Visual Programming And Development Of Manufacturing Processes Based On Hierarchical Petri Nets

Daniel Losch and Jürgen Roßmann
Institute for Man-Machine-Interaction
RWTH Aachen University
Aachen, Germany
{Losch, Rossmann}@mmi.rwth-aachen.de

***Abstract*—In this contribution we present a formalism to combine the well-known process modeling technique based on Petri nets with the recent developments in Visual Programming for robot programming. The resulting modeling approach enables process developers and robot programmers to follow a hierarchical approach to process development and to profit from the extensive analysis techniques developed for Petri nets, as well as the specialized robot programming features of the Visual Programming system. We applied our approach to an exemplary robot manufacturing process, demonstrating its modeling capabilities.**

Keywords – Petri nets; simulation; process development; visual programming; robotics; manufacturing

I. INTRODUCTION AND RELATED WORK

Efficient modeling of workflows is a crucial part of simulating manufacturing processes. Using Petri nets [1] is a popular approach to workflow modeling, because of their formal semantics, graphical nature, expressiveness, and vendor independence, amongst other reasons [2]. In many workflow modeling approaches that utilize Petri nets, workflows are modeled as state machines with places representing states, transitions representing assembly sequence steps and marks (that define the current state of a Petri net) are representing resources or partially assembled products [3], [4]. However, in robot simulation it is not always sufficient to have a high-level, event-based representation of a process. In order to conduct a detailed simulation of movements and manipulations in these processes (e.g. for Virtual Commissioning), a low-level representation of the different process steps is required. Such low-level Petri nets may also operate on either quasi-continuous or event-based time schemes, depending on the type of action they represent. In order to be able to comfortably represent arbitrary layers of abstraction and different timing models, the concept of hierarchical Petri nets can be applied [5], where the top layers represent either full sub-processes or complex process steps and the bottom layers describe the algorithms that define the execution of the upper-layer process steps. The use of

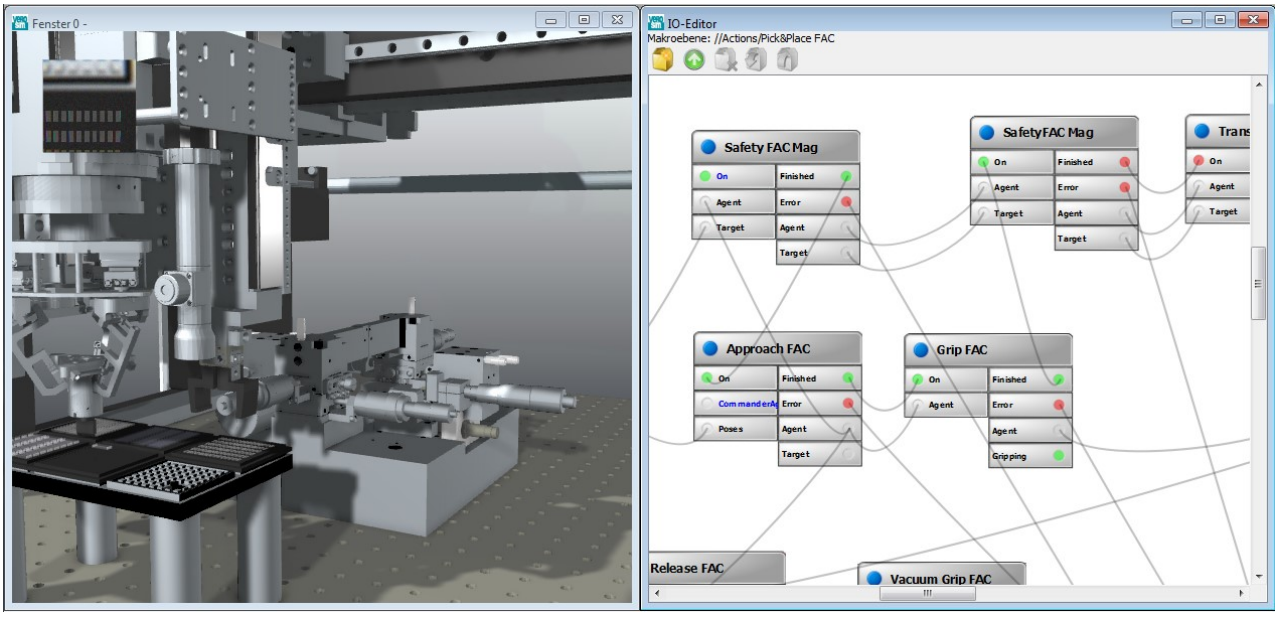

Figure 1: An assembly process modeled by ActionBlocks

978-1-5090-3696-7/16 $31.00 © 2016 IEEE
DOI 10.1109/ISCMI.2016.12

hierarchical Petri nets for process modeling is not exclusive to robot programming, e.g. Berkhan et al. proposed hierarchical Petri nets for process modeling in civil engineering [6].

Because of their graphical nature, Visual Programming techniques are especially useful when modeling Petri nets. Visual Programming is an approach to robot programming and robot process development that is generally more concrete, direct and explicit than text-based programming methodologies [7]. It also offers direct visual feedback, which is especially important for inexperienced programmers and developers that prefer prototypical approaches.

As a part of the previous work in the DeLas project, Schlette et al. developed a Visual Programming framework, focusing on its use for robot programming in the context of micro-optical assembly [8] (see Fig. 1), including Virtual Commissioning of assembly processes [9]. The central components of the Visual Programming framework are so-called ActionBlocks. ActionBlocks represent actions of robot agents and utilize their pre-programmed behaviors which range from simple actions such as PTP movements to complex actions, for instance gripping or timed glue dispensing. Processes are modeled by networks of ActionBlocks, and control flow is modeled by the connections between binary data ports that signal whether an ActionBlock can be executed or has already finished. It is also possible to combine ActionBlocks to macros to encapsulate existing action sequences into compact packages. ActionBlock networks can also be generated automatically from the results of symbolic planning algorithms that find optimal or near-optimal action sequences for given scenarios [10], [11].

ActionBlocks have been developed using the State Oriented Modeling Language++ (SOML++), a domain specific language for object-oriented Petri net definition and simulation scripting, which is fully integrated into the underlying VEROSIM simulation system [8]. Thereby, ActionBlock have access to all simulated entities, their properties and their functionality via VEROSIMs meta data management system [12], [13]. They can easily interface other simulation components, such as dynamics or sensor simulation libraries. The Petri nets contained in ActionBlocks are animated based on events or timing schemes, depending on the desired simulation behavior. Timing-based animation is useful to simulate quasi-continuous actions such as PTP movements, while event-based timing is useful for quasi-instantaneous actions such as the activation of a vacuum gripper. The effects on the simulation environment are programmed inside code blocks that can be added to transitions. Whenever a transition fires, the corresponding code, e.g. the manipulation of robot joint values during a PTP movement, is executed. These code blocks, as well as the corresponding firing conditions, can also utilize pre-existing simulation modules such as dynamics or kinematics frameworks, or control hardware via a simulation-based control mechanism [14].

However, there currently exists no formalism that combines the Visual Programming paradigm of ActionBlocks and the well-known methods of hierarchical process modelling with Petri nets. Such a combination would pose a viable starting point for further analyses or applications, and could combine the clearness of the Petri nets with the simulation capabilities of the ActionBlock approach under the accessible umbrella of a Visual Programming system.

In this contribution, we will present a formalism that aims to achieve such a symbiotic combination. The formalism is based on the mapping between hierarchical Petri nets and ActionBlock networks, as well as on the flexibility and expressivity of ActionBlocks defined in SOML++.

In section II, we discuss this formalism in detail, followed by an application example to show the practicability of the approach (see section III). We finish this publication with a conclusion and an outlook on potential future research (see section IV).

II. Visual Programming of Robotic Workflows and Processes Using Hierarchical Petri Nets

A manufacturing process can be examined in multiple different abstraction layers. The highest abstraction layer is formed by the process as a whole, consisting of different process steps that represent the tasks included in the process. These process steps are still rather high-level, e.g. picking and placing of components. The process steps themselves consist of series of complex actions such as picking or placing. The complex actions are eventually built out of basic robot actions such as PTP movements. Hierarchical Petri nets are a tool to model these abstraction hierarchies. It is desirable to utilize the hierarchical Petri net formalism in the ActionBlock Visual Programming system as well, but it is necessary to find a mapping between the elements of these two formalisms to do so.

While defining this mapping, we decided to differentiate between the higher abstraction layers of a process and the lowest, least abstract layer. The lowest abstraction layer of a process consists of often-repeated, basic robot actions, while the higher abstraction layers consist of complex, process-specific elements. Therefore, it is sufficient to map the higher, process-dependent abstraction layers to the ActionBlock Visual Programming system, while the basic actions can be pre-programmed outside the Visual Programming

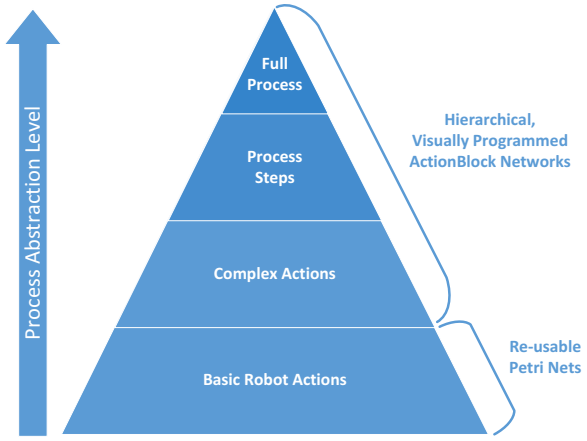

Figure 2: Abstraction hierarchy of process models and their respective proposed modeling techniques

environment by programming them with low-level, re-usable Petri nets that are directly included in the ActionBlocks. Fig. 2 shows the hierarchical process structure and the layers' proposed representation in our Visual Programming system.

In section II.A, we present a formalism to map hierarchical process descriptions based on Petri nets to ActionBlock networks, and in section II.B, we discuss the implementation of basic robotic actions as Petri nets that are contained inside ActionBlocks.

A. *High-Level Modeling of Robotic Processes With Implicit Petri Nets*

At a first glance, networks of ActionBlocks, do not contain Petri net components, and do not form Petri nets. However, it is possible to directly map Petri net-based process descriptions to ActionBlock networks, and the resulting networks implicitly contain Petri nets that represent different abstractions of manufacturing processes. In order to utilize Petri net approaches to robot process modeling inside the ActionBlock Visual Programming environment, we define a mapping between the Petri net formalism and the corresponding ActionBlock networks. Examples for different parts of the mapping are given in Fig. 3.

Considering the notion that transitions of process-describing Petri nets correspond to different complex actions that form the process, it follows that their equivalents must be ActionBlocks, as they represent the same semantics. The marks that define the current Petri net state do not have an explicit counterpart in the ActionBlock networks. Instead, the state of a network is determined by the ActionBlocks that are currently being executed. The Petri net equivalent to this notion is a mark that is currently passing a sub-network contained in one of the Petri nets' transitions. The translation of basic Petri net graph structures is also possible: Sequencing is done by simply connecting control flow data ports. Parallel actions, started by multiple outgoing arrows from a Petri net transition, are represented by parallel ActionBlocks executed by different agents, and started by connecting the control flow data ports of multiple ActionBlocks to the same predecessor. The re-synchronization (multiple places leading to a combined transition) in the Petri net is mapped by a logic AND gate that activates the following ActionBlocks only if the predecessors have all finished. A Petri net choice, modeled by multiple active transitions that follow a single place, is mapped by multiple outgoing data lines of a single predecessor, connected to different ActionBlocks. The preceding ActionBlocks have to employ internal logic to determine the next following ActionBlock, e.g. after reaching an error state, and communicate their choice by selecting the corresponding outgoing data port. The re-joining of different Petri net branches is simply mapped by an OR gate combining the potential predecessor of an ActionBlock.

In order to represent the hierarchical structure of Petri nets and thereby model processes on different levels of abstractions, the ActionBlock Visual Programming framework allows for the creation of macros that encapsulate ActionBlock networks into new ActionBlocks. Using this mechanism, one can combine basic robot actions to complex actions, complex actions to process steps and finally process steps to a full process. The movement of a mark into a sub-transition of a Petri net is mapped to the ActionBlock formalism by exporting the data connection ports of the first and last ActionBlocks to the overlaying macro.

B. *Low-Level Modeling of Robotic Process Steps With Explicit Petri Nets*

The lowest level of abstraction in the process model consists of basic robot actions that are used as parts of multiple different complex actions, e.g. movement or gripping actions. These basic actions are process-independent and are re-used frequently, between both different processes and within a single process. Therefore, we propose to mask their implementation from the Visual Programming system, and define their behavior inside the ActionBlocks by developing

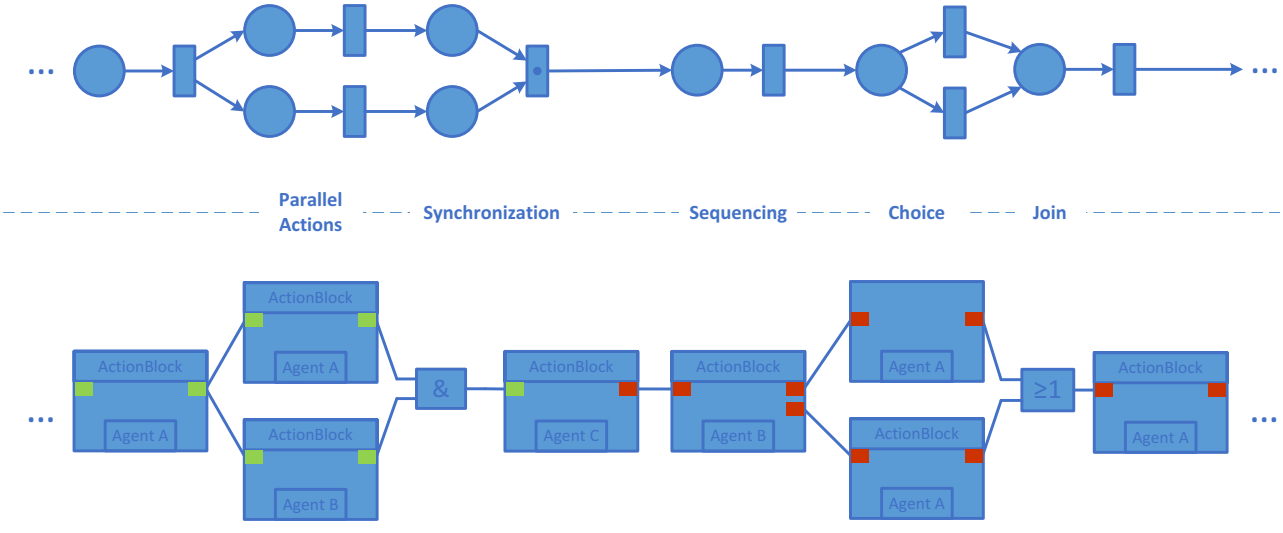

Figure 3: A segment of an exemplary Petri net and the equivalent segment of an ActionBlock network, demonstrating the direct mapping of different Petri net structures to ActionBlock networks.

low-level Petri nets directly in SOML++. By representing the basic actions that way, we can exploit the close integration of SOML++ into the simulation system. We propose to model the basic actions as Petri net state machines [15], since the SOML++ language was originally developed for this purpose. The effect of an ActionBlock's execution is defined by the code blocks inside the transitions of the internal Petri net. Due to the simplicity of the basic robot actions, this method is completely sufficient to program the simulated execution of an ActionBlock. Furthermore, if more flexibility is needed, it is always possible to include multiple independent state machine Petri nets into a single ActionBlock and synchronize them via firing conditions, event listeners, or inhibitor arcs.

In order to be flexible and re-usable, the code inside the transitions utilizes the surrounding ActionBlocks' data ports to adapt their behavior to the configuration modeled in the Visual Programming system on the higher abstraction layers. Prime examples of this dependency on the ActionBlock configuration are the assigned agent – the Petri net has to utilize the assigned agent and its behaviors – and target poses for movement actions. Furthermore, the low-level Petri nets have to react to and correctly configure the control flow data ports of the containing ActionBlock in order to act as the hierarchical sub-Petri net of the implicit Petri net defined on the abstraction layer of complex actions.

III. Application Example

To illustrate the hierarchical Petri net–based approach for process modeling, we selected an exemplary micro-optical assembly process examined during the DeLas project. During the process, two lenses (one fast and one slow axis collimation lens) and two side tabs are picked and placed from their component magazines to a mounting station, where they are held by vacuum grippers. Then, glue is applied to the side tabs, and the mounting station assembles the components to a lens system, which is then picked and placed from the mounting station to its own magazine. The process is implemented using a gantry kinematic and the precision gripper Commander C6 (developed by Fraunhofer IPT, [16]) as the main robot agents. A picture of the process simulation is shown in Fig. 1, while the process modeling is demonstrated in Fig. 4 and will be explained in detail:

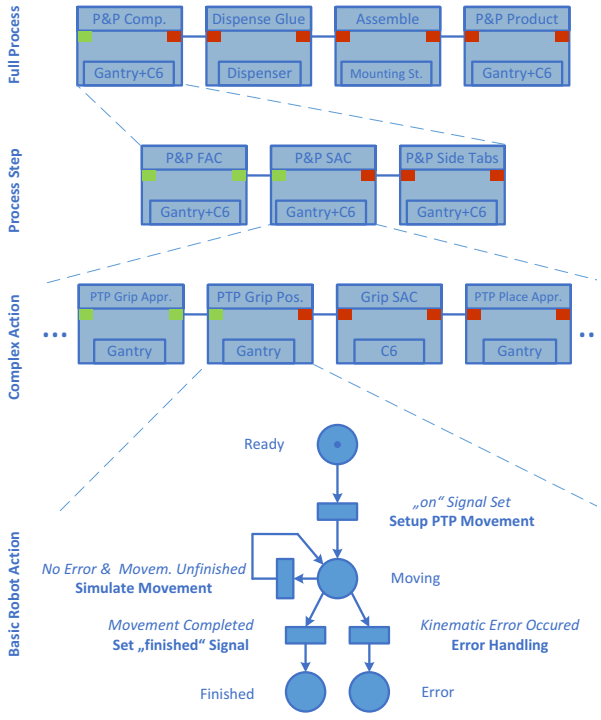

Figure 4: An exemplary Petry net hierarchy, modeling a micro-optical assembly process.

On the highest abstraction layer, the process is modeled using macro-combined ActionBlocks that represent the previously mentioned process steps. On the second layer, the inner structure of the pick and place process step is shown, consisting of three complex actions for the picking and placing of the different types of components. The third layer consists of an excerpt of the complex pick and place process of the SAC lens. This is the least abstract layer that is still represented by a structure of ActionBlocks, and consists of simple robot actions such as PTP movements, gripping, releasing, etc.

The internal Petri net, defined in SOML++, of a PTP movement ActionBlock, is shown at the bottom of the figure. The net has the form of a state machine, and the transitions contain conditions (denoted in italic font next to the transitions) and code blocks (denoted in bold font) that contain the main simulation logic. Since PTP movement control a simple action, the controlling Petri net only consists of a transition that initiates the movement, another (timed) transition that supervises and regularly controls the movement, and two different transitions that end, resp. abort the movement in case of an error. Although there seems to be a transition activation conflict when "Moving" is marked, configuring the three outgoing transitions with mutually exclusive conditions avoids nondeterministic behavior.

As required by the ActionBlock formalism, the execution of the PTP movement Petri net starts in reaction to receiving a data signal from the "on" input, and correspondingly signals a successful movement by using the "finished" data output, thereby starting the next ActionBlocks' execution.

IV. Summary and Future Work

This contribution introduces a design formalism for assembly processes that combines Petri nets and a Visual Programming framework. In order to allow for the hierarchical development of processes on all potential layers of abstraction, we presented a method that differentiates between the representations of re-usable, basic robot actions and higher-level complex actions or process steps. In order to utilize the Visual Programming framework for high-level processes while still being able to use the well-known Petri-net formalism, we developed a direct mapping between ActionBlock networks and Petri nets. We also gave a detailed description of the Petri nets that describe the low-level process steps, and their correct integration into the ActionBlock framework. The combination of modeling methods for all abstraction layers allows for complete process development

utilizing the ActionBlock approach while still keeping the advantages of a Petri net-based hierarchical process modelling formalism.

Future research in the ReconCell project will focus on potential process analysis methods for business process modeling and business intelligence, and on utilizing the approach in other fields than micro-optical assembly.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 680431 (ReconCell). The authors would like to thank Dr. Christian Schlette for sparking the idea of ActionBlocks.

REFERENCES

[1] D. Abel. 1990. Petri-Netze Für Ingenieure: Modellbildung und Analyse diskret gesteuerter Systeme. 1st ed. Springer-Verlag Berlin Heidelberg.

[2] W. M. P. Van Der Aalst. 1998. "The Application of Petri nets to Workflow Management" Journal of Circuits, Systems and Computers 08 (01). World Scientific Publishing Company: 21–66.

[3] W. M. P. Van Der Aalst. 1996. "Three Good Reasons for Using a Petri-Net-Based Workflow Management System." In Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96), 179–201.

[4] D. Dubois, and K. E. Stecke. 1983. "Using Petri Nets to Represent Production Processes." In The 22nd IEEE Conference on Decision and Control 1983, 1062–67.

[5] M. Heiner, P. Deussen, and J. Spranger. 1999. "A Case Study in Design and Verification of Manufacturing System Control Software with Hierarchical Petri Nets." The International Journal of Advanced Manufacturing Technology 15 (2). Springer: 139–52.

[6] V. Berkhahn, A. Klinger, U. Rueppel, U. F. Meissner, S. Greb, and A. Wagenknecht. 2005. "Process Modeling in Civil Engineering Based on Hierarchical Petri Nets." In Proc. 22nd Conference on Information Technology in Construction CIB-W78, Dresden.

[7] M. Burnett,. 2001. "Software Engineering for Visual Programming Languages." Handbook of Software Engineering and Knowledge Engineering 2.

[8] C. Schlette, D. Losch, and J. Roßmann. 2014. "A Visual Programming Framework for Complex Robotic Systems in Micro-Optical Assembly." In ISR/Robotik 2014; Proceedings of the 41st International Symposium on Robotics, 1–6.

[9] C. Schlette, D. Losch, S. Haag, D. Zontar, J. Roßmann, and C. Brecher. 2015. "Virtual Commissioning of Automated Micro-Optical Assembly." In SPIE LASE, 93460I – 93460I.

[10] N. Wantia, D. Losch, and J. Roßmann. 2014. "Symbolic Planning for Industrial Applications - The eRobotics Approach." In 2014 IEEE International Conference on Automation Science and Engineering (CASE), 367–72.

[11] N. Wantia, D. Losch, and J. Roßmann. 2015. "Virtual Commissioning and Symbolic Planning of Micro-Optical Assembly Processes." In 2015 7th Computer Science and Electronic Engineering Conference (CEEC) (CEEC'15). Colchester, Essex, United Kingdom.

[12] M. Schluse. 2002. Zustandsorientierte Modellierung in Virtueller Realität Und Kollisionsvermeidung. PhD thesis. VDI-Verlag.

[13] J. Roßmann, M. Schluse, and R. Waspe. 2012. "Integrating Object Oriented Petri Nets into the Active Graph Database of a Real Time Simulation System." In Proceedings of the Winter Simulation Conference.

[14] E. Kaigom, and J. Roßmann. 2014. "Interactive Physical Robot Guidance through Advanced 3D Dynamic Simulation-Based Robot control—A New eRobotics Approach." In 2014 IEEE International Conference on Industrial Technology (ICIT), 676–81.

[15] T. Murata. 1989. "Petri Nets: Properties, Analysis and Applications." Proceedings of the IEEE 77 (4). IEEE: 541–80.

[16] C.Brecher, N. Pyschny, S. Haag, and V. Guerrero Lule. 2012. "Micromanipulators for a Flexible Automated Assembly of Micro Optics." In SPIE Photonics Europe, 84280J – 84280J.