of variables, since only these can be connected to this type of handlers.

The second part of the code implements the *Trap* routine that fires sockets to the PC application whenever a monitored variable changes.

Videos from the presented application can be seen in:
http://robotics.dem.uc.pt\germanoveiga\rapidUPnP.wmv.

An important consideration needs to be made regarding the support of Rapid *functions.* Since the early stages of the development of this application, Rapid *functions* have been supported by the generation of an UPnP method with return value when a *Rapid* function was found. However, they should be used very carefully, since the UPnP answer is ruled by a timeout that can be shorter than the speed of the robot in executing a task and proceeding with the network answer. In fact, this situation is almost always present in the scenario proposed in this paper. In this approach, the idea of exploring all the capabilities of the robot programming language indicates that the exposed services should materialize into high-level functionality, which most certainly means that the robot will move. The movement of the robot implies a time delay that is incompatible with the UPnP timeout, which means that specifying Rapid *functions* as UPnP Actions is almost useless. The most flexible way to achieve coordination is to use a *Rapid* variable (properly networked with the corresponding metadata annotation) that can be addressed at the end of the procedure, as shown in the Listing 1, where a *Bool* variable, *FinishedPicking*, is used to notify the end of the pick operation.

The implementation of the TCP/IP socket system is detailed in (Veiga and J.N Pires, 2008).

3.2Speech Recognition System

In the first Automatic Speech Recognition Systems (ASR)-related studies, grammatical and syntactical rules were used to validate speech. These approaches failed due to the numerous exceptions that human languages have, like dialects and accents, but also because when someone speaks, they do not enunciate each word separately. An important corner stone in the development of continuous speech systems (in opposition to former isolated word systems) was the introduction of the language models, of which the n-gram (Manning and Schütze, 1999) is by far the most used. These models predict the relation of a word with other n-words, statistically, and are very effective in guiding a word search for speech recognition.

Despite the need for further work on the recognition of conversational speech, speech recognition technologies regarding read speech are mature and ready to complement other means of human-machine interaction. These advances leaded to the

development of several commercial products that are now reaching wider markets.

These products use two different modes that are usually supported by modern speech recognition systems: the dictation and the command modes.

In the dictation mode, data can be directly entered into the computer by speaking, allowing the user to compose an email or write a text. In this mode, the ASR tries to match the input with a complete language (English, for example).

In the command mode, the programmer of the system introduces a grammar (a new layer over the global grammar) that limits the amount of possible commands. In this way, considering a well-defined grammar, these systems provide better accuracy and performance, and reduce the processing overhead required by the application.

Even though the support for the dictation mode has been available in modern operating systems and commercial software packages since 2002 (Windows XP or Dragon NaturallySpeaking) with good levels of performance, their acceptance is still limited outside some niche markets like healthcare. There are many possible explanations for this, and they range from the social acceptance of "speaking to a machine" to the use of poor microphones that degrade the performance in a way that makes the system useless.

From a practical point of view, the most robust (speaker-independent) voice-enabled systems present in the market are still working in command mode.

The speech recognition can be divided in two major steps: recognition and interpretation. Within the VoiceXML ("Voice Extensible Markup Language (VoiceXML) Version 2.0," 2010) standard the W3C consortium these two steps match two parts of the standard: • SRGS – Speech Recognition Grammar Specification ("Speech Recognition Grammar Specification Version 1.0," 2010) and SIRS – Semantic Interpretation for speech recognition ("Semantic Interpretation for Speech Recognition (SISR) Version 1.0," 2010).

3.1.1 Speech in industrial automation

The industrial use of voice-enabled systems is a promising concept. Consider for example a line operator who has both hands occupied, one holding a polishing tool and the other holding the product. The ability to control the conveyor, to set the spindle speed or even some extra equipment with the voice provides substantial added value. In the same way, a MIG-MAG welder can tune welding parameters in the middle of a seam, without needing to stop the procedure.

The idea of using voice recognition in industrial systems is not new (see (Foster and Bryson, 1989)(Mital and Leng, 1991)) but the full exploitation of this concept suffered from the applicability problems of general ASR systems, further enhanced by the characteristics of the industrial environment, which raises some extra challenges concerning safety or the reliability of the recognition. Nowadays ASR systems, when used in command mode with restricted grammars, have finally given researchers the ability to develop systems with an industrial level of robustness (J.N. Pires, 2005), and have supported the development of the first voice recognition commercial

solutions for industrial automation ("Voxware: Voice Software for Voice-Based Warehousing," 2010).

3.1.1 Speech recognition grammars

A grammar defines the words and patterns of words that a user can say at any particular point in a dialogue. When a programmer specifies a grammar, they define a set of words and patterns of words to be recognized by the system. These grammars have the expression power of a Context-Free Grammar (CFG) (Hopcroft et al., 2000).

3.1.2 Voice recognition integration in robotic work-cells

Recent SRS can be used in the industrial environment, see ("Voxware: Voice Software for Voice-Based Warehousing," 2010). This type of technology can be integrated seamlessly in an SOA middleware because it is extensively event-driven. To achieve this integration, a software application that allows the automatic pairing of voice recognition events with UPnP events has been developed (**Error! Reference source not found.**).

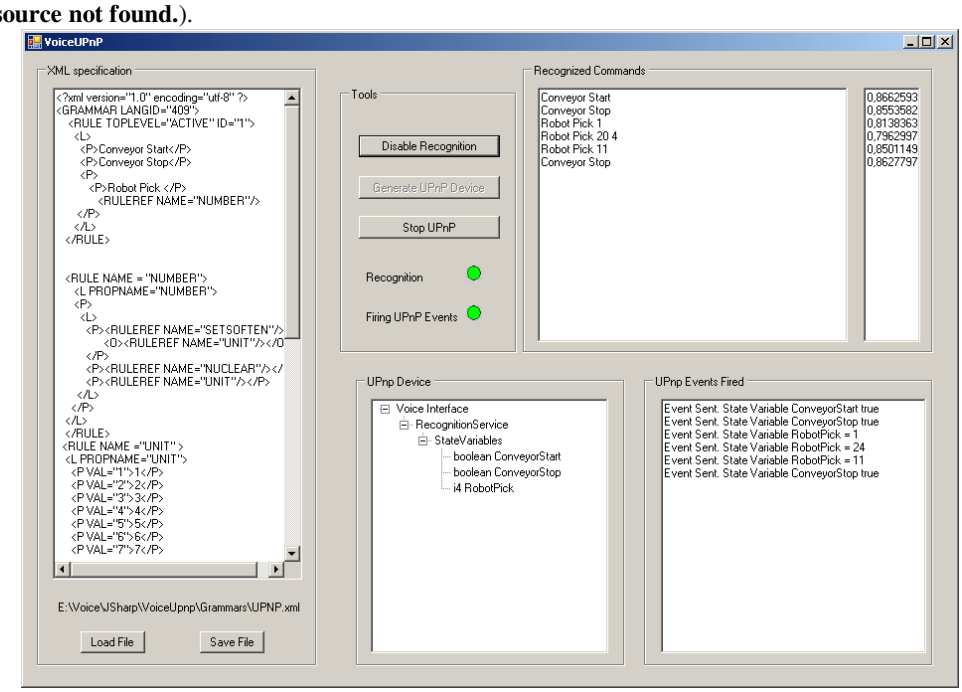

Fig. 4. Voice Recognition Interface (SAPI 5.1)

The SRS selected for use within this work was the Microsoft speech engine included in the *Microsoft Speech Application Protocol Interface* MSAPI 5.1 (Microsoft, 2008). 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 that implemented the following strategy: first, the XML grammar was parsed and an XML-DOM (Document Object Model) tree document created; this tree was afterwards traversed and UPnP state variables dynamically created and added to the RecognitionService of the voice interface device. All combinations implemented with grammar tags <L><\L> (List) were listed, and a Boolean state variable created for each one of them. The name of the state variable was the recognized sentence without spaces. Nevertheless, if this traversal method went 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 could contain a number. It is important to notice that the UPnP events were fired every time a new value was assigned to the state variable, even if the value was the same.

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

Grammars define a TopLevel Rule that include all the necessary commands. From each of these commands it is possible to call other rules. In the example presented in Fig. 4, 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 associated properties.

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 (Listing 2).

```
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();

  double 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")
              {
                 int lixo = (int)p.Value;
                 num += (double)lixo;
              }
           }
           if (num != 0)
           {
              if (this.btn_Advertise.Text == "Stop UPnP")
              {
device.GetService("RecognitionService").SetStateVariable(cmd, num);
                 this.txtBoxUpnpEvents.Text +=
                                                 "Event Sent. State Variable
" + cmd + " = " + num;
              }
           }
           else
           {
               Hanling of non- numeric variables
           }
           txtBoxUpnpEvents.Text += System.Environment.NewLine;
        }
     }
     string curr;
     curr = Result.PhraseInfo.GetText(0, -1, true);
     int aux = confidence.ToString().IndexOf("Confidence");
     this.txtBoxConfidence.Text += confidence.ToString().Substring(3,
aux - 3);
     txtBoxConfidence.Text += System.Environment.NewLine;
     this.txtBoxRecoCmd.Text += System.Environment.NewLine;
  }
}
```

Listing 2. Recognition handling delegate: retrieving semantic properties

This application provides a very interesting approach to linking 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 their respective devices, or even linked to ontology in robotics.

Detailed description of the developed software can be found in (Veiga et al., 2008).

4 Orchestration of services

The concept behind the orchestration of services (or composition in a larger sense, according to (Pautasso and Alonso, 2003)) is derived from the one introduced by component-oriented programming, which, like in many other WS*-related technologies, was extended and standardized using XML specifications. The relevance of the composition to E-Business has promoted a spate of different proposals from major players: *XLang* from Microsoft ("SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," 2010), *Web Services Flow Language* from IBM ("Web Services Flow Language," 2008), *Web Services Choreography Language* from HP ("Web Services Conversation Language (WSCL) 1.0," 2008), all promoting different interaction patterns among services. These patterns have led to the coining of several terms: choreography, orchestration, automation, coordination, collaboration, and conversation, which characterize the composition of services. Web service composition is extensively revised in (Dustdar and Schreiner, 2005), with special emphasis on the composition's main issues: coordination, transaction, context, conversation modelling and execution modelling.

From the evolution process, a standard has emerged as the most promising: *Business Process Execution Language for Web Services* (BPEL4WS) ("OASIS Web Services Business Process Execution Language (WSBPEL) TC," 2010).

4.1 Analysis

In the device-level, the problem of the orchestration specification is basically unchallenged. Within the European project SIRENA (the first European Project that arose using device-level SOA for industrial automation), the orchestration subject was approached (Jammes et al., 2005), but only under the perspective of reusing business level languages. In SIRENA's child projects, SOCRADES and SODA, several similar efforts can be found, see e.g.(Colombo et al., 2005). These research efforts passed through the manual orchestration directly to problems related with automatic reasoning systems (automatic orchestration), without questioning the adequacy of the existing orchestration languages. In fact, this problem is mentioned in a report from the European project SODA (Mensch et al., 2007), where several different possibilities for orchestration languages are questioned: process-based, event-driven and data-driven approaches. Referring to the process-based approach, inherited from BPEL4WS, this report refers that this language needs enhancements for dynamic discovery of services and the use of events.

The SME scenario that is the target of this work can be classified as a reactive system, i.e. mainly event-driven, which is one of the reasons for the success of device-level SOA dealing with this type of interactions. Therefore, the orchestration language/technique needs to make a rupture with BPEL4WS.

Furthermore, from the evaluation made in (Veiga et al., 2008), one interesting detail became prevalent: although the graphical capabilities of the MVPL were a major advantage for most of the users during the learning process, the more concrete event/action structure presented by the simple programmer interface revealed easier to understand. Furthermore, some users pointed out the resemblance of *Sequential*

Function Charts as an advantage in the comprehension process, which is mainly due to the reactive nature of the industrial processes.
For the reasons described earlier, i.e, the absence of a standard for the needs of device-level orchestration, a new orchestration language is proposed based on statechart formalism.

4.3 StateCharts

The *Statechart* formalism was described by David Harel (Harel, 1987) in 1987 as facilitating the design of complex discrete-event systems. *Harel Statecharts,* or simply *Statecharts*, is a visual formalism that extends the finite-state automata formalism (traditional state-transition diagrams) with the notions of hierarchy, concurrency, history, and communication. Statecharts are hierarchical, i.e., a state may contain another statechart down to an arbitrary depth, which highlights the modularity and clustering of the visual formalism. These characteristics combined with the Or decomposition, which is the mutual exclusion among a group of states, are the fundamentals of the abstraction capabilities that Harel (Harel, 1987) said were missing from flat statemachines. As a result, state levels can be created within statecharts, and the designer can hide complexity whenever zooming out of the state chart without losing detailed resolution when zooming in is required. Two or more statecharts may be run in parallel, which means that their parent state is in two states at the same time. This permits state orthogonality, which means independence and concurrency, and is achieved via the And decomposition. Another important feature of statecharts is the possibility to hold historical information inside a state, conditioning the re-entrance into that state.

Statechart XML (SCXML) is a W3C specification (Barnett et al., 2008) that can be described as an attempt to render Harel Statecharts in XML. The aim of this standard is to provide a basis for future standards in the area of multimodal dialogue systems. Even though this effort is being carried out by the W3C group for voice technologies, SCXML provides a generic state-machine based execution environment and a modern (XML) state machine notation for control abstraction. In fact, SCXML is a candidate for control language within multiple markup languages coming out of the W3C.

Consider for example the microwave oven model presented in Fig. 5.

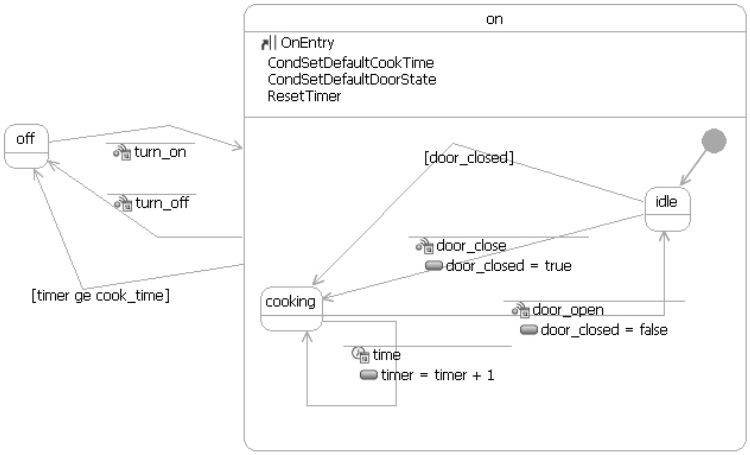

Fig. 5. Microwave oven(Adapted from ("SCXML - Commons SCXML," 2010))

The equivalent SCXML specification is:

```
<?xml version="1.0"?>
<scxml xmlns=
       "http://www.w3.org/2005/07/scxml"
       version="1.0"
       initialstate="off">

  <state id="off">
    <!-- off state -->
    <transition event="turn_on">
      <target next="on"/>
    </transition>
  </state>
  <state id="on">
    <initial>
      <transition>
        <target next="idle"/>
      </transition>
    </initial>
    <onentry>
      …
    </onentry>
    <transition event="turn_off">
      <target next="off"/>
    </transition>
    <transition cond="${timer ge cook_time}">
      <target next="off"/>
    </transition>
    <state id="idle">
      <transition cond="${door_closed}">
        <target next="cooking"/>
      </transition>
      <transition event="door_close">
```

```
        <assign name="door_closed" expr="${true}"/>
        <target next="cooking"/>
      </transition>
    </state>
    <state id="cooking">
      …
    </state>
  </state>
</scxml>
```

Listing 3. SCXML sample specification

As it can be seen in this example, an SCXML statechart can be divided into two major parts: the first one is composed by the machine states and their corresponding transitions, and the other by the executable content.

The SCXML executable content consists of actions that are performed as part of making transitions and entering and leaving states. The executable content is responsible for the modification of the data model, for raising events and invoking functionality on the underlying platform. It is worth noting that executable content cannot cause a state change, or fire a transition, except indirectly, by raising events that are then caught by transitions. This separation in the specification leaves room for platforms to add executable content corresponding to special features.

4.4 Proposed language and tests.

The language proposed for the orchestration of the work-cell was designed with simplicity in mind. The evaluation made with the users revealed that the orchestration language should be easily understandable and should be a thin layer, excluding many features, like mathematical, for instance. It is our opinion that for complex calculations the use of general programming languages like C# or Java is still a must, but that is not the target of this work; SME's are.

An example of this language is presented in Listing 4. The language is very similar to the SCXML except for the direct use of network events and actions. In this way, the need to write additional code, as with normal SCXML integrations, is avoided.

```
<?xml version="1.0"?>
<stateMachine xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" namespace="RoboticsLab"
name="CellController" initialState="Stopped" stateMachineType="Active">
  <state id="Stopped" historyType="None">
    <transition event="urn:schemas-upnp-
org:device:VoiceInterface:1;RecognitionService;ConveyorStart"
target="Conveyor Running">
      <UPnPAction name="urn:schemas-upnp-
org:device:Conveyor:1;GeneralMove;InitAuto" />
    </transition>
  </state>
  <state id="Conveyor Running" historyType="None">
    <transition event="urn:schemas-upnp-
org:device:Conveyor:1;GeneralMove;SensorCamera" target="Getting Pieces">
```

```
      <UPnPAction name="urn:schemas-upnp-
org:device:SmartCamera:1;whiteobjectdetection;getPos" />
    </transition>
  </state>
  <state id="Getting Pieces" historyType="None">
    <transition event="urn:schemas-upnp-
org:device:VoiceInterface:1;RecognitionService;RobotPick" target="Robot
Picking">
      <UPnPAction name="urn:schemas-upnp-
org:device:ABBIRC5Picker:1;ABBIRC5Picker;Pick">
        <UPnPArgument name="numToPick" val="urn:schemas-upnp-
org:device:VoiceInterface:1;RecognitionService;RobotPick" />
        <UPnPArgument name="positions" val="urn:schemas-upnp-
org:device:SmartCamera:1;whiteobjectdetection;ansGetPos" />
      </UPnPAction>
    </transition>
    <transition event="urn:schemas-upnp-
org:device:PDAInterface:1;FormInteractionEvents;RobotPick" target="Robot
Picking">
      <UPnPAction name="urn:schemas-upnp-
org:device:ABBIRC5Picker:1;ABBIRC5Picker;Pick">
        <UPnPArgument name="numToPick" val="urn:schemas-upnp-
org:device:VoiceInterface:1;RecognitionService;RobotPick" />
        <UPnPArgument name="positions" val="urn:schemas-upnp-
org:device:SmartCamera:1;whiteobjectdetection;ansGetPos" />
      </UPnPAction>
    </transition>
  </state>
  <state id="Robot Picking" historyType="None">
    <transition event="urn:schemas-upnp-
org:device:ABBIRC5Picker:1;ABBIRC5Picker;FinishedPick" target="Stopped">
      <UPnPAction name="urn:schemas-upnp-
org:device:Conveyor:1;GeneralMove;ForceForward" />
    </transition>
  </state>
</stateMachine>
```

Listing 4. XML program for the conveyor cell

To demonstrate the concepts discussed earlier, the test-bed used in (Veiga et al., 2008) was extended with an alternative form of command, a PDA Interface. This application was developed with a programming stack developed specifically to run on embedded devices (Windows CE).

Fig. 6. PDA Interface

The PDA interface includes the same functionality as the Voice Command: Conveyor Start, Conveyor Stop, Robot Pick [number of points]. With these commands, the user of the work-cell is supposed to alternatively use the voice interface or the PDA. This orchestration (Fig. 7) matches the orchestration program listed in Listing 4, and demonstrates the use of alternative state transitions, each of them with actions associated.

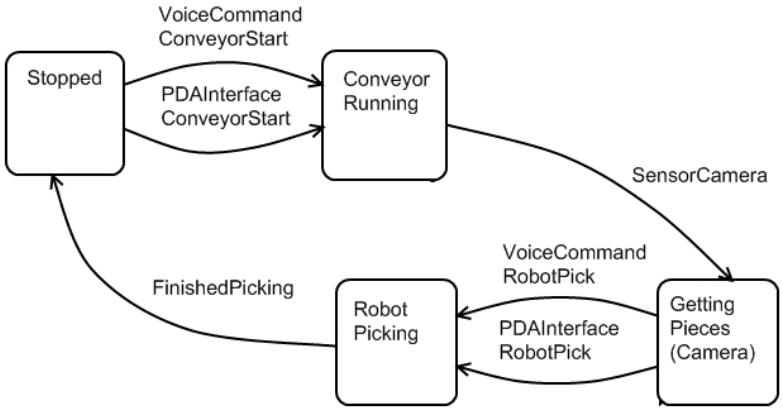

Fig. 7. Statechart with control via Voice or PDA

In this small example, the transition from the interaction stages is triggered by events raised by two different devices: VoiceInterface or PDAInterface.

The language is similar to the SCXML specification, but the actions and events are directly routed to the network.

4.5 Developed software.

The software developed can be divided into two distinct parts: the implementation of the statechart engine, and the user interface itself.

Nowadays there are too few SCXML implementations available, and the most notable effort is *CommonsSCXML* ("SCXML - Commons SCXML," 2010). Since *CommonsSCXML* is still in an 0.x version, and the need arose to extend its standard functionality, it was decided in a first approach to develop an SCXML engine from scratch. The application presented in this paper was developed in C# following the basic guidelines presented by Miro Samek in (Samek, 2002), and extended with the basic part of the SCXML language. Considering the W3C standard (Barnett et al., 2008), the implementation presented here does not include the *Extensions to the basic State Machine Model* and the *Executable Content.*

The application developed includes two main parts: on the left side of the graphical application form there is an UPnP generic control point that sniffs the network, and a log for UPnP dealings, like UPnP events, discovery notifications and subscriptions; on the upper right side of the form, a tree view of the statechart is presented and the correspondent graphical representation of the statechart. The design choice of keeping both the treeview and visual interface derived from several experiments where users mentioned the need of both interfaces to achieve a better comprehension of the orchestrated system.

The development of this application was developed in C# making use of the Windows Presentation Foundation (WPF) libraries. These libraries allowed the extensive data-binding techniques making this interface greatly extendable. It should be mentioned that some design choices make the graphical representation different from the original statechart formalism. These choices were motivated not only by practical reasons (programming time) but also based on the feedback provided by some early users. One clear example of this is the related with the deepness of the statechart, that is hidden: the user can see sub-states by double-clicking in the state. This permits a cleaner statechart that is more readable to the end-user.

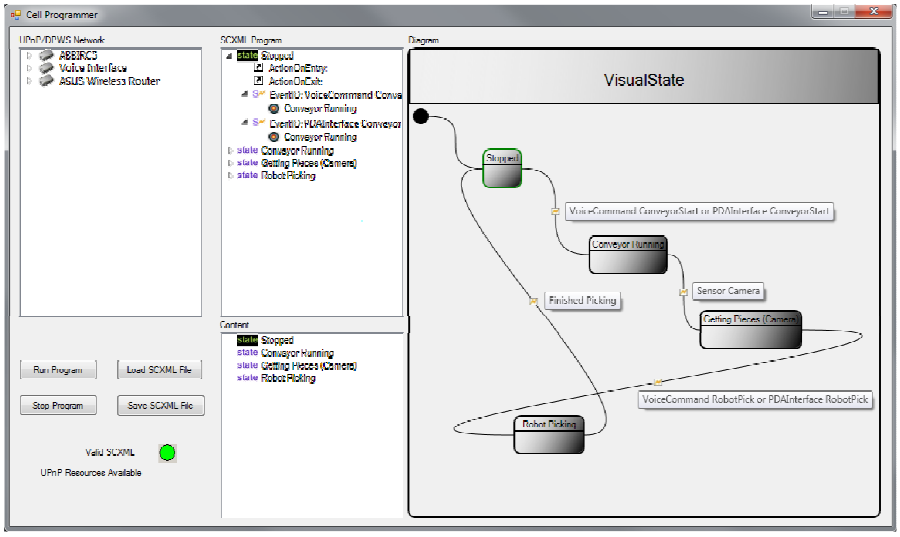

Fig. 8. Cell Programmer

The system is basically programmed in two steps. The first stage is the composition of the state machine via a context menu over the *treeview* that represents the statechart (Fig. 9).

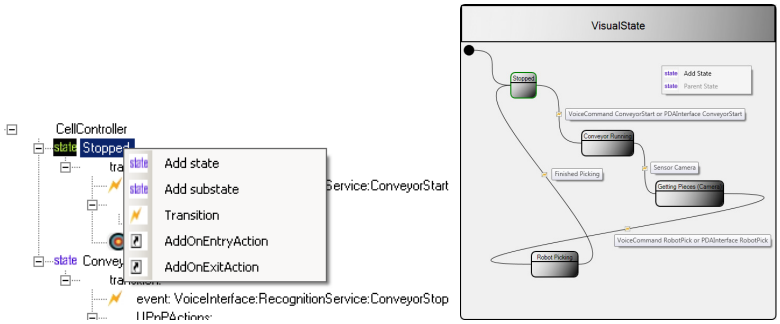

Fig. 9. State machine composition context menu

The second stage is fundamentally a drag'n'drop operation between the service network and the state machine (either to the treeview or to the graphics). UPnP actions can be dragged into state machine actions, either entry or exit, and UPnP events can be dragged to state machine events. The identification of the type of service uses the generic template mechanism described in Section 2 and concatenates device, service and event in way described in Listing 5.

```
urn:schemas-upnp-
org:device:VoiceInterface:1::RecognitionService::ConveyorStart
```

Listing 5. Conveyor start event specification

The UPnP discovery mechanism includes the definition of Hello/Bye messages as well as "keep alive" messages. Making use of these features, a validation of the program is performed in order to guarantee the presence of all the UPnP resources needed.

5 Conclusions and discussion

One of the most important SOA design principles is the loose-coupling and the good interface definition, mainly due to their importance in orchestration. The SOA vision foresees the application programmer as a client that looks at available services, selects some of them, and establishes orchestrations of services, like workflows e.g., that define process-based composite applications. To achieve this using BPEL4WS, an XML language with limited computation capabilities, the abstracted functionality of the service needs to fit the needs of the consumer. This has been a major challenge in the adoption of BPEL4WS.

As pointed out, the device-level orchestration was an open issue. The use of internet-derived tools on device-level SOA is not appropriate due to the reactive nature of industrial systems, which is not well described by process-based tools like BPEL4WS.
This work proposes an event-driven orchestration language that should be easily accepted by current robot programmers. This language, together with the automatic generation of services starting from robot programs, should give the robot operator the ability to integrate the robot in the network without any programming knowledge (besides the robot's one) and without compromising the flexibility of the system.

Several tests, including many laboratory developments, have used this platform as an orchestration basis. They range from the integration of new HMI devices like the Wiimote to the integration of stereo vision systems in robotic workcells. Some of these examples can be seen in videos present in http://robotics.dem.uc.pt/statechart/.

Another evaluation was made based on the feedback of a robot programmer, an R&D engineer from a system integrator, and an engineering student. The results are satisfactory and, comparing with the results obtained in (Veiga et al., 2008) with the Microsoft Visual Programming Language, the simplicity and the expressiveness of the statecharts, usually recalling SFCs, made them easier to learn. Another advantage noted was the separation of the cell logic from a specific device, creating a simpler way to store that information. Moreover, users claimed that the existence of a concrete cell program combined with the discovery features of the SOA platform will power a better reconfiguration experience. In such a system, the user would be able to select a given cell program, that in turn would "ask" the available services to verify if

all the necessary services were available in a compact and standardized way. Good reviews were also given to the software application like the one presented in Fig. 8.

This approach's main drawback, as described by some users, was the lack of some programming features: simple math operations, loops, conditional statements. These problems, very common in visual programming languages, are usually solved with the correct service definition, but that was not always the case in the first interaction during the tests.

Furthermore, the definition of an SCXML-derived language means that another SOA principle has been achieved in industrial automation: the orchestration's independence from the programming language. The current implementation of the statechart engine is made in C#, but there are tools available for Java, for instance.

The limitations of the study are: the lack of orchestration tests with a large number of services; the lack of formal evaluation of other event-driven techniques; and the lack of integration of the orchestration into higher layers of the business network.

References

ABB. (2008), *ABB - RAPID Instructions, Functions and Data types* (Technical Reference Manual), ABB.

"ABB WebWare Server - Software Products (Robotics)." (2008), . Retrieved January 12, 2009, from http://www.abb.com/

Barnett, J., Bodell, M., Burnett, D., Carter, J. and Hosn, R. (2008), "State Chart XML (SCXML): State Machine Notation for Control Abstraction.". Retrieved October 8, 2010, from http://www.w3.org/TR/2007/WD-scxml-20070221/

Browne, J., Hyder, S., Dongarra, J., Moore, K. and Newton, P. (1995), "Visual programming and debugging for parallel computing," *Parallel & Distributed Technology: Systems & Applications, IEEE*, Vol. 3 No. 1, pp. 75-83.

Bruccoleri, M. (2007), "Reconfigurable control of robotized manufacturing cells," *Robot. Comput.-Integr. Manuf.*, Vol. 23 No. 1, pp. 94-106.

Carriero, N. and Gelernter, D. (1989), "Linda in context," *Commun. ACM*, Vol. 32 No. 4, pp. 444-458.

Colombo, A., Jammes, F., Smit, H., Harrison, R., Lastra, J. and Delamer, I. (2005), "Service-oriented architectures for collaborative automation," Presented at the Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE, p. 6 pp.

Comau Robotics. (2005), *Comau PDL2 Programming Language Manual System Software*.

Delamer, I. and Lastra, J. (2006), "Ontology Modeling of Assembly Processes and Systems using Semantic Web Services," Presented at the Industrial Informatics, 2006 IEEE International Conference on, pp. 611-617.

Delamer, I. and Lastra, J. (2007), "Loosely-coupled Automation Systems using Device-level SOA," Presented at the Industrial Informatics, 2007 5th IEEE International Conference on, Vol. 2, pp. 743-748.

Dustdar, S. and Schreiner, W. (2005), "A survey on web services composition," *International Journal of Web and Grid Services*, Vol. 1, pp. 1-30. doi:doi:10.1504/IJWGS.2005.007545

FANUC Robot series Handling Tool (Operator's manual). (2008), .

FANUC Robotics SYSTEM R-J3iB Controller KAREL Reference Manual (Reference Manual). (2002), .

Foster, J. and Bryson, S. (1989), "Voice recognition for the IBM 7535 robot," Presented at the Southeastcon '89. Proceedings. Energy and Information Technologies in the Southeast., IEEE, pp. 759-764 vol.2.

Gelernter, D. and Carriero, N. (1992), "Coordination languages and their significance.," *Commun. ACM*, Vol. 35 No. 2, pp. 97-107.

Harel, D. (1987), "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, Vol. 8 No. 3, pp. 231-274.

"Home - Sirena 2003 - 2005." (2005), . Retrieved September 17, 2008, from http://www.sirena-itea.org/Sirena/Home.htm

"Home - Socrades - 2006." (2006), . Retrieved September 17, 2008, from http://www.socrades.eu/Home/default.html

Hopcroft, J., Motwani, R. and Ullman, J. (2000), *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*, Addison Wesley. Retrievedfrom http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0201441241

IEC. (2003), "IEC 61131-3, 2nd Ed. Programmable Controllers – Programming Languages," International Electrotechnical Commission.

Intel. (2008), "Intel® Software for UPnP* Technology.". Retrieved November 10, 2008, from http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/upnp/index.htm

Jammes, F., Smit, H., Lastra, J. and Delamer, I. (2005), "Orchestration of service-oriented manufacturing processes," Presented at the Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on, Vol. 1, pp. 8 pp.-624.

KUKA Documentation V5.x - Advance Programming (Reference Manual). (2006), .

Lastra, J. and Delamer, M. (2006), "Semantic web services in factory automation: fundamental insights and research roadmap," *Industrial Informatics, IEEE Transactions on*, Vol. 2 No. 1, pp. 1-11.

Lin, L., Wakabayashi, M. and Adiga, S. (1994), "Object-oriented modeling and implementation of control software for a robotic flexible manufacturing cell," *Robotics*, Vol. 11 No. 1, pp. 1-12.

Malec, J., Nilsson, A., Nilsson, K. and Nowaczyk, S. (2007), "Knowledge-Based

Reconfiguration of Automation Systems," Presented at the Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on, pp. 170-175.

Manning, C. and Schütze, H. (1999), *Foundations of Statistical Natural Language Processing*, The MIT Press. Retrievedfrom http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0262133601

Mensch, A., Depeisses, F. and Smit, H. (2007), *Technical Framework Description SODA*. Retrievedfrom http://www.soda-itea.org/Documents/objects/file1176731057.06

Microsoft. (2008), "SAPI.". Retrieved November 10, 2008, from http://research.microsoft.com/srg/sapi.aspx

Mital, D. and Leng, G. (1991), "A voice-activated robot with artificial intelligence," Presented at the Industrial Electronics, Control and Instrumentation, 1991. Proceedings. IECON '91., 1991 International Conference on, pp. 904-909 vol.2.

Motoman. (2006), *Motoman NX100 INFORM programming manual*. Retrievedfrom http://www.google.pt/firefox?client=firefox-a&rls=org.mozilla:en-GB:official

"OASIS Web Services Business Process Execution Language (WSBPEL) TC." (2010), . Retrieved December 18, 2008, from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

Pautasso, C. and Alonso, G. (2003), "Visual composition of web services," IEEE Computer Society, pp. 92-99.

Pires, J. (2005), "Robot-by-voice: experiments on commanding an industrial robot using the human voice," *Industrial Robot: An International Journal*, Vol. 32 No. 6, pp. 505-511.

Pires, J. and Sa da Costa, J. (2000), "Object-oriented and distributed approach for programming robotic manufacturing cells," *Robotics and Computer-Integrated Manufacturing*, Vol. 16, pp. 29-42. doi:doi:10.1016/S0736-5845(99)00039-3

Samek, M. (2002), *Practical statecharts in C/C++: Quantum programming for embedded systems*, CMP Publications, Inc.

"SCXML - Commons SCXML." (2010), . Retrieved January 11, 2010, from http://commons.apache.org/scxml/

"Semantic Interpretation for Speech Recognition (SISR) Version 1.0." (2010), . Retrieved November 23, 2010, from http://www.w3.org/TR/semantic-interpretation/

"SIARAS FP6 PROGRAMME PROJECT SUMMARY." (2008), . Retrieved December 4, 2008, from http://www.siaras.org/project.html

"SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)." (2010), .

Retrieved December 16, 2010, from http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

"Speech Recognition Grammar Specification Version 1.0." (2010), . Retrieved November 23, 2010, from http://www.w3.org/TR/speech-grammar/

Veiga, G. and Pires, J. (2008), "Using robot programming languages to specify service contracts," in*Proceedings of the CONTROLO'2008 Conference*, Presented at the Controlo 2008. Retrievedfrom http://www.google.com/search?hl=en&q=%22Proceedings+of+the+CONTROLO%E2%80%992008+Conference%22&start=10&sa=N

Veiga, G., Pires, J. and Nilsson, K. (2008), "Experiments with service-oriented architectures for industrial robotic cells programming," *Robotics and Computer-Integrated Manufacturing*, Vol. 25 No. 4-5, pp. 746-755. doi:doi: DOI: 10.1016/j.rcim.2008.09.001

"Voice Extensible Markup Language (VoiceXML) Version 2.0." (2010), . Retrieved November 24, 2010, from http://www.w3.org/TR/voicexml20/

"Voxware: Voice Software for Voice-Based Warehousing." (2010), . Retrieved November 22, 2010, from http://www.voxware.com/

Wareham, R. (1988), "Ladder diagram and sequential function chart languages in programmable controllers," Presented at the Programmable Control and Automation Technology Conference and Exhibition, 1988. Conference Proceedings., Fourth Annual Canadian, pp. 12A-14/1-4.

"Web Services Conversation Language (WSCL) 1.0." (2008), . Retrieved December 18, 2008, from http://www.w3.org/TR/2002/NOTE-wscl10-20020314/

"Web Services Flow Language." (2008), . Retrievedfrom pbfb5www.uni-paderborn.de/www/WI/WI2/wi2_lit.nsf/.../WSFL.pdf

"Yaskawa Motoman Robotics- MotoCom Software Development Kit." (2010), . Retrieved January 12, 2010, from http://www.motoman.com