Decision Support System for Petri Nets Enabled Automation Components

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

***Abstract*-The expected behavior of industrial automation systems can be defined and modeled, but since not every event is predicted at design time, additional care has to be taken during the operation to handle situations such as exceptions, failures and new production orders. Another remark is also the limitation of modeling languages used in the control to permit the intervention of higher grade decision mechanisms. This paper discusses the application of decision support system for Petri net based processes to control automation components and devices. The decision mechanisms are used for path planning, production scheduling and preventive maintenance, to support the description of processes in Petri nets formalism. As such, the same Petri nets designed for the control are also used as analytical information input to the decision support system and for the detection process options that require decision. The solution provides a dynamic complement to the static modeling and operational flexibility. Experiments were done in a real service-oriented industrial factory-cell to prove the specified approach.**

I. INTRODUCTION

Tomorrow's production systems will face challenges which are even tougher than those present today. New business forms emerge, such as the movement towards strong collaboration within the supply network [1]. Therefore, from the careful planning to the operation, processes are used to specify the system's behavior according to the planned objectives. They may be local to devices used as the control mechanisms or represent interoperability relations between distributed components. Processes are described manually using some representation or language (e.g. Ladder logic, Petri nets, C programming) or dynamically and automatically created by intelligent software entities.

At design time, the system is modeled by specifying the expectations in terms of how the system should be controlled and how processes are executed. But complex manufacturing systems are dynamic and stochastic (i.e., new orders arrive over time, machine break-downs happen, or the priority of customer orders may change) [2], and not all parameters can be considered in the beginning at design time. Additionally, a system should not only be defined by a single path of operations but also permit having different branches and choice possibilities. In any case, processes must become more flexible in order to cope with constant product changes and increasingly volatile demand [3], besides permitting some elasticity for real-time decisions that can not be done before.

The systems that are targeted in this work face also the same problems concerning process definitions, distributions and decisions. These systems are characterized by the adoption of decentralized distributed automation, with manufacturing resources composed of intelligent modules that can be easily reconfigured to suit evolving application needs. This collective functionality distributed across many mechatronic system devices and machine controls replaces the logical programming of manufacturing sequences and supervisory functions in traditional production systems [4]. Especially for automation systems guided by service-orientation [4], processes should be created manually or automatically considering the possibility to choose the best available service for a current situation, simplification and aggregation of services, besides others.

One form of describing behavior by a formal language is Petri nets [5]. Petri nets present an alternative choice to the traditional IEC 61131-3 languages, by having a wide applicability and, based on a strong mathematical foundation, they can be used for the analysis of models and processes and the rules for their execution. In industrial service-oriented automation, Petri nets are used to describe processes that represent behavior of system elements and relationships between provided services.

All these complex situations at runtime require a complement mechanism to decide over the application and execution of processes, in a form of *Decision Support Systems* (DSS). It is now a matter of question where the decision supporters should be employed. Global deciders may have the full view of the system, but introduces also hierarchical dependencies and thus reduces the reconfiguration capabilities. Local deciders that complement the control of a device are more concerned with lateral collaboration with other similar companions, enhancing the autonomy and the systems flexibility. A mix of both approaches can be used to balance the values of the several parameters, and contribute to the overall system's performance and flexibility.

The current work describes a decision support mechanism to complement the execution of Petri nets for distributed devices and software components. The outline of the paper is the following: After the introduction, section 2 describes the used approach of the decision support system for Petri nets,

section 3 introduces the application of the methodology for the used case study scenario and finally the paper is closed with the conclusions and future work.

II. A Decision Support System for Petri Nets

The following paragraphs presuppose that the reader is familiar with the basic concepts of Petri nets. Petri nets and their several extensions (designated as High-Level Petri nets [6]) are used to model systems where concurrency, resource sharing and information flow are important. Being a fixed way to represent a system, it includes mechanisms to introduce some flexibility. Decision points, alternative ways and other conflicts are characteristics that can be modeled in Petri Nets. A conflict can be viewed as a resource or state that is to be taken by more entities than its capacity or transitions that activate from the same state leading to different paths. Both of the two situations requires mechanisms that should pro-actively detect conflicts and resolve them [7]. The existence of conflicts does not strictly mean that there are design problems in our system, but should be also understood as an opportunity of applying decision to a more flexible system. Other appointment for advanced handling of Petri nets is the movement of the information along a net (represented by tokens), that has an extended meaning in colored Petri nets [6].

Such features require the intervention of decision mechanisms that extend the indirection of the static and predicted control. This is especially valid when considering that automation systems are inherited dynamically and not fully controlled; sometimes there are unexpected circumstances that a simple executed Petri net can not handle: operation's delay and canceling, synchronization among individual workflows, unexpected situations, unaccomplished operations, dynamically adding new operations, etc. In these situations, a DSS provides support to resolve them.

The following topics describe where additional help is needed in terms of decision, to increase the power of Petri nets:

- Selection of the firing transition between several ones that are in conflict (resolution of conflicts);
- Petri nets analysis to support decision, including behavioral and structural analysis, path finding and simulation;
- Selection of the best service or operation to execute when the associated transition is enabled or firing;
- Management of the Petri net: decide when to run, stop, and reset a Petri net;
- Automatic composition and aggregation of Petri nets.

The degree of complexity associated to the decision-making instance can range from simple algorithms to complex cognitive systems, such as multi-agent systems, neural networks and genetic algorithms. There may be different ways to make a DSS more intelligent; the most frequently suggested method is to integrate a DSS with an expert system [8]. Although, in order to create a distributed system and considering that the DSS is going to be implemented in embedded devices, we must be aware that this kind of systems are strictly restricted in terms of memory and processing power [9].

A general architecture of a service-oriented device or component with a DSS is depictured in Fig. 1. The DSS provides two kinds of decisions to the Petri net controller. The most basic one, the *passive decision*, is requested by the Petri net itself (for example, several transitions in conflict) to the DSS and awaits an answer. From the other hand, the *active decision* is pro-actively provided by the DSS in several situations (e.g. analyzing parameters that are over a determined threshold, such as temperature, and intervene as necessary). The DSS can be decomposed in three parts, as it will be described later on this section. In terms of behavior analysis, a simple architecture can be described as a device with a predicted control (from the Petri net) and an unpredicted control (from the DSS). Therefore, the combination of the two advance mechanisms were thought to enhance the autonomy of the component. This approach is also used to enable the component to interoperate with other components in the system, by providing and requesting services, in a form of mutual collaboration. Overall, it helps in the systems flexibility to respond and adapt to new situations, and consequently reconfigure as necessary.

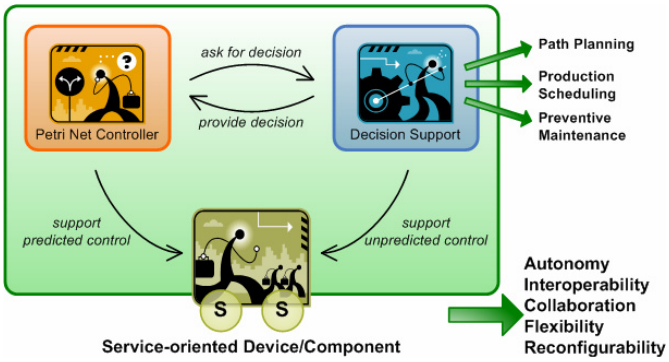

Fig. 1. General specification of decision mechanisms to support the application of Petri nets.

Even though the device is autonomous enough with its DSS to find its own services, it does not have any topology knowledge in order to know which services are also available through the rest of the system. Avoiding the enormous volume of messages to find the available services in a reactive manner and avoiding a static configuration where the inclusion or exclusion of other devices would need a total reconfiguration of the system, the DSS contains *Neighborhood Knowledge* (NK). The NK means that every time a new device is included in the system, it broadcasts a message to all others devices with its connections to the others and the available services. Since the topology of the system will only change each time a device is introduced or removed from the system, the messages exchange will only need to take place in that specific time. This way is possible to have a type of "plug&play" based service-oriented devices in a distributed and flexible system.

The ability to make use of the systems flexibility and to retain flexibility itself is a primary requirement for the decision tools to be effective [10]. Based on that, the proposed architecture for a DSS has the primary objective to represent reality as simple as possible but as detailed and flexible as necessary, i.e. without ignoring any serious real world constraints [11]. The following subsections describe the application topics of the decision support system that are considered in this work.

A. *Path Planning*

Knowing that service-orientation is basically a paradigm that defines mechanisms to publish, find and bind services [4], one of the goals of the DSS is to know how to reach required services at a given moment. In order to do so, and aware that each device in this case also integrates a Petri Net controller, the DSS module needs to have knowledge of the paths of the system. With the *Path Planning*, it is possible to return the best path for e.g. a pallet to go from its current position to the place where it can execute the next service. With this approach, it would be possible to receive e.g. "request_service" from a pallet, find out which is the best way to get to the needed service, and forward the next position the pallet should take.

In order to implement this architecture, the path planning needs to extract the following information from the Petri Net: *Market place* – A place market as being part of a path means a physical spot where a pallet can be. A place may also describe a service that can be executed; *Transition* – A transition means a possible move from one place to the other; *Path* – A path is a group of places and transitions that can route the pallet from one physical spot to another, without any conflict in the Petri net (meaning that during the path the Petri Net is able to route the pallet all the way through it).

This implementation also implies the need of one and only one transition to connect two places, and that one transition can only connect to one place, although a place can connect to one or more transitions. With this architecture, we ensure that a transition can only be assigned to one path, meaning that a service will only exist in one path.

To initialize the path planning, the DSS needs to receive two matrices that correspond to the input and output relations of transitions with the places of the Petri net. In order to create all possible paths, it is necessary to generate a structure with all the different kind of places that are considered in this method: *Start place* – A place that is not output of any transition. This means that this place is going to be a start place of one path at least; *Termination place* – A place that is not input of any transition. This means that this place is going to be a finish place of one path at least*; Join place* – A place that is output of more than one transition. If this place is output of *n* transitions, than it is going to be the finish place of *n* paths*; Conflict place* – a place that is input of two or more transitions. If this place is input of *n* transitions, then it is going to be the start place of *n* paths. With this approach the path planning already has all the places (joints and conflict or just conflict) where a decision support is going to be necessary.

An example of the creation of the path planning can be seen in Fig. 2, where the original Petri net is transformed into a simplified one with just the required places. Here just the places p4 (join place), p5 (join and conflict place), p7 (join and conflict place) and p8 (conflict place) where preserved.

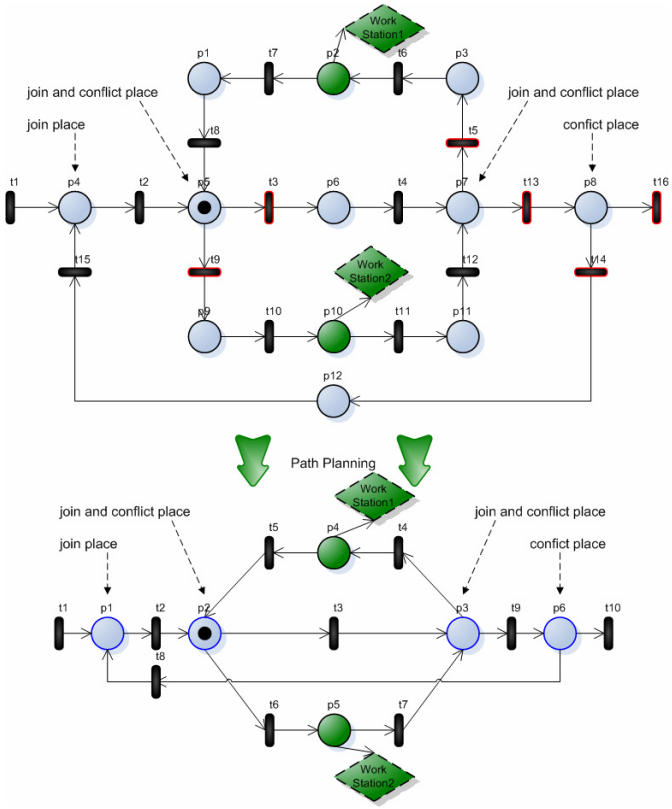

Fig. 2. Creation of the path planning from a Petri net with join and conflict places.

Although the identification of the joint place does not look necessary, the next sub-section will explain how its inclusion will prevent the system from deadlocks.

B. *Production Scheduling*

After the successful implementation of the path planning, the DSS will start a scheduled update of the system – *Production Scheduling*, by getting the state of the system each time a complete evolution in all the transitions of the Petri net occurs. With the NK, the DSS knows the services that can provide and which services are available in the rest of the system. Every time a pallet is in a conflict place, the Petri net will question the DSS which transition should be activated. Although the DSS knows all the ways (the meaning of way here corresponds to a sequence of one or more paths) for the next service, multiple possibilities can occur. The question is: Which way is the best one?

In order to choose the best way, the DSS will need more information on the paths: *Availability* – Means how many places are still available in the path; *min_time* – Will correspond to minimum time that a pallet took to pass

through the path; *mean_time* – Will correspond to the mean time that a pallet takes to pass through the path; *last_update* – The information kept here will correspond to the last time the information "mean_time" was updated.

Although this information is sufficient for the DSS to choose the best way to route a pallet, a detailed explanation on how to use this data is needed. If the requested service can be achieved by multiple ways (which is the case where this information will be needed), then the path planning will retrieve a list with all possible ways. The availability information will narrow down the choices to the ways where the first path after the current position of the pallet has positive availability. Then the best way will be the one that has the lowest total time. Since each way may have more then one path, the total time must be the sum of the mean time of all paths in that way. Finally, after the best way chosen, the DSS can find which transition should be activated by searching the one that makes the connection to the first path.

Even though this approach seams feasible, a problem may occur: If a path increases it mean time, due to some technical problems or simultaneous request of the same service, it may not be chosen again in the future. That is where the "last_update" information will be useful, meaning that if the mean time of one path has not been updated for a long time, then the DSS will start decreasing its mean time until the path is chosen again or until it reaches its minimum time.

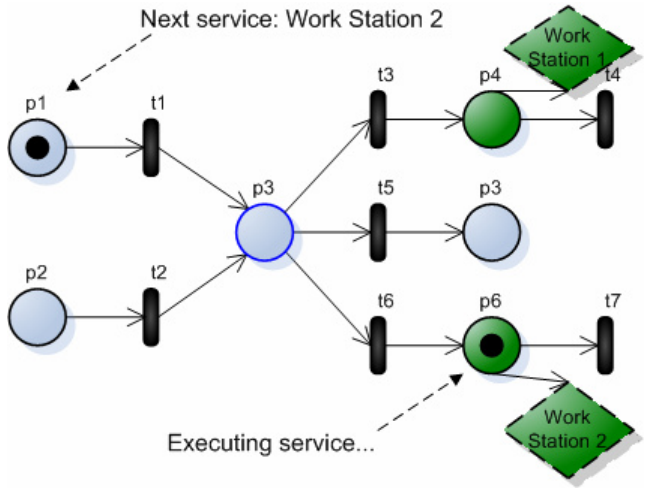

Fig. 3. Example of a possible deadlock resolution in place p3 provided by the DSS.

With the path planning not only it is possible to know which services are available in the other devices/components, it is also possible to know which transitions connect the current device with the others. Those transitions will be described as ports. The ports can be input or output and those that are connected to other devices must store their names or IDs. Each time a pallet passes by one of those transitions, a message must be sent to the other device with all the information about the pallet. This way, each time a new pallet enters the other device by a port, the device's DSS will receive a message with the information about the pallet. If an external device introduces a new pallet in the system, it must use the same type of message and send it the correspondent device's DSS.

With the aim of improving the system performance and avoiding possible deadlocks the DSS must also be able to stop a pallet one place before a joint and conflict place. This must be done every time the next path of the pallet is full, otherwise that pallet could stop all others pallets from routing though that place, and even stop the entire system. An example is depictured in Fig. 3, where the pallet in place p1 must be stopped, otherwise it would obstruct the place p3, which can still route pallets from the place p2, to the other paths.

C. *Preventive Maintenance*

Even though the path planning and the production scheduling in the DSS can provide an increase of autonomy and distributed working system, nothing has been done when it comes to diagnosis.

In past projects, little attention has been paid to diagnosis [12]. Nonetheless in future assembly systems an effective diagnosis infrastructure can represent a competitive advantage increasing the overall robustness of the system, reacting timely to unpredictable situations and allowing the optimization of maintenance cycles. In order to achieve that next step, the *Preventive Maintenance* has been included in current DSS. Each time the production scheduling updates the system, this component must check if something out of the ordinary happened.

The unpredictable situations and its resolution are those that follow:

- A new pallet appears in a non-input port: In this case the DSS will send a message asking for information about the new pallet. If the DSS does not receive any feedback then the pallet is routed through a default path in order to not stop the system;
- A pallet disappears in a non-output port: In this case the DSS will send a warning message informing the situation;
- A pallet jumped through more than one place: If the pallet skips one place routing through one path, then a warning message is sent informing the situation (for example: One place sensor is not working).
- A service that does not exist in the system: Even though the DSS has NK, that may not be extended to the entire system (in order to reduce the complexity), meaning that a pallet may request a service that is not listed in the DSS. If that happens, the DSS will send a message requesting for another service. To continue the system properly, the answer that the DSS should receive is the port where the pallet should be routed to execute the desired service. Once again, if the DSS does not receive any feedback then the pallet is routed through a default path, so that the system does not stop.

The maintenance cycle is done by introducing new information into the device: *Tolerance* – It will be the maximum percentage accepted by the time a pallet takes to

pass through one path relatively to its minimum time. This means that a device that has a tolerance of 35% and a path with a minimum time of 40 seconds cannot exceed 54 seconds in that path. Each time that this situation happens, a warning message will be sent in order to alert the user or to update a monitoring system.

One improvement of the system would be to allow the subscription of events (i.e. informing an ERP – Enterprise Resource Planning – hourly the mean time of each path, or how many pallets have passed by each path or even which services where required and when).

Implementing these maintenance schemes allows: eliminating unnecessary maintenance, reducing production lost due to failures, reducing repair parts in inventory, increasing process efficiency, improving product quality, etc. Cutting costs, maximizing the profit, is therefore one of the main reasons for embedding diagnosis capabilities in future production systems [12].

III. Application of the DSS for a Case Study Scenario

The flexible production system used to illustrate the application of the DSS for Petri nets in the case study scenario consists in a FlexLink® Dynamic Assembly System (DAS) 30. The DAS 30 transfer system combines flow-oriented production control and modular automation with ergonomic manual assembly solutions, providing flexibility and versatility. The system is composed of several conveyors that route the pallets, two workstations that can be used by operators or robots (providing the services) and two lifters responsible for the boundary between an upper and lower level of the main line [13].

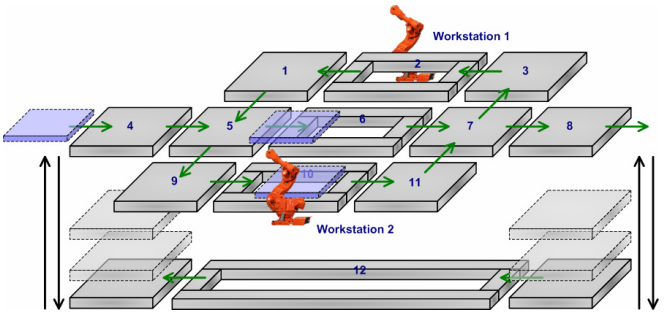

Fig. 4. Layout of the case study scenario.

After the decomposition of the DAS 30 into the several components (Fig. 4), the next step is to map the system into a Petri net and extract the information to create the path planning (Fig. 5). The approach used here to create the Petri net from the DAS 30 is described in Mendes et al. [13]. Although the conveyor devices can be Unidirectional or Cross types, for the DSS it does not make any difference since it will be built from the Petri net. In order to create the DSS properly the Petri net must also specify the available connections (ports) to the other devices, this can be seen in Fig 5 where the place 1 has an input port and place 6 has an output port, resulted from the external access in the conveyor 4 and 8, respectively, depictured in Fig. 4.

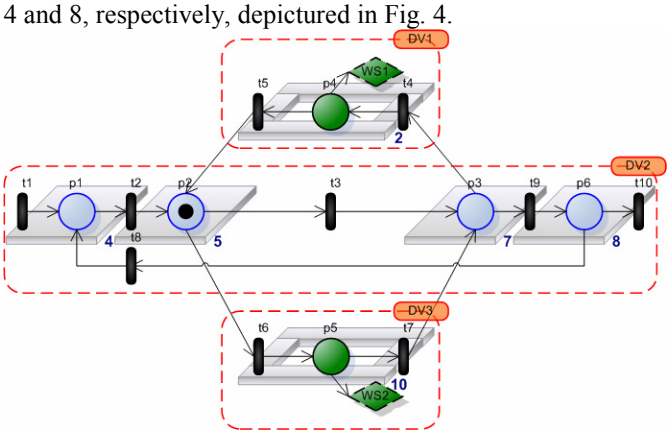

Fig. 5. Path planning information from the case study scenario.

The coordination of processes and services, depending on the flexibility that the system reveals, requires the decision-making and conflict resolution at runtime, because a system model does not describe a fixed sequence of actions, but rather all possible combinations thereof. On the other hand, it may also be necessary to choose from different available services that result from a filtered discovery matching specific criteria.

In order to successfully implement the case study scenario it is still required to deploy the Petri net of each device in each DSS. It would be possible to decompose each device in one physical component, but since it would require a lot more devices and it would not bring any advantages, it was decided to decompose the system in three devices (two of them with services available and another one just responsible for routing). The system does not have only one way to be configured and another approach would have given the same results, it is up to the user to decide the best approach for his system (granularity vs. devices cost).

For example, if a pallet arrives at device 5 that must be routed out, either to device 6, device 9 or stay for some time at device 5. This represents a conflict in the Petri net and the control will report this event to the DSS and await an answer. Considering that the answer is device 6, the Petri net controller activates the corresponding transition (t3 in Fig. 5) to synchronize the operation of both devices 5 and 6, and thus move the pallet accordingly.

As it was described in the previous section, the DSS needs to have neighborhood knowledge. With that in mind, there are two ways of deploying the Petri net in the DSS. The first one is to deploy the entire Petri net of the system to the DSS, with a property in each transition and place that specify to which device it does corresponds. This way the DSS will create the path planning from the transitions and places that belong to it and it will build the NK from the services that are available in the other devices (Fig. 5). The second way is to divide the Petri net for each device a priori, and then deploy it to the DSS. This way the DSS will start without any NK, and each time a new device is introduced in the system, it must

broadcast a message (with its current position and services available) to all other devices in order for them to update their NK. This approach is exemplified in Fig. 6, where the device 2 (the conveyor box represented in Fig. 4) starts without any NK, and after receiving the messages from device 1 and device 3, it is able to update its NK being aware of the other services available in the system.

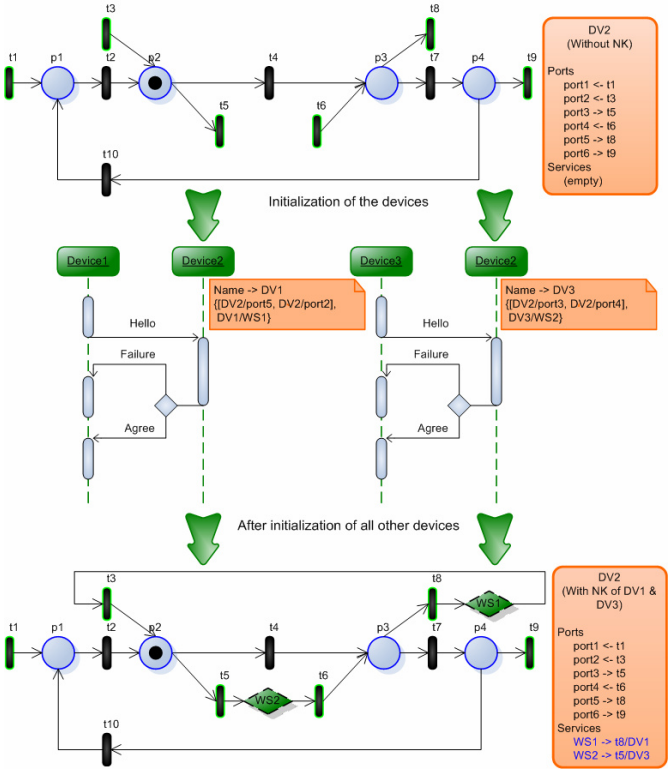

Fig. 6. Representation of device 2 building the NK with information provided by the other devices.

Both ways can be used to deploy the Petri net into the DSS. It is even possible to use the first approach to initiate the system and later on, during run time update the system with the introduction of a new device and using the second approach to broadcast a hello message. Once again, it is up to the user to decide which way to adopt. Aware that the first approach, in one hand will require less time to initiate the system, since no exchange of messages is needed, on the other hand it has higher processing and resource requirements, particularly if there is a large number of devices in the system, or if the system is composed by a very complex Petri net.

IV. Conclusions and Future Work

The present work presents a decision support system for Petri nets enabled automation components. It was demonstrated with a real industrial scenario how decision mechanisms are used to support the control based on Petri nets, using the topics of path planning, production scheduling and preventive maintenance.

There are several research topics and developments that are planned for the future. Besides the improvement of the Petri net based controller for embedded devices, specific for this work is to enhance the decision support system with experimental neural networks for pattern classification and other mechanisms that can help the decision. Not to avoid is the intrinsic analysis of Petri nets, specially the structural analysis that can be used to gather information about sequence of transitions/services for a specific path and also places/resources with mutual exclusion. Since the focus is a distributed environment, it is also important to research in the way of distributed or collaborative intelligence to support the manual planning by automation engineers.

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] R. Frei, L. Ribeiro, J. Barata and D. Semere, "Evolvable assembly systems: towards user friendly manufacturing", Proceedings of the International Symposium on Assembly and Manufacturing, pp. 288–293, July 2007.

[2] L. Mönch and M. Stehli, "ManufAg: a multi-agent-system framework for production control of complex manufacturing systems", *Information Systems and E-Business Management*, vol. 4, n. 2, pp. 159–185, April 2006.

[3] S. Bussmann and K. Schild, "An agent-based approach to the control of flexible production systems", Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation, vol. 2, pp. 481–488, October 2001.

[4] A.W. Colombo, F. Jammes, H. Smit, R. Harrison, J.L.M Lastra and I.M. Delamer, "Service-oriented architectures for collaborative automation", Proceedings of the 32nd Annual Conference of IEEE Industrial Electronics Society, IECON 2005, 6 pp. –, Nov. 2005.

[5] C.A. Petri, *Kommunikation mit Automaten*, Doctoral Thesis, Bonn Institut fuer Instrumentelle Mathematik, Schriften des IIM, Nr. 3, 1962.

[6] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* Volume 1, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag, 2nd corrected printing 1997.

[7] K. Feldmann, C. Schnur and A. Colombo, "Modularised, distributed real-time control of flexible production cells, using Petri nets", Control Engineering Practice, Int. Journal of IFAC, pp. 1067–1078, 1996.

[8] W. Cheung, L.C. Leung and P.C.F. Tam, "An intelligent decision support system for service network planning", *Decision Support Systems*, vol. 39, pp. 415–428, May 2005.

[9] R. Engelen, "Code generation techniques for developing light-weight XML Web services for embedded devices", Proceedings of the 2004 ACM symposium on Applied Computing, ACM Press, New York, NY, USA, pp. 854–861, 2004.

[10] G. Y. Lin and J. J. Solberg, "An agent-based flexible routing manufacturing control simulation system", Proceedings of the 26th conference on Winter simulation, Society for Computer Simulation International, San Diego, CA, USA, pp. 970–977, 1994.

[11] B. Fleischmann, H. Meyr and M. Wagner, "Advanced Planning", in *Supply Chain Management and Advanced Planning*, 4th ed., H. Stadtler and C. Kilger, Springer Berlin Heidelberg, pp. 81–106, 2007.

[12] J. Barata, L. Ribeiro and M. Onori, "Diagnosis on Evolvable Production Systems", Proceedings of the IEEE International Symposium on Industrial Informatics, pp. 3221–3226, June 2007.

[13] J.M. Mendes, P. Leitão, A.W. Colombo and F. Restivo, "High-Level Petri Nets Control Modules for Service-Oriented Devices: A Case Study", Proceedings of the 34th Annual Conference of the IEEE Industrial Electronics Society, pp. 1487–1492, Nov. 2008.