PLUG-AND-PRODUCE TECHNOLOGIES
On the Use of Statecharts for the Orchestration of Service Oriented Industrial Robotic Cells

Germano Veiga and J. Norberto Pires
Mechanichal Engineering Department, University of Coimbra
Rua Luís Reis dos Santos, Coimbra, Portugal

Keywords: Service Oriented Architectures, SCXML, Industrial robotic cells.

Abstract: Programming industrial robotic workcells is a challenging task, namely because it means dealing with several types of machines, mange the data flow between them and orchestrate their basic functionality into a working program. In this work service oriented architectures are used for the task of programming robotic workcells along with managing the communication between cell components, and a statechart model engine is implemented to orchestrate the system logic. The objective of this paper is to focus in merging service oriented architectures with StateCharts XML, and in evaluating that robotic workcell programming approach using a simple laboratory test bed.

1 INTRODUCTION

The integration of different components in an industrial robotic cell is a time consuming task. Nowadays, industrial automation is using technologies originally designed for wider range of non-traditional industrial companies. The evolution of vision systems, 3D scanners, intelligent sensors PLC's, with their special languages and programming environments, etc., originated an enormous collection of interesting and powerful devices which are easier to program, although harder to integrate in their full extent. Consequently, porting plug-n-play concepts from PC's to the industrial automation environment is a promising opportunity. Even though similar to plug-n-play, in many aspects, the plug-n-produce concept (Veiga et al. 2007) has to deal with some specifics from the industrial automation world. One of these specifics is related with the presence of many highly programmable devices. Considering an industrial robotic cell, connecting a sensor to a robot controller can be compared to connecting a mouse to a PC, but integrating a programmable vision system or a PLC with an industrial robot requires some orchestration logic. To materialize this plug-n-produce concept regarding highly programmable devices, service oriented architectures (SOA) have been pointed as a promising approach (SMErobot™ 2007-2009). service oriented architectures are composed by autonomous services and are extensively event driven.

Finite-state automatons are very commonly used in the task of modeling the behavior of industrial automation systems. Due to their discrete event nature, industrial systems are well described by states and event driven transitions.

Harel StateCharts are a widely used extension to the finite-state automata model, and SCXML (Barnett et al. 2007) is a modern standardized way of specifying them.

The purpose of the current paper is to evaluate a SCXML-derived language to program service-oriented industrial robotics cells. This paper also presents a software application that materializes this concept, and the results obtained using a simple laboratory robotic test bed.

2 SOA - UPNP

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 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 which

Veiga G. and Norberto Pires J. (2008).
PLUG-AND-PRODUCE TECHNOLOGIES - On the Use of Statecharts for the Orchestration of Service Oriented Industrial Robotic Cells.
In *Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics*, pages 271-277
DOI: 10.5220/0001505102710277

supports the service is more business (or process) oriented (instead of being technology oriented). SOA enables software engineers to focus 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 inherent complexity (necessarily hidden from the user) associated with the presence of many types of devices and machines makes the concept of service-oriented architectures particularly attractive (SIRENA 2005, SODA 2006, SOCRADES 2006). In fact, it leads to the idea that each workcell programming block (that is, not only physical devices) should be considered as a potential device (SOA device style) that offers programming services.

Considering a *holonic* workcell structure (Gou, 1998), 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. 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.

Considering an industrial robotic cell ecosystem, past work (Veiga et al, 2007) revealed that service-oriented architectures can provide a suitable platform to a plug-n-produce environment.

Nevertheless, there are several approaches to SOA, namely, if we consider only the four most relevant platforms: Jini (Jini 2006), UPnP (UPnP, 2007), DPWS (Chan, 2006) and DSSP (Nielsen, 2007).

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

UPnP and DPWS rely extensively on standard network protocols such as TCP/IP, UDP, HTTP, SOAP, XML, and the 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 enterprise world with the pair WSDL/SOAP.

Although similar in many aspects, the UPnP and DPWS architectures use different languages for device description and different protocols for discovery and event notifications. There is an enormous dynamics around DPWS. Nevertheless, the new Microsoft operating system, Microsoft Vista, supports both technologies under the name plug-and-play extensions for Windows (PnP-X, 2006)

DSSP is a simple SOAP-based protocol that defines a lightweight, REST-style service model (Nielsen, 2007) that also relies extensively on web technology. Paired with concurrency and coordination runtime (CCR) it constitutes the major parts of the Microsoft Robotics Studio (MSRS) platform.

DSSP architecture style is radically different from the WSDL/SOAP model. UPnP and DPWS are very similar technologies which mean that concepts and design styles can be easily ported between each other.

In this work UPnP was selected due to representativeness of the platform, the quantity and quality of the tools available, and our experience with the UPnP based services.

3 SCXML

StateChart XML (SCXML) 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 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.

Harel StateCharts are an extension to finite-state automata. These extensions are needed in order to make finite-state automata useful, and they include:

Hierarchy – StateCharts may be hierarchical, i.e., a state may contain another statechart down to an arbitrary depth.

Concurrency – Two or more statecharts may be run in parallel, which means that their parent state is in two states at the same time.

History – A state holds information that allow a "*pause and resume*" behavior.

Consider for example the microwave oven model presented in Figure 1.

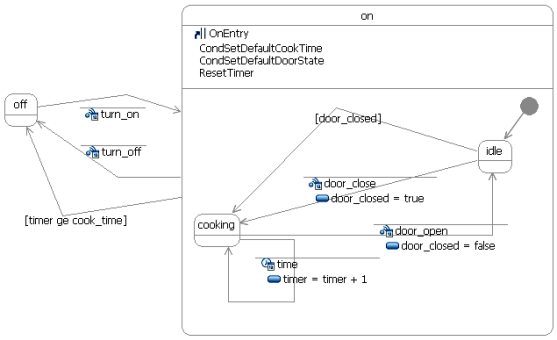

Figure 1: Microwave oven (Adapted from Barnett et al 2007).

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>
```

Has it can be seen in this example, an SCXML statechart can be divided in two major parts: the first composed by the machine states and correspondent transitions, and the other composed by the executable content.

The SCXML executable content consists of actions that are performed as part of taking 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's 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 additional executable content corresponding to special features.

4 EXPERIMENTS

4.1 Experimental Setup

The robotic cell used in this demonstration is composed of an industrial robot (ABB IRB 140), equipped with the modern IRC5 controller, a conveyor controlled by a PLC (Siemens S7-200) and a USB web camera (Figure 2).

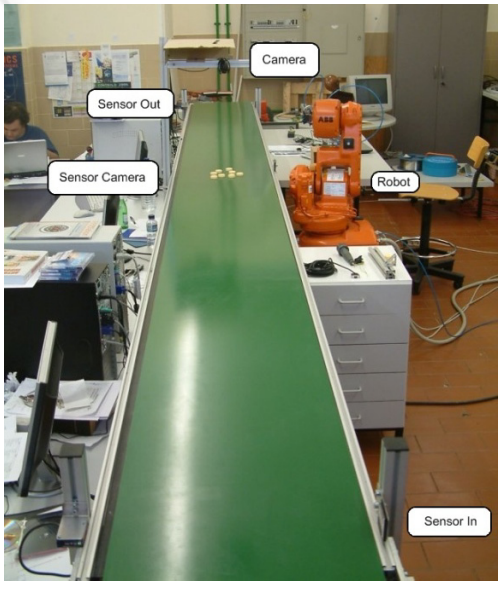

Figure 2: Equipment for experimental setup.

Basically, the conveyor transports sample pieces over the machine vision system which calculates the

number and position of the pieces. The results are sent to the robot controller to command the robot to pick them from the conveyor and place them into a box.

Two different applications were developed to operate the cell: a speech interface and a PDA interface.

4.2 Previous Work

In previous work (Veiga et al, 2007) this Industrial Robotic cell test-bed was used in order to validate SOA in a plug-n-produce environment. The cell components were represented in the network by UPnP devices that provided services as way to expose their functionality (Figure 3).

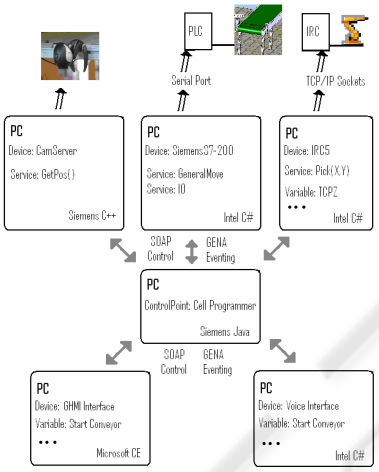

Figure 3: UPnP Network of the industrial test-bed.

Five UPnP devices representing five workcell components were developed. In some of these devices an extra layer was needed because native UPnP support could not be implemented. Detailed description can be found in (Veiga et al, 2007).

The *Cell Programmer Interface* (Figure 4) is a software application developed to control the flow of high level tasks in a manufacturing cell. Basically, it's an UPnP *control point*, with some tools suitable to build a generic stack. This stack represents the control flow of process related tasks. In the left side of this interface a tree shows all UPnP devices founded on the network.

Clicking over them it is possible to get additional information (access the presentation page, for example). Using the "arrow" button, actions or events are added to the stack. Furthermore, when running the resulting program and the program counter is pointing to an event, it means that the program is "waiting" for that event to occur. Inversely, if the program counter is pointing to an action, it means that it is calling that action and waiting for the return. There is also the possibility of defining auxiliary variables to store values that can be used as arguments in later stack steps.

4.3 Analysis

The simple stack approach revealed to be very limited to handle more complex systems. These systems often have concurrent tasks, multiple transition events and many other orchestration requirements which are impossible to model with a simple stack system.

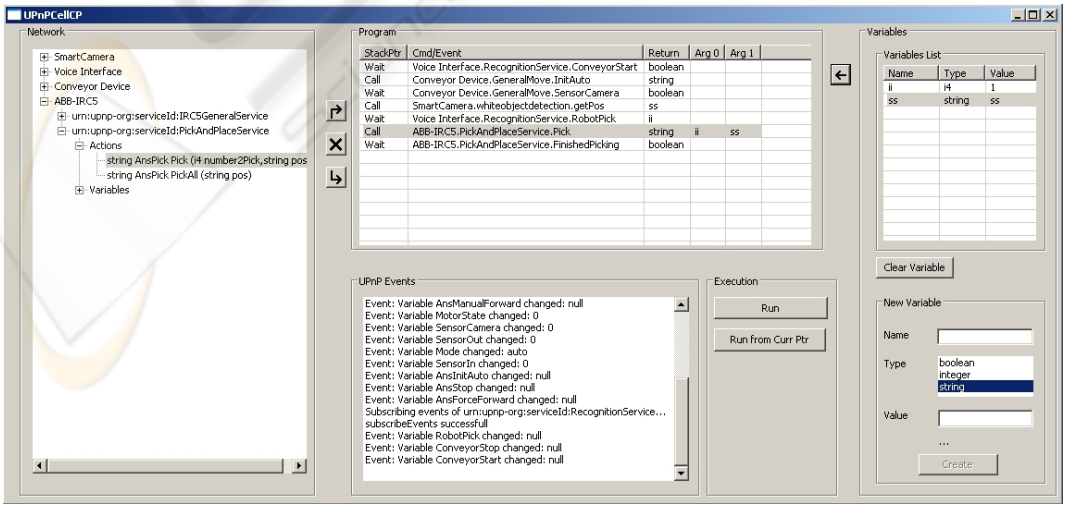

Figure 4: Simple orchestration system (Veiga et al, 2007).

To address this problem, this paper proposes a more powerful orchestration model that relies on standard technologies (SCXML) and solves many of the described problems.

Another issue with the previous approach was related with the use of the auxiliary variables and the need to check the presence of remote functionality on distributed systems. A robot cell program is only valid if all the services needed are available, but also if the used variables have the desired value. If this value is obtained by previous steps it's not possible to check if that value is still valid (corresponding to a live device, for example). To address this issue the solution proposed only relies on UPnP state variables to manage data, discarding the use of auxiliary variables

4.4 Implementation

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

4.4.1 StateChart Engine

Actually there are too few SCXML implementations available, and the most notable effort is *CommonsSCXML* (CommonsSCXML 2007). Since *CommonsSCXML* is still in a 0.x version, and there was the need to extend the standard functionality, it was decided here to develop an SCXML engine from scratch.

The application presented in this paper was developed in C# following the guidelines of (Samek, M. 2002) including the basic part of the SCXML language. Considering the W3C standard (Barnett et al. 2007), our implementation doesn't include the *Extensions to the basic State Machine Model* and the *Executable Content.*

This approach was taken not only for simplicity but also with the objective of keeping the cell program as simple as possible. This objective of simple orchestration programs is sustained by the *holonic* cell structure referenced earlier, where devices expose high-level functionality services and the cell orchestration programs are reduced in terms of flow control, managing data etc. As such, all executable content within the cell orchestration program is always related with processing UPnP events or UPnP actions.

4.4.2 User interface

The *UPnPSCXMLCellProgrammer* is a graphical user interface that allows the composition of workcell orchestration programs relying on UPnP devices and the SCXML derived model (Figure 5). To behave like this the StateChart engine has the added capability to recognise whether an event or executable content are UPnP or not.

When the engine gets an UPnP event it converts it into an SCXML event. When the statechart engine finds an UPnP action inside an executable content block it just calls that action with the parameters (UPnP arguments) enclosed.

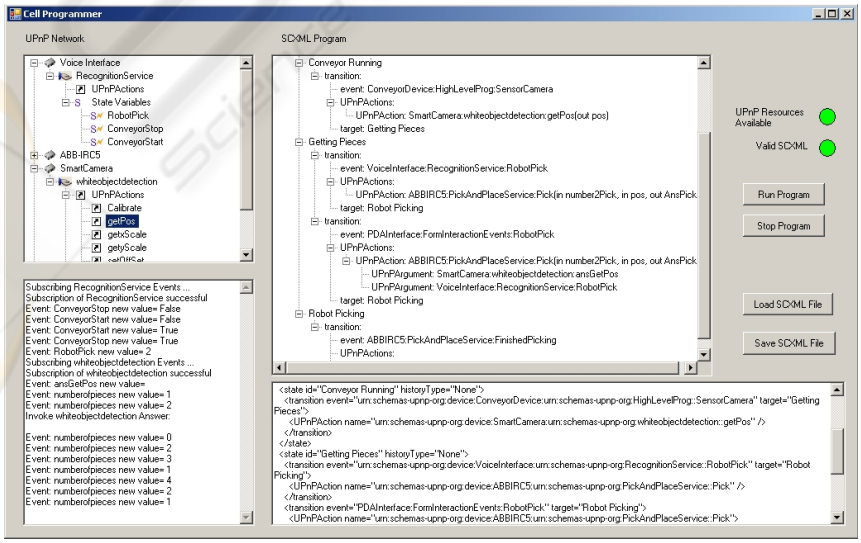

Figure 5: UPnPSCXMLCellProgrammer Interface.

In this application the user can drag and drop UPnP actions and UPnP events from the UPnP network and place them into the SCXML program. It's worth noting that the events in this program are always network events, and that only UPnP actions can be assigned to the *OnEntry* and *OnExit* handlers and to the executable content of the transitions, which are executed between the *OnExit* handler of the source state and the *OnEntry* handler of the current state.

UPnP events and UPnP actions are defined by their complete Unified Resource Name (URN) that includes the name of the action or service, plus the URN of the owning service, plus the URN of the device.

5 EXPERIMENTS AND RESULTS

The test bed used to experiment this new approach that merges UPnP with SCXML is the same used previously (Figure 2). The program logic is a little bit more complex with the objective to show some of the new possibilities (Figure 6).

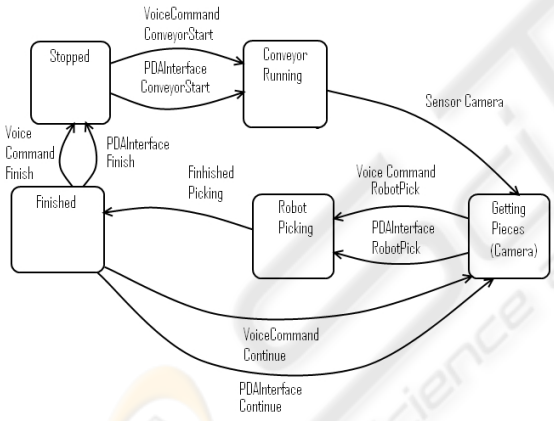

Figure 6: Program statechart.

In fact, the new program logic allows the user to give alternatively speech commands or PDA commands, and re-picking leftovers before asking the conveyor to run again.

In comparison with the previous situation much more complex orchestration schemes can be obtained. Statecharts provide a very nice way to model systems logic but are very limited when dealing with data processing. With this combined approach pairing statecharts with SOA all data processing is made within SOA, leaving statecharts for modeling systems states and logic. Considering the experiments made so far we can point statecharts as a suitable model to orchestrate *holonic* automation workcells, due to their capabilities in dealing with events and states in opposition of dealing with data processing.

6 CONCLUSIONS

This paper reports results from an ongoing research work. Experiments done so far revealed that the added features enable the cell programmer to define powerful and more complex orchestrations that can handle complex systems.

Consequently, merging service oriented architectures with statecharts proved to be an interesting approach to model the workcell orchestration logic. Future work will focus on the evaluation of this approach with more complex systems and in providing a more user friendly graphical interface.

ACKNOWLEDGEMENTS

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

REFERENCES

Abb, 2005 ABB IRC5 Documentation, ABB Flexible Automation, Merrit, 2005

Ahn S. C., Kim J.H., Lim K., Ko H.,Kwon Y and Kim H., 2005 UPnP Approach for Robot Middleware *P Proceedings of the 2005 IEEE International Conference on Robotics and Automation Barcelona, Spain, April 2005*.

Barnett, J. et al, 2007. State Chart XML (SCXML): State Machine Notation for Control Abstraction. http://www.w3.org/TR/2007/WD-scxml-20070221/

Chan, S., Conti, D., Kaler, C., Kuehnel, T., Regnier, A, Roe, B., Sather, D., Schlimmer, J., Sekine, H., Thelin, J., Walter, D., Weast, J., Whitehead, D., Wright, D., and Yarmosh, Y. (2006). "Devices Profile for Web Services." http://schemas.xmlsoap.org/ws/2006/02/devprof/

CommonsSCXML, 2007, available from: http://jakarta.apache.org/commons/scxml

James, F. and H. Smit ,2005 Service Oriented Paradigms for Industrial Automation. In: *IEEE Transactions on Industrial Informatics, Vol. 1, no. 1 February 2005*.

Gou L., P. Luh, and Y. Kyoyax (1998). Holonic Manufacturing Scheduling: Architecture, Cooperation Mechanism, and Implementation. *'97., IEEE/ASME International Conference on Advanced Intelligent Mechatronics* Vol 37, 213-231,

Harel D. 1987. StateCharts: A Visual Formalismfor Complex Systems. *Science of Computer Programming* 8, North Holland.

Nielsen, H. and G. Chrysanthakopoulos. (2007) Decentralized Software Services protocol – DSSP/1.0

PnP-X (2006): Plug and Play Extensions for Windows Specification. Available: www.microsoft.com/whdc/Rally/pnpx-spec.mspx.

Samek, M. (2002). *Pratical StateCharts in C/C++*, CMPBooks

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

SIRENA Project (2005), Service Infrastructure for Real-time Networked applications, Eureka Initiative ITEA. Available: www.sirena-itea.org.

SOCRADES. (2006). "Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded devices." http://www.socrades.eu/

SODA. (2006). "Service Oriented Device and Delivery Architecture." http://www.soda-itea.org/

UPnP forum (2004).Available: http://www.upnp.org

Veiga G., Pires JN, Nilsson K.. On the use of SOA platforms for industrial robotic cells: Intelligent Manufacturing Systems Proceedings IMS2007, Spain, 2007

SMErobot™ (2007-2009), The European Robot Initiative for Strengthening the Competitiveness of SMEs in Manufacturing, www.smerobot.org