Customizable Service-oriented Petri Net Controllers

J. Marco Mendes[1], Francisco Restivo[1], Paulo Leitão[2], Armando W. Colombo[3]
[1]Faculty of Engineering - University of Porto, Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal
[2]Polytechnic Institute of Bragança, Quinta S[ta] Apolónia, Apartado 134, 5301-857 Bragança, Portugal
[3]Schneider Electric Automation GmbH, Steinheimer Str. 117, D-63500 Seligenstadt, Germany
E-mails: {marco.mendes,fjr}@fe.up.pt, pleitao@ipb.pt, armando.colombo@de.schneider-electric.com

***Abstract* – In industrial automation, service-orientation is a relatively new and ascending concept and thus, concrete integrated methodologies are missing to accomplish the required development tasks. A suitable approach is to use the powerful set of features that Petri nets formalism provides for such dynamic systems. This paper presents a token game template that is part of the open methodology for the development of customized Petri nets controllers, targeting the engineering of service-oriented industrial automation. This template is based on a state machine specification for the life-cycle of transitions that leaves several options open for extending it with features depending on the application. The practical use and implementation should bring, among others, featured-full and integrated modeling, analysis and control capabilities, which is required by service-oriented ecosystems. This core structure was used and validated in the development of control applications for an industrial automation system.**

I. INTRODUCTION

Today, there are several solutions which take advantage of the newest mechatronics, information and communication technologies, to increase the modularity, flexibility and re-configurability of distributed automation systems. One of the most recently adopted and with promising applicability are the *Service-oriented Architectures* (SoA). The root of many production and automation companies is made of electronic devices and this represents a major task for SoA. Nevertheless, several efforts are being done in integrating and managing *Service-Oriented Industrial Automation* systems, such as the as the SOCRADES project (see http://www.socrades.eu). The other question is related to the industrial adoption of SoA principles, since most of the factories are heavily based on the centralized IEC 61131 standard for Programmable Logic Controllers (PLC). Even the relatively recent standard of IEC 61499 (to modernizing some aspects in the sense of distributed control and automation) has seen a slow to nonexistent application by the major control system equipment vendors [1].

In service-oriented industrial automation, the control method is partially open. Instead of incorporating all the flexibility once at the beginning, it should incorporate basic process models – both hardware and software – that can be rearranged or replaced quickly and reliably [2]. For this reason, the missing key aspects are related to more complex engineering, coordination and aggregation methods, especially tailored for the system's requirements. One option is to use the already applied standards in the business and e-commerce fields; other one being the adaptation of the IEC 61131 and IEC 61499 to the emergent requirements. The authors suggest the use of *Petri nets* to provide a balance between the typical SoA methodologies and the programming languages of the IEC 61131 standard. However, practical usage of Petri nets is limited by the lack of computer tools which would allow handling large and complex nets in a comfortable way [3]. From the other hand and considering also the use of Petri nets in the runtime of SoA, methods are missing for the development of customized Petri nets libraries for software applications and devices of multi-tasked usage. Based on this motivation, efforts were done in terms of creating a basis for the missing aspects.

The main contribution of this work is the specification of a token game template for the development of customized Petri nets controllers, suitable for diverse tasks of service-oriented automation. Due to the complex nature, these systems require several engineering aspects, and the choice of Petri nets covers a wide set of the requirements based on their well known foundation. The outline of the paper is the following: after the introduction, the domain of application of Petri nets is discussed. The open methodology for applied Petri nets is presented in section 3 and the template solution for Petri nets is given in section 4. A concrete application of Petri nets based on the previous methodology is shown in section 5. Finally, the paper resumes the conclusions and future work.

II. PETRI NETS IN SERVICE-ORIENTED INDUSTRIAL AUTOMATION: BACKGROUND AND DISCUSSION

Petri nets have a wide applicability associated to their modeling and analysis capabilities, but also the ability to represent system dynamics, especially concerning distributed, parallel and shared resources. Automation systems are a common target for the application of Petri nets and their higher-level deviated structures (e.g. colored Petri nets). In service-oriented systems, the use of visual modeling techniques such as Petri nets in the design of complex (Web) services is justified by many reasons. For example, visual representations provide a high-level yet precise language which allows expressing and reasoning about concepts at their natural level of abstraction. From the application point of view, a (Web) service behavior is basically a partially ordered set of operations. Therefore, it is straight-forward to map it into a Petri net and vice-versa. Operations are modeled by transitions and the state of the service is modeled by places. The arrows between places and transitions are used to specify causal relations [4].

978-1-4244-4649-0/09/$25.00 ©2009 IEEE

The traditional major application of Petri nets in SoA environments seems to be *orchestration* and *choreography*. For short, an orchestration defines the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function, and choreography is a model of the sequence of operations, states, and conditions that control the interactions involved in the participating services [5]. There are several protocols dealing with both concepts, and the most prominent is the *Web Services Business Process Execution Language* (WS-BPEL) [6], providing a powerful technology to aggregate encapsulated functionality and to define high-value Web services – backed by various development and runtime environments of major software companies [7]. Petri nets and high-level Petri nets are used in orchestration and choreography for modeling processes/composition [4], analysis purposes [7] and negotiation [8].

The application of BPEL directly in automation environment can be discussed. From one side, it has already a well defined syntax in eXtensible Markup Language (XML), development tools and can be used directly with Web services, providing a way of orchestration. From the other side, BPEL is a specification mainly targeting business requirements for both intra-corporate and business-to-business spaces. Therefore, it is unknown to automation system engineers that are used to the 61131 languages. Petri nets are here much more known and studied, as well as comparable to the sequential function charts of the IEC 61131. Other aspects are that the application of BPEL is probably too complex and descriptive to be interpreted by resource constrained devices (typically used in automation) and that it is not suitable for internal service process description based on device/software capabilities. BPEL depends on Web services and therefore it is technology dependent that is affordable to adapt to non-Web services based SoA. Last but not least, BPEL is missing analysis and validation support that is in fact an active research topic in the SoA business community. However, some efforts have been done in terms of the application of BPEL and orchestration for industrial automation with results (see [9] and [10]).

The kind of automation and production systems addressed in this work is characterized by having a flexible material flow with possibly many different flow specifications that can be offered by a defined layout. Therefore, high-degree of concurrency, competency relationships among components and non-deterministic sequences are present. In addition, the introduction of service-orientation makes possible to represent needs and conditions of the system's components in form of services that can be accessed by others. The formal specification and modeling of physical systems that have the characteristics addressed above can only be performed by using a mathematical tool able to represent all the characteristics without exceptions. The Petri nets theory offers all the required characteristics, being a bi-partite graph that can represent states and changes of states and representing a graphical and mathematical tool with a well-founded and proved theory. Besides this, it also has the necessary flexibility to develop higher-level structures based on the foundation, such as colored Petri nets. Since services, modeling, analysis, synthesis, integration and flexibility are used as the synonym of SoA, Petri nets based structures are strong candidates to fulfill the requirements.

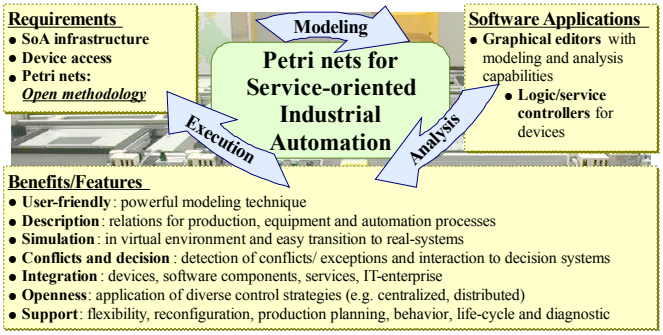

Fig. 1. Modeling, Analysis and Execution of Petri nets in Service-oriented Industrial Automation.

The use of Petri nets can be presented in the life-cycle of processes as modeling, analysis (simulation) and execution (control). Fig. 1 refers to this life-cycle approach that is also compliant to the development of traditional automation and production systems. For the development of the required software applications, there are some basic needs, namely a SoA infrastructure, device access (for software that runs on devices) and particularly for this work an open Petri net framework (see next section).

III. Open Methodology for Applied Petri Nets

Numerous extensions of this basic model of Petri nets have been introduced over the years [11]. The definition of high-level Petri nets is here used to describe any extension or addition that can be used over the formal definition of Petri nets. Several kinds of high-level Petri nets exist, however colored Petri nets are the widest used [12], and commonly associated as an analogy for high-level Petri nets. Colored Petri nets were considered, but the basis for the work was to permit user customization over the core Petri net formalism, so that requirements and objectives could be considered. An approach was specified for a concrete Petri nets methodology that is feasible and customized for the studied needs. Fig. 2 represents a requirement graph with the topics (or packages) that were found to be necessary for a full featured basis. The dashed arrows indicate that the input is optional and depends on the application.

For the ongoing description in this section and the following ones, the reader should be familiar with the basic concepts of Petri nets. The core for the Petri nets used in this work is based on the *formal definition of Petri nets*, extracted from the work of T. Murata (1989) [13]. The packages of Petri nets analysis and conflicts are mainly used for validation purposes and detection and resolution of conflicts that may be introduced in the Petri nets models. The Property system permits the enrichment of the Petri net and its elements (e.g.

transitions, places and arcs) with user-defined information that is not present in the formal definition of Petri nets, such as labels, priorities, actions, etc. Timed Petri nets provide the rules for delaying the step-wise behavior of Petri nets and thus permits relating them with real-time systems (see [14] as an example).

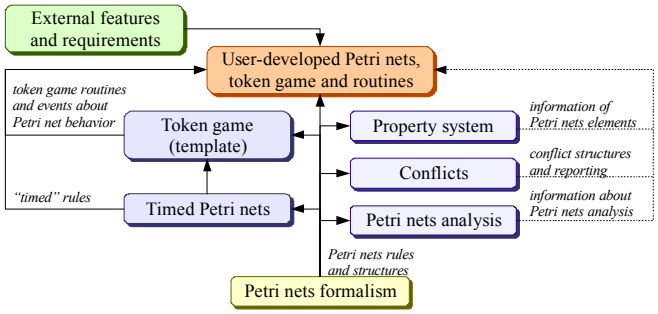

Fig. 2. Approach for the open methodology for applied Petri nets.

As said before, the main kernel of this work is the token game template, which is explained later on the next section. Including external features and requirements, and using specially the token game template, the final application can be tailored using the rules introduced in the methodology. An example is given in section 5 that defines a Petri nets controller for service applications.

IV. A Template for Token Games

The goal of this template is to provide a basis for Petri nets *token games* (an "engine" that runs a Petri net), based on the standard Petri nets formalism and extensible to permit the inclusion customized features. In its core and to maintain an asynchronous (independent) operation of the transitions, the template is a state machine specification for each transition and consequently responsible for managing the transition's state and evolution. The template itself is not a fully functional token game because it only defines a set of basic operations for analyzing and evolving transitions of the Petri net. Concrete token games can then be customized from this template.

The state machine is formally given in the following definition and Table I presents its state table: The *Petri nets transition state machine* for a transition $t \in T$ (where T represents a finite set of transitions of a Petri net) is defined by a 5-tuple $TSM_t = (\Sigma, \Phi, \Omega, \Gamma, \sigma_0)$, where $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_{16}\}$ is a set of 16 states, $\Phi : \Sigma \to \{\phi_1, \phi_2, ..., \phi_{16}\}$ is a set of flags for each state, $\Omega = \{\omega_1, \omega_2, ..., \omega_n\}$ defines a finite set of input/output symbols, $\Gamma : \Sigma \times \Omega \to \Sigma$ defines edges between two states as caused by the input/output and $\sigma_0 \in \Sigma$ defines the initial state ($\sigma_0 = \sigma_1 = 1$). *Note*: transitions in the state machine are referred as *edges*, not to be confused with the transitions of the Petri net.

The state machine related to each transition is made of several states representing the different combination of flags and the change between states is made by occurrence of events and execution of operations. Besides the *enabled* and *firing* flags that indicate if a transition is enabled or firing, *sleeping* and *jump* parameters were added to permit a more flexible management for the external time consuming functions (see definition below). In *summa*, there are four flags, and their combination result in 16 different states for the transition. The state of a transition is evolved by two main situations: Implicitly, by calling the *evolve function* over the transition. This function analyzes the actual state of the transition and proceeds according to the state machine in Table I (it corresponds to the white background rows); Explicitly, by 1) the enabling/disabling of the transition during previous processing of other transitions and their evolution and also 2) by calling the *wake-up function* over a transition due to occurrence of events (it disables then the *sleeping* flag). These situations are marked in darker background rows in Table I.

TABLE I
STATE TABLE OF A PETRI NETS TRANSITION

Note that the flags (ϕ) and the results of the flag testing function (ω_f) marked with underscored/bold are *true*, else *false*.

Actual state		Edge γ and $\omega(\gamma)$			Next state	Actual state		Edge γ and $\omega(\gamma)$			Next state
σ	Flags φ	ω_h	ω_u	ω_f	σ	σ	Flags φ	ω_h	ω_e	ω_x	σ
(1)	E F J S	–	–	–	(1)	(9)	E F J **S**	–	–	**E**	(10)
		–	–	**E**	(2)			–	–	S	(1)
(2)	**E** F J S	–	–	E	(1)			–	–	**E** S	(2)
		ω_{he} = U	ω_{ui}	**E**	(4)			ω_{hx} = W	–	–	(1)
		ω_{he} = U	ω_{ui}	E	(3)			ω_{hx} = I	–	–	(9)
		ω_{he} = I	–	–	(2)	(10)	**E** F J **S**	–	–	–	(10)
		ω_{he} = J	–	–	(14)			–	–	E	(9)
		ω_{he} = S	–	–	(10)			–	–	S	(2)
(3)	E **F** J S	–	–	**E**	(4)			–	–	E S	(1)
		ω_{hf} = U	ω_{uo}	**E**	(2)	(11)	E **F** J **S**	–	–	–	(11)
		ω_{hf} = U	ω_{uo}	E	(1)			–	–	**E**	(12)
		ω_{hf} = I	–	–	(3)			–	–	S	(3)
		ω_{hf} = J	–	–	(15)			–	–	**E** S	(4)
		ω_{hf} = S	–	–	(11)	(12)	**E F** J **S**	–	–	–	(12)
(4)	**E F** J S	–	–	E	(3)			–	–	E	(11)
		ω_{hf} = U	ω_{uo}	**E**	(2)			–	–	S	(4)
		ω_{hf} = U	ω_{uo}	E	(1)			–	–	E S	(3)
		ω_{hf} = I	–	–	(4)	(13)	E F **J S**	–	–	**E**	(14)
		ω_{hf} = J	–	–	(16)			–	–	S	(5)
		ω_{hf} = S	–	–	(12)			–	–	**E** S	(6)
(5)	E F **J** S	–	–	–	(5)			ω_{hx} = W	–	–	(1)
		–	–	**E**	(6)			ω_{hx} = I	–	–	(13)
(6)	**E** F **J** S	–	–	E	(5)	(14)	**E** F **J S**	–	–	–	(14)
		–	ω_{ui}	**E**	(4)			–	–	E	(13)
		–	ω_{ui}	E	(3)			–	–	S	(6)
(7)	E **F J** S	–	–	**E**	(8)			–	–	E S	(5)
		–	ω_{uo}	**E**	(2)	(15)	E **F J S**	–	–	–	(15)
		–	ω_{uo}	E	(1)			–	–	**E**	(16)
(8)	**E F J** S	–	–	E	(7)			–	–	S	(7)
		–	ω_{uo}	**E**	(2)			–	–	**E** S	(8)
		–	ω_{uo}	E	(1)	(16)	**E F J S**	–	–	–	(16)
								–	–	E	(15)
								–	–	S	(8)
								–	–	S E	(7)

The previous definition of TSM_t has several remarks that are discussed. The flags for a state $\sigma \in \Sigma$ are defined by $\phi_\sigma = (\phi_E, \phi_F, \phi_J, \phi_S)$. The meaning of a flag when it is true is:

- *Enabled* (ϕ_E): the transition is automatically enabled depending on the actual marking of the Petri net;
- *Firing* (ϕ_F): the transition is on firing process. A particular note considered here is that if a transition is during firing process and enabled again, the enabling is ignored until the firing process is concluded;
- *Jump* (ϕ_J): the handler functions (see below) are not called in the next analyzing iteration of the transition;

– *Sleeping* (ϕ_S): the transition is waiting for external signal and consequently is blocked.

An input/output symbol $\omega \in \Omega$ has three types of functions that are analyzed/executed in a sequential way, $\omega = \{\omega_h, \omega_u, \omega_f\}$. The functions have mixed input/output.

Handler functions $\omega_h \in \{\omega_{he}, \omega_{hf}, \omega_{hx}\}$ (generate both input and output) are used for the external communication (and thus must be defined). Their return values contribute to the definition of the next steps in the state machine. In practice, the main distinction between enabled and firing functions is that the first one does not guarantee the number of tokens, not even if the transition will effectively enter the firing process (e.g. it may disable again):

– *Enabled function* (ω_{he}): called when a transition is enabled and thus permitting it to enter the firing process (and consuming tokens). It may return {*U*, *S*, *I*, *J*} (for testing as input), meaning *update* (if still enabled, the transition then enters effectively the firing process by calling consequently one of the update functions), *sleep* (puts the transition in sleeping, awaiting an call of the *wake-up function*), *ignore* (does nothing and does not change the status of the transition) and *jump* (does the same as returning *S*, but next time, the *enabled function* is not called);
– *Firing function* (ω_{hf}): called when a transition is on firing and thus permitting it to leave this process (and expel tokens). The return status is similar to the *enabled function*, but for leaving the firing process;
– *Exception function* (ω_{hx}): called on exceptions when a transition is disabled during sleeping (see state 9 and 13 of Table I). Returning *W* (*wake-up*), the sleeping mode (and jump mode if active) is disabled. Returning *I* (*ignore*) the actual state is not changed.

Update functions $\omega_u \in \{\omega_{ui}, \omega_{uo}\}$, (only generate output) are responsible for evolving the transition according to the actual state of flags, i.e. they are used to effectively consume and expel tokens. The *update input* ω_{ui} updates the input of a transition, i.e. consuming tokens and the *update output* ω_{uo} is responsible for updating the output of a transition (expelling tokens). Last, the *flag testing function* ω_f (only input) tests the status of a state, namely its flags (ϕ_E, ϕ_F, ϕ_J, ϕ_S) if they are true or false.

The previous defined template provides the basis for Petri net token games, in sense of affecting the life-cycle of transitions. For the whole Petri net, it is left open several aspects that permit its customization (e.g. in which order to analyze the transitions, conflict management, external functions call, etc.). More details are given in the next section.

V. Using the Token Game Template in Industrial Service Environments: Hands-on Experiences

The open methodology and the token game template do not define a concrete Petri nets application and leaves to the engineer and/or developers the possibility to customize their Petri nets application. Therefore, when defining a concrete application based on the previous methodology (especially a token game), several aspects must be considered:

– Create the necessary Petri nets structures based on the methodology;
– Define the *handler functions* of the token game and the call of the *wake-up function*;
– Consider a specification of properties and their association to the elements of the Petri net and the evolution of the net;
– Decide about how conflicts are managed and reported;
– Define the life-cycle of the whole Petri net based on the ones from the transitions. A special attention must be taken in the order of analyzing transitions, when to begin/finish the token game and also deadlock detection.

In service-oriented automation, the resulting Petri nets applications must also consider their environment and what features should be handled. Additionally the necessary service infra-structure must be available and also, when targeting devices, hardware access must be considered. From the Petri net side, one obvious conclusion is that services should somehow be related to Petri nets, whatever it should represent the logic of a service, work-flow between services or other situations. Other thoughts have to be done if the final structure should help in the modeling, analysis and/or be used in the control of a service environment. The next subsection exemplifies a concrete Petri nets structure and token game that can be used in service applications.

A. A Petri Nets Controller for Service Applications

The following application library defines a Petri nets controller that can be used for the modeling, analysis and execution, embedded into software applications and/or devices. In terms of services, needs and requests are used in the description of predicted device behavior and also in the definition of service-workflows (as in traditional orchestration). The purpose of the library is to provide Petri nets definition capabilities, properties extraction from analysis and also to be used in simulation and real time service-based systems.

First, some background information for the used service approach. For this work, a service is not considered as an implementation or as a specification by a Petri net model (such as in most Petri net based orchestration publications), instead service elements (e.g. ports and operations) are viewed as associations to Petri nets elements. Moreover, the concept of *service interface* is referenced instead of the service itself, because the interface provides the description of services to be used by providers (for its realization) and requesters (for its use). For sake of simplification, a service interface can be defined by its name, description (requirements to use the service), location, provider and set of ports. Each port is a realization of a port type. A port type is an abstract collection of operations that describe actions and related message exchange patterns.

Fig. 3 represents the specified *Petri nets controller* (library) and additional modules. The modules have dedicated tasks that complement the work of the Petri nets engine. For short,

Service Infrastructure has the necessary functions to provide service capabilities; *Device Interface* permits to access the hosting device (if the application runs on one), mainly for reading and setting I/O signals; *Graphical User Interface* (GUI) can be used for visual representations and also communication to the user; *Decision Support System* (DSS) is used for conflict resolution and exception handling. For more info about this modular architecture see [15].

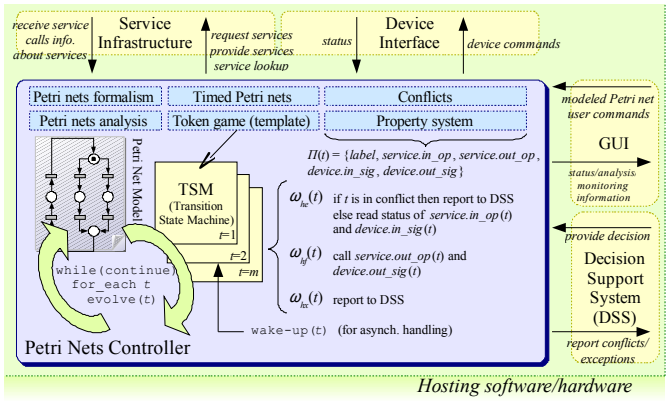

Fig. 3. Developed Petri nets controller/library using the open methodology.

Besides the formal structures/rules and analysis capabilities provided by Petri nets, the real "juice" of the controller is the ability to interpret Petri nets models and their association to services and device signals. First of all, properties must be defined and associated to Petri nets elements. For the sake of simplicity, only transitions were characterized, because they represent the interaction to the "outside world". Transitions have a label and also several action properties, e.g. label, *service.in_op*, *service.out_op*, *device.in_sig*, *device.out_sig*. The service action properties are used to describe input and output operations of a service port, i.e. messages to be waiting for (in) and messages to send (out). For the device, the action properties define signals to be tested (in) and signals to be written (out). The action properties can be used by the Petri nets engine by accessing the corresponding module, in this case the Service Infrastructure and the Device Interface. The *handler functions* of the token game template for each transition operate over these properties. The *enabled function* ω_{he} tests first if the transition is in conflict and if this is true, it reports the information to the DSS module and awaits its instruction. If no conflict is present, ω_{he} considers the *service.in_op* and *device.in_sig*. Only if both are true (i.e. service message is available and valid signal from the device), the transition enters the firing mode (corresponding to returning (*U*)pdate by the enabled function or (*S*)leep/(*J*)ump for asynchronous handling). If some of the action properties are not defined, they are not considered (in this case meaning true). Similar, the *firing function* ω_{hf} considers the *service.out_op* and *device.out_sig*, sending a message and writing a signal (if defined). The *exception function* ω_{hx} is only used on transitions that were disabled (e.g. in case of conflict resolution) and this event is transmitted to the DSS. The *wake-up function* is called in asynchronous handling from the modules to signalize that events are finished and that the token game can consequently enter or leave the firing process of a transition. Each transition has its own state machine defined in the token game template *TSM*(*t*). In a whole, the evolution of the transitions is done in a sequential loop (as seen in Fig. 3) until a stop command is received or a dead-end was detected. Commands can be received for example via the GUI and from the other hand, it may send monitoring/status information back to the GUI (for visualization purposes).

B. Application in Industrial Scenarios and Software Implementations

The experimental case-studies that were realized used a production cell as basis, which is built by autonomous smart devices, representing conveyors, cross-tables and work stations. There are several decision points for work-pieces (pallets that transport products along the conveyor system), which indicate alternative paths and stops at the work stations. Petri nets were used to model, analyze and execute the whole scenario based on several sub-models representing the predicted behavior of the system's components.

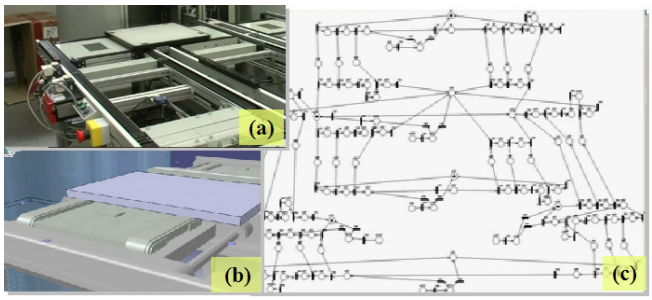

Fig. 4. Demonstration of the service-enabled scenario. (a) real equipment, (b) virtual representation and (c) Petri nets model used for the control.

A software project is being developed that basically integrates the several software pieces build before under the same umbrella. The *Continuum Project* is constituted by several packages, including *Continuum Petri Nets Kernel* – implementation of the methodology explained in this paper, *Continuum Petri Nets Module* – the library engine conceptually described in the previous subsection and that is used by several applications inside this project, *Continuum Development Studio* – a visual tool for several tasks in automation, including the development of Petri nets models, and *Continuum Bot Framework* – basic shell for developing automation software components, besides others.

First successful experimentations were done with "virtual" services that were hosted by DELMIA Automation engineering tool, having a full representation of the case-study scenario (see Fig. 4). A demonstration video of this experimentation ("*IP SOCRADES Demonstration of Service-Oriented Architecture integrating Real and Virtual Devices in the Electronic Assembly Scenario*") can be seen at http://www.youtube.com/watch?v=0aRRvqEln2I. Petri nets models were developed, analyzed and executed from the editor to access the services of the virtual cell and controlling

its behavior. The connection of the models is done through specific interfaces (ports), which, when connected, drive the correct collaboration between the two connected components. When executing a Petri net inside the editor, its status is made visible, giving information about the actual marking, the enabling and the firing of transitions of the executed Petri net.

Some aspects in flexibility where demonstrated. Mechatronic components expose its functionality as services and have the plug-in/-out capability which allows performing dynamic reconfiguration of the layout. Since the pallet modeled by tokens of the Petri net are able to carry any part and the flow is not pre-fixed by a scheduler system, the system presents a high flexibility in the quantity of different products to be processed at the same time. The token-game manages the dynamic execution and the Petri net is inherently exposing conflicts situation that appears e.g. in the material flow control. These open points (the conflicts) can be used for calling decision making systems via the invocation of specific services that will give an answer to decide over the conflict. One example is the material flow that can be represented in the Petri nets, but requires decision when there are several possibilities represented.

VI. Conclusions and Future Work

This research and development work was started from the missing aspects of Petri nets applications in service-oriented industrial automation environment, especially a methodology for the development of custom based Petri nets software. The main contribution can be highlighted, representing a novel approach for defining customizable Petri nets controllers based on the defined template for token games which basis is a formal defined state machine for transitions. Other contributions are included in the open methodology but are not subject of the work described in this paper; they comprise a property system for Petri nets, active conflict management, and the effected studies and software developments, targeting the flexible engineering of service-oriented ecosystems. The methodology and the token game template permit the specification of custom Petri nets applications, but also maintaining the formal Petri nets foundation. These aspects were demonstrated by the specified and developed Petri nets controller for service environment and supported by the creation of a development project for the engineering of these systems. The result was used in the modeling, analysis and execution of a virtual representation of the service-enabled factory cell, specified and controlled by Petri nets models.

The experimentation with the real devices of the scenario is on early stage and not presented as a result, representing the major input for the future work to be done in this field. Efforts are being done in integrating the Petri nets controller (the same as used in the Continuum Development Studio) into service-enabled automation devices that are known for being resource constrained. This makes possible in the near future having both service-oriented virtual representation and real cell developed with all the features that Petri nets provide. One challenge is also the maintenance of the same Petri nets models for both virtual and real scenario, decreasing transitional efforts. Last, but not least, the Continuum project is in continuation to be enhanced and extended, particularly the framework for automation components, the visual editor and also some aspects in the Petri nets controller.

Acknowledgment

The authors would like to thank the European Commission and the partners of the EU IST FP6 project “Service-Oriented Cross-layer infrastructure for Distributed smart Embedded devices” (SOCRADES), the EU FP6 “Network of Excellence for Innovative Production Machines and Systems” (I*PROMS), and the European ICT FP7 project “Cooperating Objects Network of Excellence” (CONET) for their support.

References

[1] K.H. Hall, R.J. Staron and A. Zoitl, “Challenges to Industry Adoption of the IEC 61499 Standard on Event-based Function Blocks”, Proceedings of the 5th IEEE International Conference on Industrial Informatics, vol. 2, pp. 823-828, 2007.

[2] M.G. Mehrabi, A.G. Ulsoy and Y. Koren, “Reconfigurable Manufacturing Systems and their Enabling Technologies”, *International Jornal of Manufacturing Technology and Management*, vol. 1, n. 1, pp. 113-130, 2000.

[3] Z. Suraj, B. Fryc, Z. Matusiewicz and K. Pancerz, “A Petri Net System - an Overview”, In *Fundam. Inf.*, vol 71, n. 1, pp. 101-120, 2006.

[4] R. Hamadi and B. Benatallah, “A Petri net-based model for web service composition”, Proceedings of the 14th Australasian Database Conference, pp. 191-200, 2003.

[5] Web Services Architecture. W3C Working Group (available at http://www.w3.org/TR/ws-arch/), 2004.

[6] Web Services Business Process Execution Language. OASIS Standard. (available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html), 2007.

[7] A, Martens, S. Moser, A. Gerhardt and K. Funk, “Analyzing Compatibility of BPEL Processes.”, Proceedings of the International Conference on Internet and Web Applications and Services/Advanced International Conference on Telecommunications, pp. 147, 2006.

[8] H. Jiang, J. Gu and Q. Yu, “Modeling of Web-based collaborative negotiation systems using colored Petri net.”, Proceedings of the 12th International Multi-Media Modelling Conference, pp. 8, 2006.

[9] F. Jammes, H. Smit, J.L.M. Lastra and I.M. Delamer, “Orchestration of service-oriented manufacturing processes”, Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation, vol. 1, pp. 8, 2005.

[10] J. Puttonen, A. Lobov and J. Lastra, “An application of BPEL for service orchestration in an industrial environment”, Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, pp. 530-537, 2008.

[11] E. Badouel, J. Chenou and G. Guillou, “Petri Algebras”. In: *Automata, Languages and Programming, Lecture Notes in Computer Science*, vol. 3580, pp. 742-754, 2005.

[12] K. Jensen, “Coloured Petri Nets”, Slides about Coloured Petri nets, Department of Computer Science University of Aarhus, Denmark (available at http://www.daimi.au.dk/~kjensen), 2000.

[13] T. Murata, “Petri nets: Properties, analysis and applications” In: *Proceedings of the IEEE*, vol. 77, pp. 541-580, 1989.

[14] C. Ghezzi, D. Mandrioli, S. Morasca and M. Pezze, “A general way to put time in Petri nets”, Proceedings of the 5th international workshop on Software specification and design, pp. 60-67, ACM Press, New York, NY, USA, 1989.

[15] J.M. Mendes, P. Leitão, A.W. Colombo and F. Restivo, “Service-oriented control architecture for reconfigurable production systems, Proceedings of the 6th IEEE International Conference on Industrial Informatics, pp. 744-749, 2008.