Composition of Petri Nets Models in Service-oriented Industrial Automation

J. Marco Mendes[1], Paulo Leitão[2,4], Francisco Restivo[1,4], Armando W. Colombo[3]
[1]University of Porto - Faculty of Engineering, Rua Dr. Roberto Frias, 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
[4]LIACC – Artificial Intelligence and Computer Science Laboratory, University of Porto, Portugal
E-mails: {marco.mendes,fjr}@fe.up.pt, pleitao@ipb.pt, armando.colombo@de.schneider-electric.com

***Abstract* – In service-oriented systems, composition of services is required to build new, distributed and more complex services, based on the logic behavior of individual ones. This paper discusses the formal composition of Petri nets models used for the process description and control in service-oriented automation systems. The proposed approach considers two forms for the composition of services, notably the offline composition, applied during the design phase, and the online composition, related to the synchronization of Petri nets models on the fly. An experimental case study is used to illustrate the proposed composition approach.**

I. INTRODUCTION

Service-oriented architecture (SOA) is being seen as a new playground for experimentation in industrial automation since its relative success in electronic commerce and business domains in the beginning of the 21st century. The use of the SOA paradigm implemented through *web services* technologies enables the adoption of an unifying technology for all levels of the enterprise, from sensors and actuators to enterprise business processes [1]. This approach was handled by the EU IST FP6 SOCRADES (Service-Oriented Cross-layer Infrastructure for Distributed Smart Embedded Devices) project, which primary objective was to develop a design, execution and management platform for the next-generation of industrial automation systems, exploiting the SOA paradigm both at the device and application levels [2].

A main concern in service-based systems is how the services "play" together. Since services aren't isolated entities exposed by the intervenient software components, it is necessary to consider some kind of logic that will be responsible for their interaction patterns. *Orchestration* is the most well-known process used to define the invocation order of services. The achieved sequence can be used to form a composition, i.e. to create a new service (*composite service*). The model-based *orchestration engine* is able to interpret a given sequence made of services (an orchestration) and execute it. The work-plan associated to services can be defined using different methods [3], namely the Business Process Execution Language (BPEL) [4], the Petri nets formalism [5-7] or even the IEC 61131-3 languages [8].

In automation systems, it is not usual to deal in parallel with the specification of the automation system behavior and the configuration of the system equipments, which implies that re-adjustments in one side would require major efforts in the other one. One example, and also the main problem targeted in this work, is the development of orchestration models in Petri nets formalism without knowing the final control composition in terms of service-enabled and orchestration-capable devices. The number of embedded devices and also their distribution over transport and production equipments would affect the previously designed models. Moreover, the objective is the reduction of the configuration and design efforts (in this case Petri nets models with associated service representations), when the control layout has changed or is unknown.

This paper discusses the composition of Petri nets models in service-oriented automation systems, introducing an approach that comprises two types of composition, notably the offline composition (the composed model is a result of the combination of several smaller ones) and the online composition (where individual models are distributed and synchronized together via service logic). The proposed approach is powerful enough to create complex and flexible services from simpler ones, both at designing and operation phases. The Continuum Development Tools [9] was used to design, analyze and compose Petri nets and also to configure service-enabled orchestration-capable embedded devices. Experiments were done over an industrial transport system with modules represented by services accessible via the network. The offered features of composition, as well as the characteristics of these Petri net models, demonstrate that the design of the system's behavior can be abstracted from the device distribution where these models should run afterwards.

The remainder of this paper is organized as follows: first, section 2 overviews the basic concepts of SOA in automation domain and the use of Petri nets formalism to represent the services' process behavior. Section 3 introduces the proposed approach for the composition of Petri nets models, and section 4 illustrates the application of the proposed concepts into an experimental case study. Finally, section 5 rounds up the paper with conclusions.

II. SOA IN AUTOMATION: A TECHNICAL OVERVIEW

In a SOA architecture, services are the primary organizing principle [10]. A service is a software module that encapsulates the business/control logic or resource

functionality of an unit that responds to specific requests and/or is a source for events. From the practical standpoint, web services offer a technology that is rich and flexible enough to make SOAs a reality [11]. One of the major protocols used in web services (and also important to this work) is WSDL (Web Services Description Language) [12], which is a W3C specification that provides an abstract, technology neutral language for the definition of published operations of a service [13]. In a simple manner, services are a set of operations, which in turn can be one of different message exchange patterns, for instance request/response or events. The exchange of information of the operations is via the network using SOAP (Simple Object Access Protocol) formatted messages (which is another web service standard).

Web services (and SOA) would be difficult to be practicable in industrial automation if there were not some trade-offs considering the domain and available resources. The Devices Profile for Web Services (DPWS) [14] was defined considering some specific web service protocols but also restricting the usage of web services to keep aspects of the limitations in embedded systems [15].

Aiming to achieve the collaborative global behavior, one or more services may interact. The composition of two or more services generates a new, more complex service, providing both the original individual behavioral logic and a new collaborative behavior for carrying out a new composite task [16]. In literature, several works are focused on composition and orchestration of web services, but they are mainly directed to e-business/e-commerce (see [17-19]). In automation and manufacturing, the service composition and collaboration have been studied with web semantics [20], service classification [21], service binding [22] and the collaboration/integration of multi-agent systems [23].

As previously referred, the process behavior of services can be described using different languages [3], one of them being the Petri nets formalism. Petri nets are a well-known process modeling technique with a strong mathematical foundation. A Petri net is a directed, bipartite graph in which each node is either a place or a transition. Tokens occupy places representing resources or states of the system. For a more elaborate introduction to Petri nets, the reader is referred to [5-7]. In this work, the orchestration of services is modeled using Petri nets, with service operations being mapped into transitions.

Since automation systems are a rich playground for diverse types of equipments and embedded computational systems, services and their orchestration has been applied to these specific systems. *Smart embedded devices* are the host for the most of the services exposed in the system and also responsible for the coordination and control activities (Fig. 1). These devices have two main interfaces: one to mediate the shop floor equipment via I/O (e.g. a lifter) and other one to manage the access to the service bus by exposing and requesting services. An automation entity is considered the software representation of the device and can also be hosted in different computational equipments. The internal *orchestration engine* is used to "link" services into higher ones and is the host for the service-enriched Petri nets model.

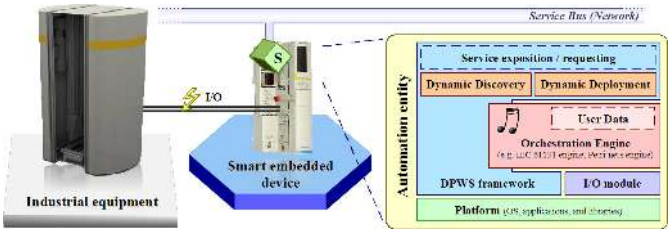

Fig. 1. Smart embedded device with orchestration engine.

These entities are configurable software components with the dynamic deployment feature of the used DPWS (SOA4D at http://forge.soa4d.org/). This does not only configure the automation entity and its services but also the orchestration engine. The setup of the embedded Petri net engine, for example, is started by receiving the XML representation of a Petri net included in the uploaded deployment file. Afterwards, the engine is able to interpret Petri net models and coordinate the available services on devices. Due to the service-based communication it is also possible the lateral collaboration with other entities.

The application of Petri nets can range from typical systems with defined behavior to more complex ones with distributed participants. In any case, system engineering and associated tools are required to facilitate the developer's intervention. The Continuum Development Tool (CDT) and the visual editor Continuum Development Studio (CDS) are being developed with the objective to facilitate the design, analysis and configuration of distributed service-oriented automation systems, by using until now, a special kind of Petri nets associated to service information. Other important feature of this tool is the capability of validating methodologies applied to the automation control with an integrated application (client) and distributed resources (servers). Petri nets based orchestration engines can be configured with this tool, which uses DPWS as the framework for the integration of services. For a more information, the reader is referred to [9].

III. Composition of Service-based Petri nets

The composition of Petri nets is viewed as additional logic to synchronize the process of two or more models. As illustrated in the left side of Fig. 2, three Petri net models are composed and their intersection is defined as *composition logic*. This can be done *offline* using a *composition tool* by generating a new Petri net model that is the composition of several individual ones and also *online*, where individual models are maintained in their distributional units and synchronized together on the fly via the network. The online composition can also be designated as *virtual composition*, because no new model is generated, but individual models have to be linked together as they were part of one. The online composition is done by using service-oriented architecture as means of information exchange and service

representation (i.e. to identify the synchronization points of the Petri nets model). In both cases, at runtime an orchestration engine will get the Petri nets model and run it.

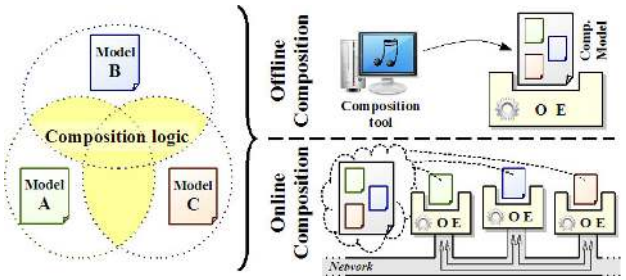

Fig. 2. Composition of Petri net models (offline and online) and their execution in orchestration engines (OE).

The two forms of composition will be explained in the following sub-sections.

A. Offline Composition: The Petri Net Composer Tool (PNC)

Offline composition comprises the generation of a new Petri net model based on the connection of two or more individual ones. If PN_1 and PN_2 are two Petri nets, $PN_{1\cup2} = PN_1 \cup PN_2$ represents its composition. This composition procedure can be applied to two or more Petri nets models. Fig. 3 represents an example where two Petri nets are composed via the addition of composition logic.

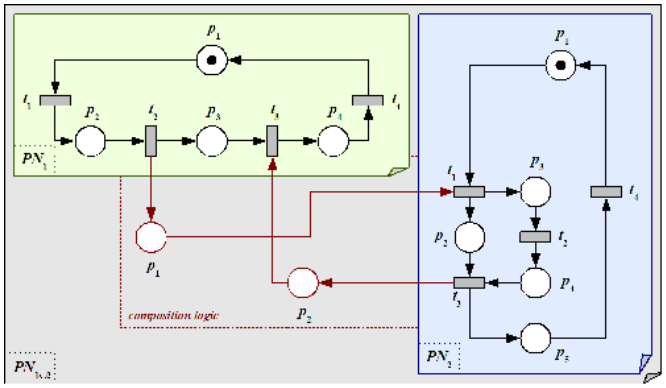

Fig. 3. Offline composition of Petri nets PN_1 and PN_2 resulting in $PN_{1\cup2}$.

When the composition is completed, new inter-logic is generated. The basic idea is to match two transitions from different models by connecting them via a place (and corresponding arcs). Additionally, the specification of the direction should be considered (e.g. the transition t_2 from PN_1 is the input of the transition t_1 from PN_2). This approach, besides to simplify the development of bigger and more complex models, also facilitates the synchronization of models viewed as individual entities.

Since composition is not limited to a set of transitions and thereof, transitions may be grouped into logical connection groups, the concept of ports was introduced into the Petri net models. An example is given in Fig. 4 where the conveyor A is the client and the conveyor B is the server. The transition t_1 represents a start order of the conveyor B, the transition t_2 represents that it has started and the transition t_3 notifies when it is completed. All these three transitions are part of the same port (*port=in* of the conveyor B) and are input or output transitions in contrast to their counterparts in the other model of the conveyor A. Additionally, they have a sequence reference to indicate the order of connection of the transitions (e.g. the output transition t_4 of *port=out* from the conveyor A will be connected to the input transition t_1 of *port=in* from the conveyor B, because they have the same sequence reference, *port_out_seq=port_in_seq*=1). In a few words, for two interconnected ports from different models, input transitions will be connected to output transitions with the same sequence reference. This can be seen in the connection table of Fig. 4, where transitions have two properties, {*port*, *port_in_seq*|*port_out_seq*}, which indicate, respectively, the port they belong and also the input/output sequence reference.

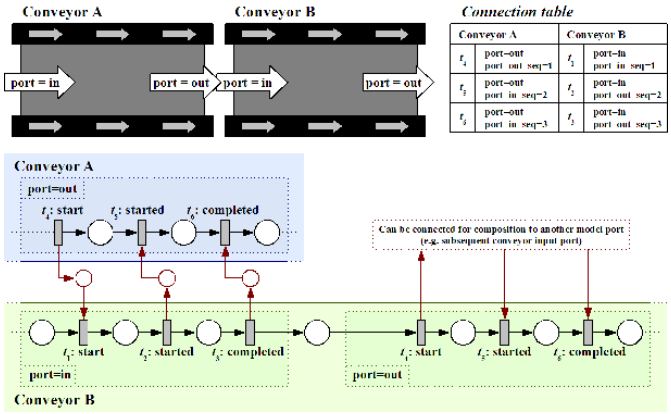

Fig. 4. Port-based composition of two Petri net models representing the behavior of two conveyors.

As seen in Fig. 4, ports are also useful when the models represent mechatronic devices. For example, a conveyor may have two ports (one for the input of pallets and another one for the output of pallets). This situation is also represented in its control model, in which it contains two ports that are used to connect to other models from other devices (e.g. adjacent conveyors). Moreover, to facilitate the composition, information about the layout and displacement of equipment can be used for (semi-) automatic composition. For this purpose, the Petri net model should include a label (in case of Fig. 4, Conveyor A and Conveyor B) and the layout information defining which resources/devices may be connected and through which port.

A XML file was adopted to represent the resource connections and to improve the automatic offline composition. The resource connections for the example of Fig. 4 looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
  <connections>
    <connection
      resource1="Conveyor A" port1="out"
      resource2="Conveyor B" port2="in"
    />
  </connections>
```

In the example, only one connection is included, but multiple connections can also be defined. For this purpose, new `<connection/>` tags must be defined inside the `<connections>` tag. If the conveyor B would be connected

to a conveyor C on the right, then this information is represented by one additional <connection/> tag.

For the offline composition, a special tool is needed and therefore was developed in the top of the CDS. The *Petri Net Composer Tool* (PNC) allows the user to create simplified models and then link them together into a global model.

The individual models can be stored into XML-formated files (Extensible Petri Net File Format, with the extension *.xpn). In order to be successful, the user must select the transitions that belong to a connection port and define the two properties as previously explained (*port* and *port_in_seq*|*port_out_seq*). After defining the models, a layout information file has to be created with the resource connections and saved with the extension ".xrc".

Once all referred files are available, the user may use the PNC menu entry to proceed with the composition, by selecting the resource connections file (*.xrc) and the several Petri net files (*.xpn) to be composed. The composition is automatically done by extracting the information from the *.xrc file, creating a new empty Petri net model, copying all the sub Petri nets models into the new model, generating the composition logic and saving the new model representing the composition. The new *.xpn file can then be opened and processed as a normal Petri net model.

The example of Fig. 5 shows the result of a composition of two individual models using the PNC tool. The window on the right side is the property editor that is used to define the properties of the transitions.

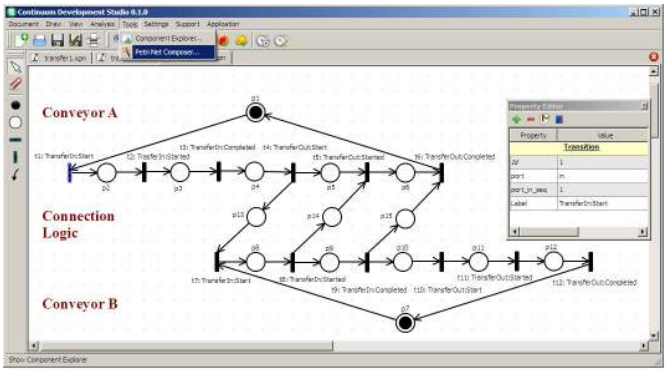

Fig. 5. Composition of Conveyors A and B using the PNC tool.

One characteristic not explained in this section is the representation of services, their operations and service calls in the model. This issue will be introduced in the next subsection, because the online composition is based on the online connection of models via service technology. Note that the services represented in the models are not only used for the online composition, but also for the access and exposition of other services in the system (e.g. production services, automation services and maintenance services).

B. Online Composition: Synchronization of Models

Online composition means that each model runs separately in its own orchestration engine and they are synchronized via a connection logic. In this case, the connection logic represents a service-based communication act, where services and their operations are described in Petri net models.

Online composition requires that Petri nets have information on how to invoke and represent services to synchronize with the other models. This is done by describing transitions in the Petri net model. A transition willing of sending a request/response or an event must be enabled, and the action is done when it fires. In the other hand, a transition receiving a message from a request, response or event, will only fire if it is enabled and the message is there. Fig. 6 represents these two types of associations.

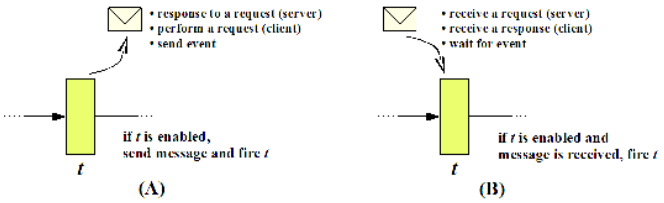

Fig. 6. Two types of service message association to transitions: (A) transition outputs a message, (B) transition waits for a message.

The information to be used by transitions is gathered by an imported WSDL file that contains the description of the service. Depending on the operation, transitions can be part of a client request/response, server request/response, client event and server event. The first two types require two transitions: one for initializing the request and one for the response. It is also possible to test responses by their return parameters, implying the use of one response transition for each test (resulting in a conflict in the Petri net model). The difference of an operation being a server or client is obvious: a server waits for the request and then gives a response, and a client makes a request and waits for a response. Events are possible as client and server, but only require one single direction (and consequently, one transition).

Fig. 7 represents the connection of the two conveyors illustrated in Fig. 4 with the connection logic based on the service infrastructure. In the example, the conveyor A is the client and the conveyor B is the server. Both use the transfer interface (transfer.wsdl) to express the service in the model. The sequence *start*, *started* and *completed* will be transformed into a *TransferIn(request)*, *TransferIn(response)* and *TransferStatus("completed")* sequence, to be compliant with the WSDL file.

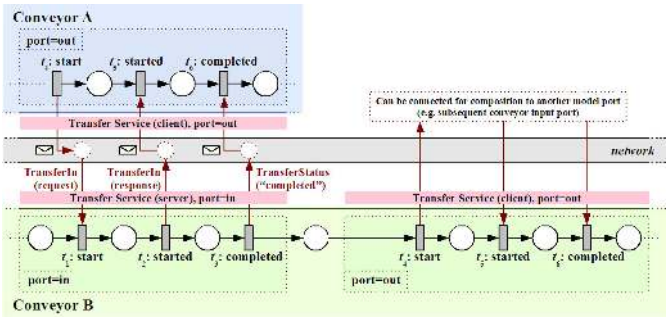

Fig. 7. Online composition of the Conveyors of Fig. 4 using service-orientation.

In the CDS tool, WSDL files can be imported (see the lower window of Fig. 8) and their service operations listed. Service operations can be applied over a selected transition. If the operation is a request/response, then the user has to select one of them (request or response) and if it is a client or a server operation (i.e. the direction of the message). A request must also be defined with the device class reference and transition(s) that will receive the response. In case of events, the server or client viewpoint is selected, as well as the device class reference. The properties are automatically applied to the transition (this can be seen in the Property editor, in the right side of Fig. 8), and can be edited, including the addition of parameters to the message fired by the transition (or to be tested by incoming message).

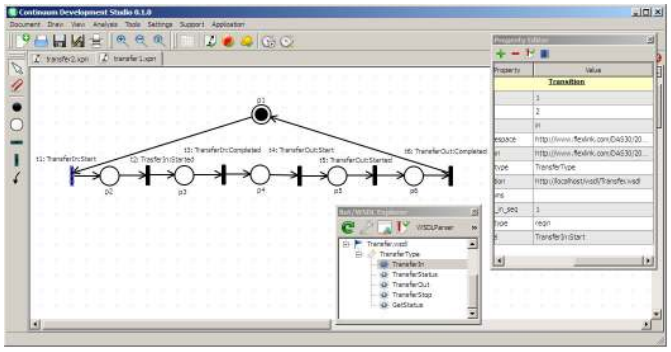

Fig. 8. Upload of the service operation information (TransferIn) from a WSDL file to a transition, using the CDS.

After the modeling phase it is possible to configure and upload the achieved composed model to a device embedding an orchestration engine, which will interpret and run the Petri nets model. This issue will be detailed in the next section.

IV. Configuration and Experiment

The industrial equipment used for the experimentation of the proposed concepts is a flexible transport system, used in the EU FP6 SOCRADES project and illustrated in Fig. 9, made of conveyors and lifters, supporting the routing of pallets to different workstations.

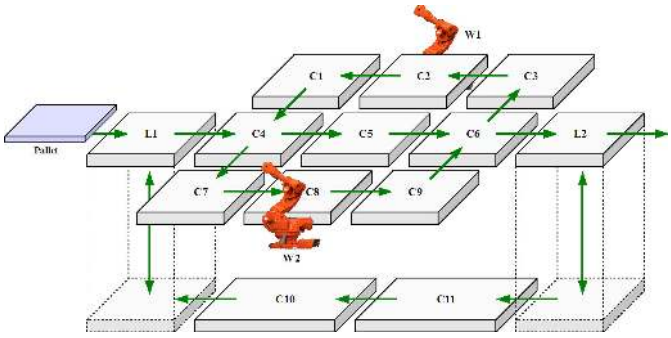

Fig. 9. Layout of the transport system (C1-C11: conveyor modules, L1/L2: lifters, W1/W2: workstations).

The several modules of the system are connected to industrial controllers that "transform" the I/O logic into atomic services. These atomic services can then be used by the smart embedded devices (i.e. the orchestration engines) to run the logic with the Petri net models and thus orchestrate the whole system.

The CDS provides a deployment tool that will generate deployment files and upload them into a selected device. The deployment files have XML-formated information to configure the device and associated resources (e.g characterisitics of the provided services, Petri net model for the orchestration engine, etc.). Afterwards, the device can auto configure itself with the deployed information, i.e. generate the necessary service, prepare the Petri net logic, discover the required atomic services, etc.

At the time of the experimentation, there were only three available devices embedding Petri net orchestration engines, which one able to run one model at a time. Therefore, this situation represents a major problem when there are much more models to execute (e.g. one for each conveyor unit/lifter). However, this situation was the main motivation behind this work, involving both offline and online composition. The solution was using the offline composition to generate only three composed models (one for each orcestration device) and let them work together in real-time using the online composition.

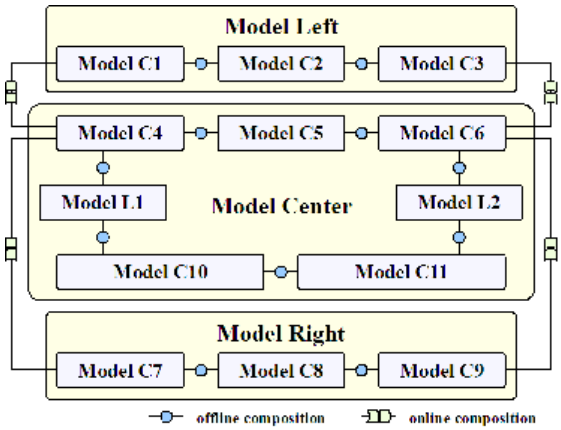

Fig. 10. Composition approach for the transport system case study.

Fig. 10 represents the composition applied to the system. Individual Petri nets models were developed for each unit (C1-C11 and L1-L2). Most of them are simply copy&paste of others, only the device information is changed (e.g. C10 and C11 have the same logic, use the same service interface, but offer different services). Afterwards, the decision was to split the system into 3 clusters of units (to be representative of the limitation of 3 orchestration devices), resulting in the right side, center and left side of the transport system. This division was taken into account to make the offline composition, ending up in three composed Petri nets models (model left, model center and model right of Fig. 10). The last step was to configure the devices with the deployment tool, uploading the models and the other information to the devices. The system is then ready to receive pallets and orchestrate the transport system according to the pallet needs (defined in the product process plan information).

The composition application shows that it is possible to design individual models without knowing the availability and disposability of the final orchestration devices. The experiment shows one possible way to compose the system using three devices and a defined distribution, but it could also be done with a different number of devices and other ways of division.

Offline composition is used to limit the use of devices, network traffic, but introduces more complex models to be orchestrated (considering the limitations of embedded devices). Online composition is focused more on the distributed orchestration and the synchronization thereof. The correct division and use of the composition types depends always on the available resources, the optimization strategies and the layout of the system, but orchestration models can be individually developed without knowing this information.

V. Conclusions and Future Work

This paper discusses the composition of Petri nets models used to represent the process behavior of services in service-oriented automation systems. The composition of services allows creating new and more complex services; each individual service behavior being modeled using Petri nets, the composed model is by sure more complex. The proposed approach for the composition of Petri nets models considers two forms, namely the off-line composition and the on-line composition. Both compositions can be used depending on the design choices and available resources, but in both cases they maintain the original behaviour planned for the individual equipment. At the end, the whole composition represents the specification of the system made of several well specified elementar models. The re-use of the models is also achievable, where a model of an equipment class can be part of several compositions, without defining it from the scratch. The composition of Petri nets models in service-oriented systems was illustrated through an experimental case study, namely a flexible transport system made of conveyors and lifters. It shows that modeling individual units could be done before or in parallel to the specification of the control layout. This feature is a crucial issue in modern automation systems design.

Further work is planned to explore more in detail the features of composition, including automatic composition using semantics and the direct information from anounced devices. Improvements have to be done in the CDT tool to facilitate the composition process. In terms of analysis of Petri nets, a study must be taken on how the properties of individual models are reflected in the composition.

Acknowledgment

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

References

[1] A. Bepperling, J. Mendes, A.W. Colombo, R. Schoop, A. Aspragathos, "A Framework for Development and Implementation of Web Service-based Intelligent Autonomous Mechatronics Components", Proc. of the IEEE Conf. on Industrial Informatics, 2006, pp. 341-347.

[2] A. Cannata, M. Gerosa and M. Taisch, "A Technology Roadmap on SOA for Smart Embedded Devices: Towards Intelligent Systems in Manufacturing", Proc. of the IEEE Intern. Conference on Industrial Engineering and Engineering Management, 2008, pp. 762-767.

[3] N. Milanovic and M. Malek, “Current Solutions for Web Service Composition”, IEEE Internet Computing, 8 (6), 2004, pp. 51-59.

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

[5] T. Murata, “Petri nets: Properties, Analysis and Applications”, IEEE, 77, 1989, pp. 541-580.

[6] A. Desrochers and R. Al-Jaar, “Applications of Petri Nets in Manufacturing Systems: Modeling, Control and Performance Analysis”, IEEE Press, 1995.

[7] R. Zurawski and M. Zhou, "Petri nets and Industrial Applications: A Tutorial", IEEE Transactions on Industrial Electronics, 41 (6), 1994, pp. 567-583.

[8] IEC 61131-3, “Programmable Controllers - Part 3: Programming Languages”, 2003.

[9] J.M. Mendes, A. Bepperling, J. Pinto, P. Leitão, F. Restivo and A.W. Colombo, “Software Methodologies for the Engineering of Service-oriented Industrial Automation: The Continuum Project”, Proc. of the 33rd IEEE Conf. on Computer Software and Applications, 2009, pp. 452-459.

[10] I. Melzer, et al., “Service-orientierte Architecturen mit Web Services”, 2. Aufl., Elsevier, Spektrum Akademischer Verlag, 2007.

[11] K. J. Ma, "Web Services: What's Real and What's Not?", IT Professional, vol. 7, n. 2, 2005, pp. 4-21.

[12] Web Services Description Working Group, Information about WSDL available at http://www.w3.org/2002/ws/desc/ (21 January 2010).

[13] M. Brenner M. Unmehopa, "Service-oriented Architecture and Web Services Penetration in Next-generation Networks", Bell Labs Technical Journal, 12 (2), 2007, pp. 147-159.

[14] The Devices Profile for Web Service Specification, OASIS Web Services Discovery and Web Services Devices Profile (WS-DD) TC.

[15] S. Pruter, G. Moritz, E. Zeeb, R. Salomon, D. Timmermann, F. Golatowski, "Applicability of Web Service Technologies to Reach Real Time Capabilities", Proc. of the 11th IEEE Intern. Symposium on Object Oriented Real-Time Distributed Computing, 2008, pp. 229-233.

[16] R. Hamadi and B. Benatallah, “A Petri net-based Model for Web Service Composition”, Proceedings of the 14th Australasian Database Conference, 2003, pp. 191-200.

[17] M. Nanda, S. Chandra and V. Sarkar, "Decentralizing Execution of Composite Web Services", Proc. of the 19th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages and Applications, 2004, pp. 170-187.

[18] I. Santos, M. Fluegge, N. Tizzo and E. Madeira, "Challenges and Techniques on the Road to Dynamically Compose Web Services", Proc. of the 6th Intern. Conf. on Web Engineering, 2006, ACM, pp. 40-47.

[19] E. Karakoc, K. Kardas, P. Senkul, "A Workflow-Based Web Service Composition System", Proc. of the IEEE/WIC/ACM Intern. Conf. on Web Intelligence and Intelligent Agent Technology, 2006, pp. 113-116.

[20] I.M. Delamer and J.L. Martinez Lastra, "Ontology Modeling of Assembly Processes and Systems using Semantic Web Services", Proc. of the IEEE Conf. on Industrial Informatics, 2006, pp. 611-617.

[21] Y. Zhao, J. Zhang, L. Zhuang and D. Zhang, "Service-oriented Architecture and Technologies for Automating Integration of Manufacturing Systems and Services", Proc. of the 10th IEEE Conf. on Emerging Technologies and Factory Automation, 2005, pp. 350-355.

[22] A. Pohl, H. Krumm, F. Holland, I. Luck and F. Stewing, "Service-Orientation and Flexible Service Binding in Distributed Automation and Control Systems", Proc. of the 22nd Intern. Conf. on Advanced Information Networking and Applications, 2008, pp. 1393-1398.

[23] W. Shen, Y. Li, Q. Hao, S. Wang and H. Ghenniwa, "Implementing Collaborative Manufacturing with Intelligent Web services", Proc. of the 5th Conf. on Computer and Information Technology, 2005, pp. 1063-1069.