Event Router-Scheduler for the Modular Anatomy of Service-oriented Automation Components

J. M. Mendes[1], J. de Sousa[2], P. Leitão[3], A. W. Colombo[4], F. Restivo[1]

[1]Department of Informatics Engineering, Faculty of Engineering - University of Porto, Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal

[2]Department of Electrical and Computers Engineering, Faculty of Science and Technology, New University of Lisbon, Quinta da Torre, 2825-114 Caparica, Portugal

[3]Department of Electrical Engineering, Polytechnic Institute of Bragança, Quinta Sta Apolónia, Apartado 1134, 5301-857 Bragança, Portugal

[4]Schneider Electric GmbH, Steinheimer Str. 117, D-63500 Seligenstadt, Germany

Abstract

Automation and production systems are evolving in the direction of autonomous and collaborative components, approaching the idea of an ecosystem. A single habitant of this system is responsible for different and concurrent activities and thus it requires a special adapted anatomy that is balanced for the several requirements. This work introduces an anatomical-like structure for the development of functional and reusable modules of service-oriented automation components. The central attention will be given to their internal structure and the mechanism that bind the modules together, called the Event Router-Scheduler. The resulting software automation components are customized for different tasks due to the inclusion and management of the specialized functional modules and provide the ability to operate in a service-oriented automation and production environment.

Keywords:

Service-oriented Architecture, Distributed and Component-based Automation and Production Systems

1 INTRODUCTION

In distributed automation and especially in the branch of production systems, the set of equipment and other components in the system may be comparable under some circumstances to a society of living beings. Taken a closer look into a component itself, its internal mechatronical organization may correspond to functional organs that are responsible for specific tasks, providing the "vital" properties to be able to fulfill its requirements. A central question is how these functional modules or “organs” may be integrated, controlled and able to pass impulses between them and therefore to form complex and operational structures.

Service-oriented systems are one approach to specify the environments for heterogeneous “organisms” that require interaction. Service ecosystems are well known in the field of business and electronic commerce (see [1,2]) but in industrial automation and production systems (especially concerning distributed devices) it is a relatively new research area with promising results. One of the major starting points was the EU Research Project SIRENA [3] which goal was to develop a service infrastructure for real time embedded networked applications. The implementation of the service framework was developed in conformance to the Device Profile for Web Services (DPWS) specification to permit the communications between resource constrained embedded devices [4]. Since then, other projects are centering in service-oriented devices and supporting environment, such as the EU Research Project SOCRADES [5].

An important issue for these systems is where the control is located and how its granularity and distribution are affected. Since the introduction of the common Programmable Logic Controller (PLC), significant effort has been done in research and development to overcome the PLC’s limitations in terms of centralized usage. Solutions to this challenge may come from different directions such as Multi-Agent Systems [6, 7], Holonic Systems [8-10] and recently Service-oriented principles [3,11].

Architectures for devices and control software have also been focus of research, commonly dealing with the IEC61131-3 languages [12-15], the IEC 61499 function blocks for distributed control and automation [16-18] and other control techniques such as Petri Nets [19, 20]. For service-oriented distributed devices, the control method is partially open to any control approach, but should also consider specific requirements of service-orientation. Serving different functionalities, a single service-oriented control device should be ready for multiple activities, and thus requires a special adapted internal framework that handles differentiate and concurrent processes.

This work introduces an anatomical-like structure for the development of functional and reusable modules of a component, part of a service-oriented automation system. The central focus will be directed to its internal structure and especially to the mechanism that tie functional modules together, called the Event Router-Scheduler (ER-S). The ER-S can be compared to the nervous system of living beings in sense of carrying impulses from and to different organs and so maintaining the dynamic information flow. Intelligent behavior can be reached when these nerves are linked to the “brain”, that provides static control based on workflow processes and also autonomy to respond to unexpected events, undocumented situations and internal objectives. Being inserted in a service-oriented environment, interaction with other components is achieved only by providing and requesting services to reach local and global objectives.

The paper is organized in the following way: after the introductory notes, Section 2 describes the scope and domain of the work, namely the environment of Service-oriented Automation Components, and Section 3 resumes the internal anatomy of the components. The Event Router-Scheduler is explained in section 4 and a prototype implementation and operation is presented in the Section 5. Finally, Section 6 rounds up the paper with conclusions.

2 THE SOCIETY OF SERVICE-ORIENTED AUTOMATION COMPONENTS

Production and automation systems are heterogeneous in nature, made of different components with distinguished roles. It is therefore predictable that the specifications of those systems are moving from the traditional central-controlled manner to the corresponding distributed counterpart, assimilating the natural appearance and layout of the real system.

Thus, one promising guideline in this respect is to have a conglomerate of distributed, autonomous, intelligent, fault-tolerant, and reusable manufacturing units, which operates as a set of co-operating entities. Each entity is capable to dynamically interact with each other to achieve both local and global manufacturing objectives, from the physical/machine control level on the shop floor to the higher levels of the factory management systems [8]. This new generation of systems is referenced as Intelligent Manufacturing Systems (IMS) [21].

One of the rising solutions to adapt the majority of the concepts behind IMS into feasible principles is Service-oriented Architectures (SoA). The concept SoA has gained significant attraction in just a few years and will undoubtedly have a major impact in many branches of technology. According to [4], "*A service-oriented architecture is a set of architectural tenets for building autonomous yet interoperable systems.*" and this proposal is facing one of the challenges of IMS, namely providing interoperability between autonomous systems.

Adapting the service-orientation concepts to the automation and production "ecosystem" at the shop floor and considering the principles of IMS, a "society" of service-oriented automation components is born. Each participant in the system is referred as Service-oriented Component and in some extends, Service-oriented Automation Component (when it has automatic control duties). Components may have different roles (e.g. production, transportation and monitoring) and operate autonomously. Since services are the main guide, these components should have the need of requesting services and also the desire in providing services to the community. Services itself are a form of providing resources and actions that are shared in some circumstances, much similar to the real-life services.

Fig. 1 shows the basic description of a Service-oriented Component and its integration into the environment of automation and production shop floor. The given example is a component that represents a physical conveyor (Mediator of: Conveyor) and has the transportation role (Role: Transportation). Implicitly, the communication to the outside world would be via services (Orientation: Services), being able to provide and request services when needed. The integration into the IT-enterprise is also reached by the service-orientation. A component has a set of tasks or activities (Tasks: Transport, Monitoring, etc.) and those may be used as services provided by the component.

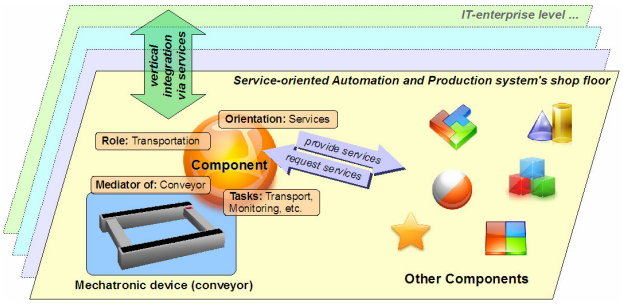

Figure 1: Service-oriented Automation Component and its environment.

Interaction between components is done by the two-way service orientation, in sense of requesting and providing services. It is expected in production and automation that heterogeneous components work together for mutual benefit and global objectives. This can be distinguished as *symbiosis*, similar to the interactions between different biological species [22]. It is also possible that components may compete with each other for resources (services), but in the end the global goal must be respected.

In some situations Service-oriented Components can be seen as software agents according to the definition given by Schoop et al. [7], adapted from Jennings and Wooldridge [6], to flexible production systems:

"An agent is considered a software entity situated in a flexible production environment, with enough intelligence that is capable of autonomous control actions in this environment and of co-operation relationships by participating in associations' agreements with other entities in order to meet its design objectives".

Moreover, Multi-agent Systems (MAS) [23] are of special interest since these systems bring the idea of collaborative agent society, in which each of them can take autonomous actions over their environment or over the system that they represent. On the other hand and differentiating from the agent concepts, the true meaning of service-orientation is centered in the requirement of providing services and in the necessity of requesting services by a component in the system. The real architecture, habitat and objectives of the system are truly open to the developer and thus it may adopt different strategies to cover the requirements. For more information about some discussion points and applications of MAS with SoA, the user can consult the following documentation: [24-27].

The specification of an internal anatomy of components that may simplify their development remains an open issue. Since Service-oriented Components may have different and sometimes concurrent activities, it may be useful to consider a functional and independent structure in favor of reusability and easy development. The following section describes an anatomical-like structure to develop service-oriented components.

3 INTERNAL ANATOMY OF COMPONENTS

Each Service-oriented Component may be implemented independently and differently. The only requirement is that it should share its functions as services and obey to the protocols of communication and processes. To be able to construct and deploy these components in a simple but functional way, an anatomical-like framework was specified.

A general component is structured in an anatomical form comprising several "organs" (functional modules) that are responsible for individual tasks, as illustrated in Fig. 2: Logic Controller, Decision and Exception Handler, Communication, Device Interface and Event Router-Scheduler. These modules are included in the control component according to its needs and possibly implemented using different technologies. It is also possible to develop and integrate other modules for diverse functionalities, if they respect the rules provided by the framework for the integration (task of the Event Router-Scheduler).

The Event-Router-Scheduler and Communication modules are the kernel modules to develop a Service-oriented Component based on the proposed anatomy. They are responsible, respectively, for the main framework of the component (event-based inter-module

communication and integration) and external communication with other components (service-oriented inter-component communication). Other modules may be added to the structure according to the component's requirements.

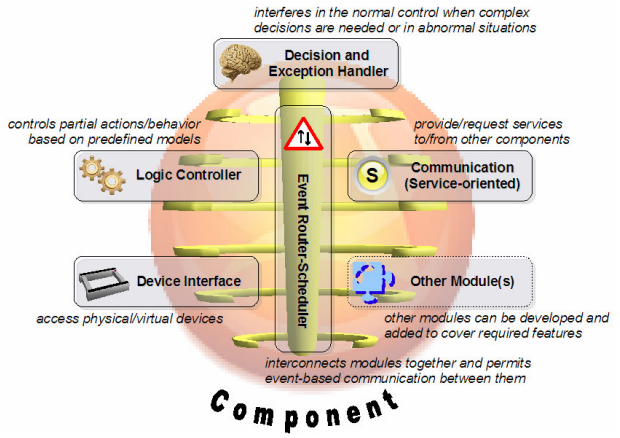

Figure 2: Concept of a modular anatomy for a Service-oriented Component.

In more detail, the Communication module provides the necessary functions to expose the services from the associated component and request services from other components. Other functions include, among others, discovery and negotiation mechanisms. The remaining modules of the component may use the Communication module to access these functions through impulses (events) provided by the Event Router-Scheduler module. As an example, a conveyor may provide the *Transfer* service to handle the movement of pallets, which is controlled by the Logic Controller module and accessed by the Device Interface module. The *Transfer* service may be used by the other components, but the component itself can also call external services when needed (e.g. to be connected to other conveyor it requests the *Transfer* service of another conveyor) [28].

A suitable technological solution to implement the service-oriented communication module is to use Web technology, and most specifically Web services. At its core, Web services technology is quite simple and it is designed to move XML (eXtended Markup Language) documents between service processes using standard Internet protocols. This simplicity helps Web services to achieve the primary goal of interoperability and also means that it is necessary to add other technologies to build complex distributed applications. A profile has been specified for adopting Web services at the device level known as Device Profile for Web Services (DPWS) [4].

The remaining modules are described briefly in Fig. 2. The goal to include the other modules is to provide an example of a Service-oriented Automation Component that is mediator of some physical equipment with control capabilities. For example, the resulting component of Fig. 2 represents a smart controller of a conveyor device, by providing several features such as control and access over the physical device, ability to decide in unexpected and undocumented situations and also the possibility of service-oriented communication to other components. Other example is a service-oriented PLC-like controller, which may interpret control models (e.g. in IEC61131-3 languages) and give the necessary orders to other components via the invocation of the provided services by them. In this case, it is not necessary to have the Device Interface module, since it does not command directly the devices.

Finally, the "nervous system" of the anatomy represented in Fig. 2 is managed by the Event Router-Scheduler. More detail to this module follows in the next section.

4 THE EVENT ROUTER-SCHEDULER MODULE

Components and devices that implement several of the expressed aspects of service-orientation require a consistent anatomy to deal with the different function modules ("organs") in order to fulfill the necessary requirement. Other problems may arise from the asynchronously operating modules, possible data inconsistencies and concurrent processes/threads. For this purpose, it is proposed a mechanism to provide an "impulse" (event) passing and scheduling feature to guide the impulses to different modules, thus permitting the synchronized communication between them. The heart of the component is the Event Router-Scheduler (ER-S) module.

During the design phase it was clear that the ER-S should meet the following objectives:

- Common event routing/scheduling mechanism for the communication and integration of modules;
- Provide some transparent functions for creating and managing modules;
- Suitable for software application that are deployed both in traditional PC and embedded systems;
- High performance, especially in critical situations and targeting real-time applications;
- Use of C language, aiming to balance between performance, portability and features;
- Tread safety and management of data concurrency;
- Easy to use by developers, in sense of building modules and how events are processed.

The function of the ER-S is comparable in some parameters to the nervous system of living beings, including humans. H. Gray wrote in his book "Gray's Anatomy of the Human Body" [29]:

"The Nervous System is the most complicated and highly organized of the various systems which make up the human body. It is the mechanism concerned with the correlation and integration of various bodily processes and the reactions and adjustments of the organism to its environment."

In the case of Service-oriented Components, the "environment" is captured and manipulated by specific modules (e.g. Communication and Device Interface), but the natural equilibrium with impulses (events) of the several modules and their integration is reached with the help of the ER-S.

Fig. 3 shows the generic conceptual structure of the Event Router-Scheduler. The feature groups are separated in blocks that correspond to the Scheduling and Routing of Events, Hardware/Software Abstraction, Threading and Data Consistency and Template/Interface for Event-based Modules.

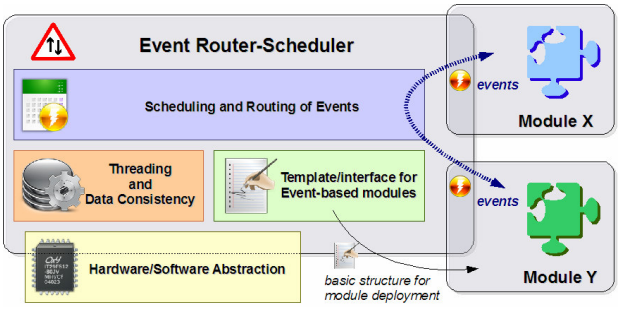

Figure 3: Structure of the Event Router-Scheduler.

The main feature is to provide event-based communication between functional modules and the corresponding routing and scheduling of events (see

Scheduling and Routing of Events block of Fig. 3). From the practical point of view, the component's internal impulses (events) between its functional modules are integrally managed by the ER-S. The ER-S allows synchronous and asynchronous event calling between any modules (which is extremely important in real-time applications), and offers several additional procedures to realize more complex operations, like events generated by other events and time-triggered events. In the most basic form, a sender module must only emit an event to a specific destination (other module) and the ER-S routes it to the destination. There are also other options for sending and processing events, such as events with reply and multicast events to several destination modules (see Fig. 4).

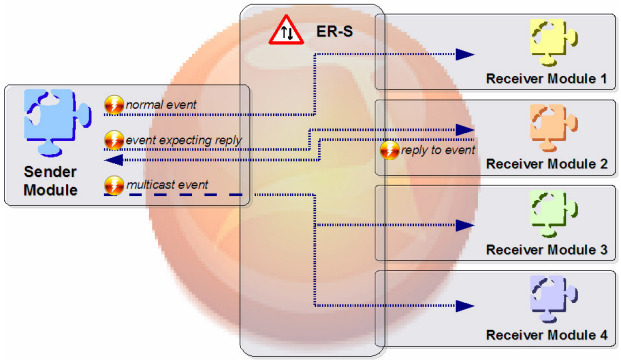

Figure 4: Different types of sending events by a module through the ER-S.

An event is a structure with all the information a module needs to know regarding various possible situations. Besides the standard information as the pretended action and the parameters from who the event came, it can ask for a reply, for an information forwarding, can have a fault message's receiver, and the event's receiver can check if it is a reply. Also, an event sent more than once, by error, is detectable.

The ER-S uses lists as a way of transmitting and queuing events between the modules, so the number of events waiting to be processed is only limited by the available memory. The ER-S uses some techniques to avoid memory fragmentation, because the creation and elimination of new data is a very frequent operation in the modules, as the world is constantly changing. In some cases when the number of events is high, the ER-S offers the possibility to give different priorities to the events. Like this, an event sent to a certain module will always pass by all the waiting events of that module which have lower priority than the sent one.

Being capable of both synchronous and asynchronous operations, the asynchronous ones are managed using threads. The synchronous operations can be either freezing or non-freezing for the receiver. For example, on event reading the operation can be a module-freezer or not. In case of the freezing mode, the module freezes until any event arrives for it. After that, the module continues its normal proceeding, as shown in Fig. 5(a). This is a very low CPU resource-taking procedure, useful for embedded devices. However, it is not useful for real-time multi-task modules, as this kind of module should not freeze. On the other hand, the non-freezing event-reading always receives an event. However, it can be an invalid event. An invalid event means that there were no events for the module, so it can continue its other tasks. Obviously, if it is a valid event, the module should process it. This is represented in Fig. 5(b).

Asynchronous event triggering is also possible. Callbacks are used to perform this type of operation, as it must occur when it is called. However, the event is not triggered immediately, because of data-protecting, and it should only occur when the module activates an authorization (*mutex*) to allow callbacks, which will possibly change the module's data. Each module has its *mutex* for this matter, and developers who want to enable asynchronous event handling should be very careful with this protection.

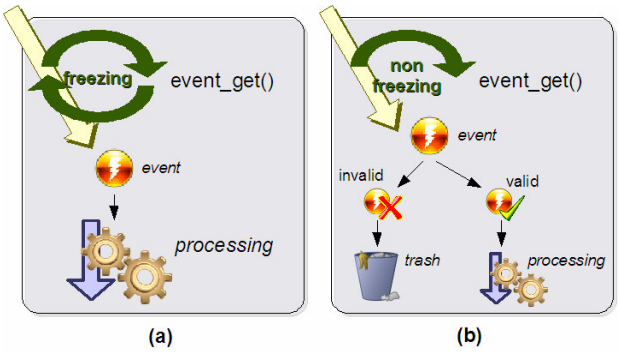

Figure 5: Event reading: (a) module-freezing (b) non-freezing.

The remaining blocks of Fig. 3 are responsible for adjacent tasks of the Scheduling and Routing of Events block, specifically to its and other modules' management. The Hardware/Software Abstraction provides some functions transparent to the system architecture that can be accessed by all modules. Since the ER-S and other modules are in a multi-functional and concurrent environment, a special block of the ER-S, namely the Threading and Data Consistency block, introduces simple thread manipulation and data protection (such as *mutex*).

Finally, the Template/Interface for Event-based Modules block provides the basis for creating functional modules and associates them to the ER-S. Each module can be programmed independently. This means that it is possible to remove, replace, upgrade or add new modules. This makes a program using the ER-S very flexible. The module ID is the module's identification and it is unique for each module. This variable is what the other modules need to know to send an event to a specific module. It is comparable to the code that the nerves carry to reach some organ. However, it is also possible to search a module by its type like "controller" or "user interface", as this way is much more practical for a developer to reach a module without many information.

5 IMPLEMENTATION AND OPERATION

A prototype implementation has been done to test the proposed framework, integrally coded using the C programming language and compatible with Windows and GNU/Linux operating systems (targeting also others, such as VxWorks). Some implementation details are given next.

The functions provided by the framework to develop and operate components are explained with an example component representing a mechanical arm (articulated robot to move small objects) made of three modules (besides the ER-S), represented in Fig. 6. The modules correspond to a subset of the ones in Fig. 2 (excluding the Decision and Exception Handler module) that are briefly commented in section 3. The major difference is that it is connected to the mechanical arm via the Device Interface module, instead of the conveyor of Fig. 2.

In terms of data structures, the ER-S includes several structure types for storing and relating different information about modules, events and other aspects. The Module Structure, which represents a module in the program, identifies its module by a unique ID. It also provides storage for local information such as the

module's incoming events list, which is where the module is going to get the events sent by the other modules and a pointer to the module's callback implemented function, which is triggered by new events when the asynchronous mode is activated. The Event Structure has all the information to handle an event: action name, parameters, who is sending/sent it (module ID), and some variables for reply handling, an ID of the event and ID of the reply. Finally, the Database Structure of the ER-S is where pointers to all modules are allocated.

First the modules must be created. Thus, the respective function shall be called, and each module must have an ID and type, as seen below:

```
module_create(1,DEVICE_INTERFACE);
module_create(2,LOGIC_CONTROLLER);
module_create(3,COMMUNICATION);
```

Sending and receiving events is very straightforward. For example, to send an event from module 1 to module 2, the developer must create an event, put the sender, the action and the parameters, and then send it with low or high priority, to the destiny, using the *event_send()* function. To read an event, presuming that callbacks are disabled, module 2 must call the synchronous event handle by either freezing while there are not events, or not freezing. This variable is a parameter when calling the event-reading function, as it can be something like *event_get(2, FREEZE)* or *event_get(2, NO_FREEZE)*.

component provides one service, *Transfer*, to be called externally in case objects are available to be transported. Finally, the Process Controller module is responsible for coordinating the components activity, generally synchronizing service calls with the pick & place program execution of mechanical arm.

A simple algorithm is presented in Fig. 6 inside the Logic Controller module. Each time a function is required by one module to another one, events are sent through the ER-S. In case of the algorithm of Fig. 6, an instance of it is executed when the Transfer service is requested and then the Communication module of the component emits an event to the Logic Controller. It is assumed that the *Transfer* service is called when an object is ready to be moved. From the other hand, the operation of pick & place program can only be started if the mechanical arm is not occupied and if the destination where to place the object is free. For the sake of simplification, these checking functions are represented in the algorithm but their behavior is absent in Fig. 6., which would involve sending/receiving events to/from the Device Interface and possible also an entity representing the destination place. On successful conclusion of the pick & place program of then Device Interface, an event is sent back to the Logic Controller, and by its turn to the Communication module that then notifies the external component and thus concludes the service usage.

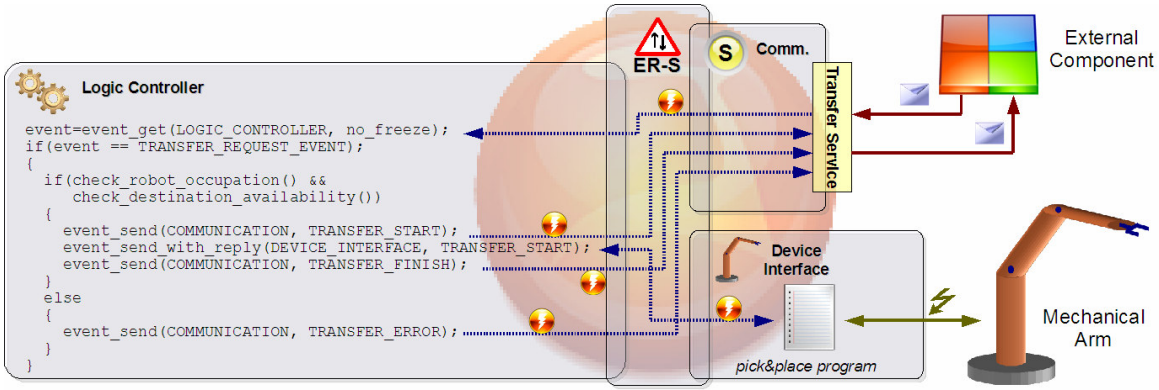

Figure 6: Example component of a mechanical arm and its operation.

The not-freezing way of getting an event always returns an event, but it may be an invalid event. On this case, valid events always have valid senders, this is, the *from* variable, which corresponds to the sender ID, is always bigger than zero. So, invalid events have negative sender IDs. If the module 2's callback is *ON* instead, and if the callback mutex allows it, the new event would immediately trigger the callback, so it would run the function pointed on the module 2's structure.

More flexible operations can be done with multicasting and reply to events. In case of multicasting, there is a special function to emit an event to several destination modules: *event_send_multicast()*. One of the parameters is a list of destination modules that are intended to receive the event. Some events may expect replies and this can be done in two ways: asynchronously (non-freezing) using the *event_send()* function with the attribute *reply_id* and synchronously (freezing) using the special *event_send_with_reply()* function.

For the example, the modules of the mechanical arm component have simple functionalities. The Device Interface provides the access to the mechanical arm in sense of calling the programs of pick & place to move the objects from one place to another. Its Communication module uses a service-oriented infrastructure, described in [30], based on a DPWS (Device Profile for Web Services) implementation, namely SoA for Devices (SOA4D). Through the communication module, the

6 CONCLUSIONS

This paper presents a framework for developing service-oriented automation systems, specially dedicated to the mechanism of passing events between the different functional modules that build up the component. The adoption of this "bio-inspired" modular structure makes possible to design and develop modules with distinct and independent functions but complementary to each other, forming complex, intelligent and social components. The resulted component's structure may help in decreasing the development time and effort in the integration into the system. The prototype development shows the feasibility and features of the concept, providing the possibility to develop reusable and functional modules and deploy them into service-oriented components.

Future work is to enhance both concept and development, and also provide a case scenario based on real equipment that represents a production system. A special case is to enhance the flexibly in the deployment of components and its modules, by developing a specification of metadata for modules that would permit the creation of them without worrying about how the information comes from the other modules.

7 ACKNOWLEDGMENTS

The authors would like to thank the partners of the Innovative Production Machines and Systems (I*PROMS) Network of Excellence (http://www.iproms.org) and the EU project SOCRADES (http://www.socrades.eu), for their support.

8 REFERENCES

[1] Barros, A., Dumas, M., 2006, The Rise of Web Service Ecosystems, IT Professional, 8: 31-37

[2] Sawatani, Y., 2007, Research in Service Ecosystems, International Center for Management of Engineering and Technology, Portland, 2763-2768

[3] Jammes, F., Smit, H., 2005, Service-oriented architectures for devices - the SIRENA view, 3rd IEEE International Conference on Industrial Informatics, 140-147

[4] Jammes, F., Mensch, A., Smit, H., 2005, Service-oriented device communications using the devices profile for web services, Proc. of the 3rd international workshop on Middleware for pervasive and ad-hoc computing, ACM Press, 1-8

[5] Taisch, M., 2007, The Socrades European project (Service-Orientated Cross-layer InfRAstructure for Distributed Smart Embedded Devices), Presentation at the Second World Congress on Engineering Asset Management and The Fourth International Conference on Condition Monitoring

[6] Jennings, N. R., Wooldridge, M., 1998, Applications of intelligent agents, Springer-Verlag New York, Inc., 3-28

[7] Schoop, R., Neubert, R., Colombo, A., 2001, A multiagent-based distributed control platform for industrial flexible production systems, 27th Annual Conference of the IEEE Industrial Electronics Society, 1: 279-284

[8] Colombo, A., Neubert, R., Schoop, R., 2001, A solution to holonic control systems, Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation, 2: 489-498

[9] Deen, S, 2003, Agent-based manufacturing: advances in the holonic approach, Springer Verlag Berlin Heidelberg.

[10] Leitao, P., Colombo, A., Restivo, F., 2005, ADACOR: a collaborative production automation and control architecture, IEEE Intelligent Systems, 20/1: 58-66

[11] Colombo, A., Jammes, F., Smit, H., Harrison, R., Lastra, J., Delamer, I., 2005, Service-oriented architectures for collaborative automation, 32nd Annual Conference of IEEE Industrial Electronics Society, 6

[12] Bonfatti, F., Gadda, G., Monari, P. D., 1995, Re-usable software design for programmable logic controllers, SIGPLAN Not., ACM, 30/11: 31-40

[13] Erickson, K., 1996, Programmable logic controllers, IEEE Potentials, 15/1: 14-17

[14] Aramaki, N., Shimokawa, Y., Kuno, S., Saitoh, T., Hashimoto, H., 1997, A new architecture for high-performance programmable logic controller, 23rd International Conference on Industrial Electronics, Control and Instrumentation, 1: 187-190

[15] Huang, J., Li, Y., Luo, Z., Liu, X., Nan, K., 2003, The design of a new-type PLC based on IEC61131-3, International Conference on Machine Learning and Cybernetics, 2: 809-813

[16] Thramboulidis, K. C., 2003, Towards an engineering tool for implementing reusable distributed control systems, ESEC/FSE-11: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering, ACM, 351-354

[17] Hall, K., Staron, R., Zoitl, A., 2007, Challenges to Industry Adoption of the IEC 61499 Standard on Event-based Function Blocks, 5th IEEE International Conference on Industrial Informatics, 2: 823-828

[18] Hirsch, M., Gerber, C., Hanisch, H., Vyatkin, V., 2007, Design and Implementation of Heterogeneous Distributed Controllers According to the IEC 61499 Standard - A Case Study, 5th IEEE International Conference on Industrial Informatics, 2: 829-834

[19] Murata, T., Komoda, N., Matsumoto, K., Haruna, K., 1986, A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and its Applications in Factory Automation, IEEE Transactions on Industrial Electronics, 33/1: 1-8

[20] Nascimento, P. S. B., Maciel, P. R. M., Lima, M. E., Santana, R. E., Filho, A. G. S., 2004, A partial reconfigurable architecture for controllers based on Petri nets, SBCCI '04: Proceedings of the 17th symposium on Integrated circuits and system design, ACM, 16-21

[21] Hayashi, H., 1993, The IMS International Collaborative Program, Proceedings of the 24th ISIR, Japan Industrial Robot Association

[22] Moran, N. A., 2006, Symbiosis, Current Biology, Cell Press, Elsevier Inc., 16/20: 866-871

[23] Wooldridge, M., 2002, Introduction to MultiAgent Systems, Wiley

[24] Huhns, M., 2002, Agents as Web services, IEEE Internet Computing, 6/4: 93-95

[25] Ardissono, L., Goy, A., Petrone, G., 2003, Enabling conversations with web services, AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, ACM Press, pp. 819-826

[26] Shen, W., Li, Y., Hao, Q., Wang, S., Ghenniwa, H., 2005, Implementing collaborative manufacturing with intelligent Web services, The Fifth International Conference on Computer and Information Technology, CIT 2005, 1063-1069

[27] Blanchet, W., Stroulia, E., Elio, R., 2005, Supporting adaptive Web-service orchestration with an agent conversation framework, IEEE International Conference on Web Services.

[28] Mendes, J. M., Leitão, P., Colombo, A. W., Restivo, F., 2008, Service-oriented Control Architecture for Reconfigurable Production Systems, to appear in the Proceedings of the 6th IEEE International Conference on Industrial Informatics

[29] Gray, H., 2000, Gray's Anatomy of the Human Body, 20th Edition (Original by Philadelphia: Lea and Febiger, 1918), New York: Bartleby.com

[30] Mendes, J. M, Rodrigues, A., Leitão, P., Colombo, A. W., Restivo, F., 2008, Distributed Control Patterns using Device Profile for Web Services, Submitted to the 13th IEEE International Conference on Emerging Technologies and Factory Automation