Runtime Modeling of Flow for Dynamic Deadlock-free Scheduling in Service-oriented Factory Automation Systems

Corina Popescu, Maria de los Ángeles Cavia Soto[1] and Jose L. Martinez Lastra

Institute of Production Engineering, Tampere University of Technology, Korkeakoulunkatu 6, 33101 Tampere.
[1]E.T.S. Ingenieros Industriales y de Telecomunicación, Avda. de los Castros s/n 39005 Santander, Spain.

Abstract

Changes in equipment and production demand cannot be predicted at the design stage. Therefore, decision taking mechanisms must rely on real time information collected from the shop floor. To perform scheduling and routing optimization, not only modifications in values of parameters of interest, but also in the flow itself must be accounted for. This paper addresses this problem and proposes a method to formally model, at runtime, the flow within a service-oriented manufacturing line. The resulting representation assists deadlock- free dynamic scheduling of the system.

Keywords

Deadlock handling, Factory automation, Manufacturing systems, Petri nets, Runtime modeling, Scheduling, Service oriented architecture.

1. Introduction

Frequent production demand changes are reflected in corresponding modifications of a manufacturing line. The required adjustments range from PLC-level program changes to machine/robot replacements and sometimes even reorganization of the entire line. The constant increase in time-to-market pressure imposes an additional critical constraint on the feasible duration of these modifications.

The bridging of production engineering with other domains has been recognized to have a huge potential for addressing these problems [1,2]. In particular, Service Oriented Architecture (SOA), as a philosophy, and Web Services (WS), as a technology to support it, provide the necessary solutions.

Services are encapsulations of processes and can be thought of as interfaces. A service provides a clear separation between the way the encapsulated process is executed and the view other entities have of the process from the outside. Services are loosely coupled and can be (de)composed to whichever level of granularity may be required. Moreover, if annotated semantically, a service may be automatically discovered, invoked and composed.

From a SOA perspective, a manufacturing line is seen as a set of service encapsulations of provided and requested processes. The provided processes are the equipment skills. The requested processes are the product needs. Each product can be described in terms of its orchestrator. The orchestrator specifies the order of execution (the flow) of its needs – the services that should operate upon the raw material to obtain a final product. Following the SOA pattern [Figure 1], pallets (service requestors) search and locate the needed services in the order specified by their corresponding orchestrators. The devices (the service providers) publish the processes that they can offer. Selections of each device to execute upon a pallet are made gradually, as the orchestrator executes it task. Each time a device is selected for execution, the transportation services needed to carry the pallet to its chosen destination are subjected to discovery and selection as well. These steps take place for each service specified in the orchestrator of a pallet, until all product needs are satisfied and the pallet exits the line.

Service-oriented manufacturing systems allow both changes in the values of parameters of interest (online

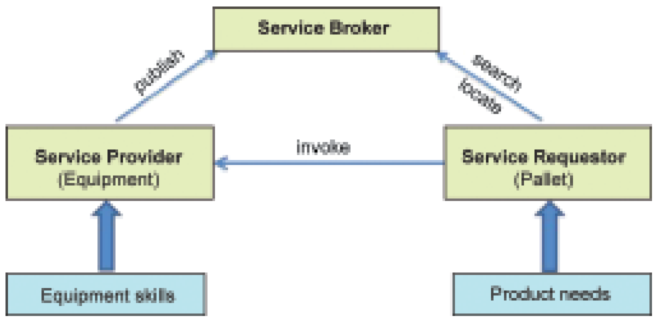


Figure 1: The service oriented architecture pattern.

equipment modifications) and the flow itself (variations in product type demand order) to be recognized and to be responded to in a natural way. However, optimal support of re-configurability and adaptability through WS technology is ensured only if dynamic decision taking mechanisms rely on formal flow representations that are obtained at runtime.

This paper extends previous results on automatic representation of formal models of standalone orchestrators [3]. The contents are organized as follows: Section 2 presents the background of this work. In particular, a brief overview of the state of the art in dynamic scheduling is presented, to illustrate the importance of making use of real time information in such decision taking mechanisms. The review is followed by an introduction to the syntax and semantics of the Petri Net–derived formalism that is used for modeling here [4,5]. The section concludes with a summary of the earlier reported research [3] and a comparison of the presented approach with related efforts. Section 3 gives details on the runtime modeling of flow. Section 4 discusses the application of traditional PN-based scheduling search procedures on the reachability graph of the dynamically constructed model. Section 5 presents the conclusions and the scope for future research.

2. Background

2.1 Dynamic Scheduling – An Overview

A *production schedule* is a specification, for each resource required for production, of the planned start time and end time of each job assigned to that resource.

Scheduling is the process of creating a production schedule for a given set of jobs and resources, while optimizing some performance measure (increase of productivity, minimization of operation costs, etc.). Based on production schedules, resource conflicts can be identified and the release of jobs to the shop can be controlled, for a better overall coordination of the activities in the manufacturing line.

Rescheduling is the process of updating an existing production schedule in response to disruptions such as machine failures and repairs, urgent job arrival, job cancelation, due date change or change in job priority.

Three main types of *rescheduling strategies* have been identified in the literature: completely reactive scheduling, predictive-reactive scheduling and robust pro-active scheduling:

Completely reactive rescheduling methods do not generate firm schedules in advance, but use dispatching rules to assist real time execution. (Panwalkar and Iskander [6] have provided an extensive list and classification of such rules. A comparative study in this field is provided by Rajendran and Holthaus [7]). The main problem associated with these techniques is the difficulty in predicting system performance, because the decisions are taken locally.

Predictive/Reactive scheduling is an iterative process of repairing previously-created schedules [8,9] or completely regenerating schedules [10]. Depending on the implemented *rescheduling policy*, the revisions may be triggered in response to unexpected events, altering the system status (event-driven) periodically, or in a hybrid manner.

Robust pro-active scheduling refers to the construction of predictive schedules that satisfy performance requirements predictably in a dynamic environment.

Several researchers have discussed the existing gap between scheduling theory and scheduling practice [11,12,13,14,15]. Only a small percentage of factories use scheduling tools or theories [13], because scheduling models and algorithms fail to consider the dynamic characteristics of a manufacturing system. As stated by Cowling and Johansson [14], 'scheduling research has failed to keep pace with technological developments in process control and monitoring systems.' Real time data is monitored and processed for control purposes; however, it is insufficiently used to improve schedules dynamically. This real time information should not only account for changes in the values of parameters of interest (online equipment modifications), but also in the flow itself (variations in product type demand order), and should be used by the dynamic scheduling system as it arrives.

A wide variety of dynamic scheduling techniques have been discussed in the literature [12,16,17,18].

Heuristics are schedule repair methods that target the finding of reasonably good solutions in a short time. Heuristic dispatching rules are defined based on experience and are assessed through simulation, with respect to various performance criteria (e.g. tardiness, flow time etc.). The choice of policies is problem specific, and no rule performs well for all performance criteria [8]. Dispatching rules are used extensively in multi-agent based dynamic scheduling [19,20]. Multi-agent architectures address the drawbacks of central and hierarchical scheduling through a network of individual problem solvers that cooperate. The system behavior is influenced by concurrent independent local decisions. In this manner, system complexity and cost are reduced and flexibility and fault tolerance are increased.

Mathematical programming techniques ignore practical constraints such as material handling capacity and complex resource sharing/routing, and, therefore, have only a few real applications in industry [16,17].

Meta-heuristics seek to avoid entrapment in poor local optimums obtained through local neighborhood search methods. The most popular meta-heuristic techniques include tabu search, simulated annealing and genetic algorithms [18].

Knowledge based systems, genetic algorithms (e.g. [9]), fuzzy logic, case-based reasoning and neural networks have also been considered as potential solutions to the scheduling problem. However, most of these techniques have been found to entail considerable computational effort.

Petri Nets can finely describe shared resources, synchronization, lot sizes and routing flexibility [21,22]. PN-based scheduling implies a search in the state space of the system for a sequence of feasible transition firings from an initial state to a goal state. The found schedule is deadlock free (one of the main advantage of Petri Nets over the other discussed dynamic scheduling techniques). Additionally, it is event-driven, which makes this type of scheduling perfectly suitable for real time implementation.

2.2 Timed Net Condition Event Systems: Syntax and Semantics

This work uses a Petri Net (PN) [23] derived formalism called Timed Net Condition Event Systems (TNCES) [4,5].

TNCES enhances the expression capabilities of Petri Nets with typed modularity, and adds to the originally defined elements of a PN the notions of event arcs and condition arcs. Event arcs report changes in the state of the system, while condition arcs carry state information. TNCES can model simultaneous start, has a clear notion of interfaces (event inputs/outputs and condition inputs/outputs) and a modular hierarchy.

An example of a simple TNCES module of name and type 'Atomic Service' is depicted in Figure 2. Apart from sets of places ({p1, p2}), transitions ({t1,t2}) and flowarcs ({(p1,t1), (t1,p2), (p2,t2), (t2, p1)}), which are present in any PN, this TNCES module has event inputs ({*start*}), event outputs ({*running*, *end*}) and condition outputs (={*s_available*, *s_not_available*}). Event arcs ({(*start*, t1), (t1, *running*), (t2, *end*)}) link event inputs to transitions / transitions to event outputs. Condition arcs ({(p1, *s_available*), (p2, *s_not_available*)}) link places to condition outputs. Condition arcs may also link condition inputs to places, however this is not illustrated in Figure 2.

Figure 3 illustrates a slightly more complex example of a TNCES module (of name 'orchestrator') composed of three internal modules (of name 'Sequence' and type 'sequence', and of name 'A'/'B' and type 'Atomic Service'). The internal modules are interconnected by means of module event arcs ({(Sequence.*start_s_1*, A.*start*), (Sequence.*start_s_2*, B.*start*), (A.*end*, Sequence.*end_s_1*), (B.*end*, Sequence.*end_s_2*)}) and module condition arcs ({(A.*A_av*, Sequence.*s1_available*), (B.*B_av*, Sequence.*s2_available*)}).

Condition and event arcs influence the firing rules in a TNCES module. A transition that is marking enabled (i.e. has at least one token in each of its input places) may fire *at any point in time* in case it is also condition enabled. Considering the example in Figure 3: Transition Sequence. t2 may fire at any point in time if there is at least one token in the place Sequence. p2 (i.e. the transition is marking enabled) and if there is one token in the place A.p1 (i.e. the transition is condition enabled through the module condition arc (A.A_av, Sequence.s1_available)). A transition that is marking enabled will fire immediately in case it is also event enabled. In the module depicted in Figure 3, transition Sequence.t3 fires immediately if there is at least one token in place Sequence. p3 and once transition A.t2 fires (change in state signaled through the module event arc (A.end, Sequence.end_s_1)).

TNCES modules may be associated delay times with flowarcs outgoing from places.

The extensions provided by TNCES have a fully defined mathematical backbone [4]. The notions of condition arcs and event arcs, together with their impact on the flow of tokens within the net can be derived algebraically. Therefore, extending the modeling power by using TNCES instead of an ordinary PN fully preserves the capability to perform deadlock-free scheduling and verification.

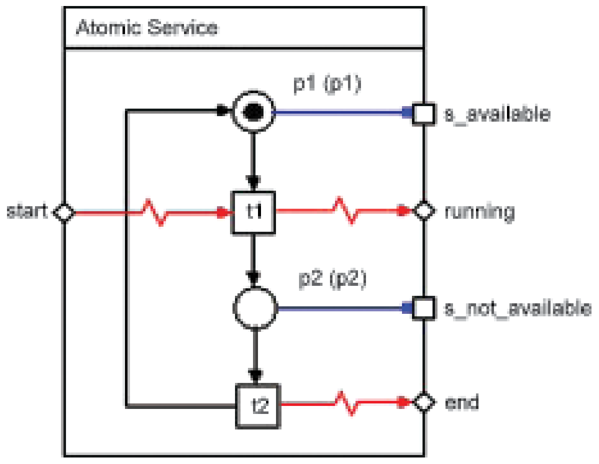


Figure 2: TNCES representation of an atomic service.

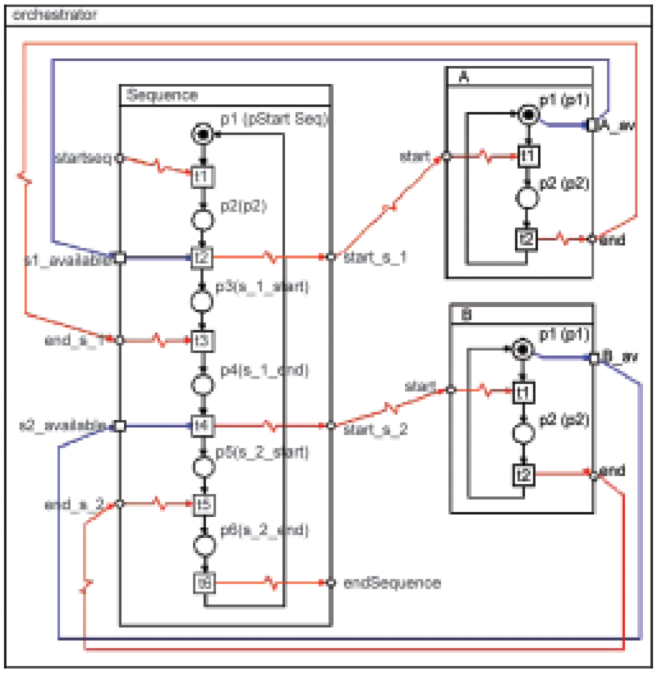


Figure 3: TNCES module. Sequence of two atomic services.

2.3 Previous Work and Related Efforts

The modeling of service orchestration is approached in a modular manner. A library of typed and composable TNCES modules has been defined to represent an atomic service and eight flow descriptors capable of expressing multithreading and synchronization (*Split*, *Split+Join*), looping (*RepeatUntil* and *RepeatWhile*), sequencing (*Sequence*, *AnyOrder*) and choice (*Choice*, *If-Then-Else*). Each of these formal models can be generated automatically as a function of the number of participants (a tool for this purpose – a plug-in to an earlier developed set of tools - is available online at http://www.pe.tut.fi/movida3/tools/). The generation procedures are detailed in [3].

The clearly defined interfaces facilitate the easy interconnection between the modules. Formal representations of any kind of flow can be created automatically in this way.

The earlier discussed example of Figure 3 illustrates the formal representation of a Sequence TNCES module engaging two participant atomic services A and B. Transition t1 of the Sequence module is marking enabled and will fire immediately upon receiving a change in state notification through the event input Sequence.*startSeq*. The internal functionality of the atomic services A and B is not included in the model. The only information concerning the atomic services that is available to the other TNCES modules is related to their busy/idle status. The details on how the processes are executed are left aside.

A comparison of this approach to related work described in the literature should highlight the following points: Unlike the models of Narayanan and McIlraith [24], this approach is modular and hierarchical. Luo *et al.* [25] rely on expansion of transitions for composition; this approach achieves the goal through clearly defined interfaces carrying information about states and changes in states. The set of flow descriptors is not defined by the authors (see [26]), but taken from the accepted OWL-S W3C Note [27]. The works on Business Process Execution Language [28] representations to PNs [29,30] are focused on a lower level, that of the peer to peer interactions that take place between services. This work aims to dynamically capture only the flow of services within the line, for deadlock-free scheduling purposes; therefore, it abstracts from these interactions. The work of Van der Aalst and colleagues ([31,32,33,34] is focused on modeling workflow patterns in general, and achieves composition of these patterns by fusing input and output places. Recently the group has directed their efforts at creating a complete specification of the formal semantics and analysis of control flow in WS-BPEL [34] – again, the modeling is taken to peer to peer interaction level.

3. Runtime Modeling of Flow

The flow sequence specified by a pallet's orchestrator is automatically translatable to TNCES [3]. Each time a pallet is introduced in the line, its standalone orchestrator formal model must be generated and combined with the existing overall flow model into a final *orchestrator mix model*.

A simple example is used here to clarify the dynamic generation procedure for the orchestrator mix model. Consider the situation illustrated in Figure 4 (left side):

A pallet with required sequencing described by orchestrator O1 [Figure 4, top side] is first introduced in the line. O1 is a representation of a higher level of granularity of the orchestrator depicted in Figure 4 (ordered sequence of two atomic services S1 and S2). The inner elements of each internal module are abstracted from for simplicity. Upon entering the line, the TNCES model of O1 is automatically generated. As there is only one pallet in the line, this model is a full formal representation of the current *orchestrator mix.*

Another pallet, characterized by O2 [Figure 4, bottom side], follows the first one after some time. O2 is a sequence of three atomic services: S1 (the same service searched by O1 initially), S3 and S4. The newly generated formal model (O2) must be added automatically to the already existing *orchestrator mix* model:

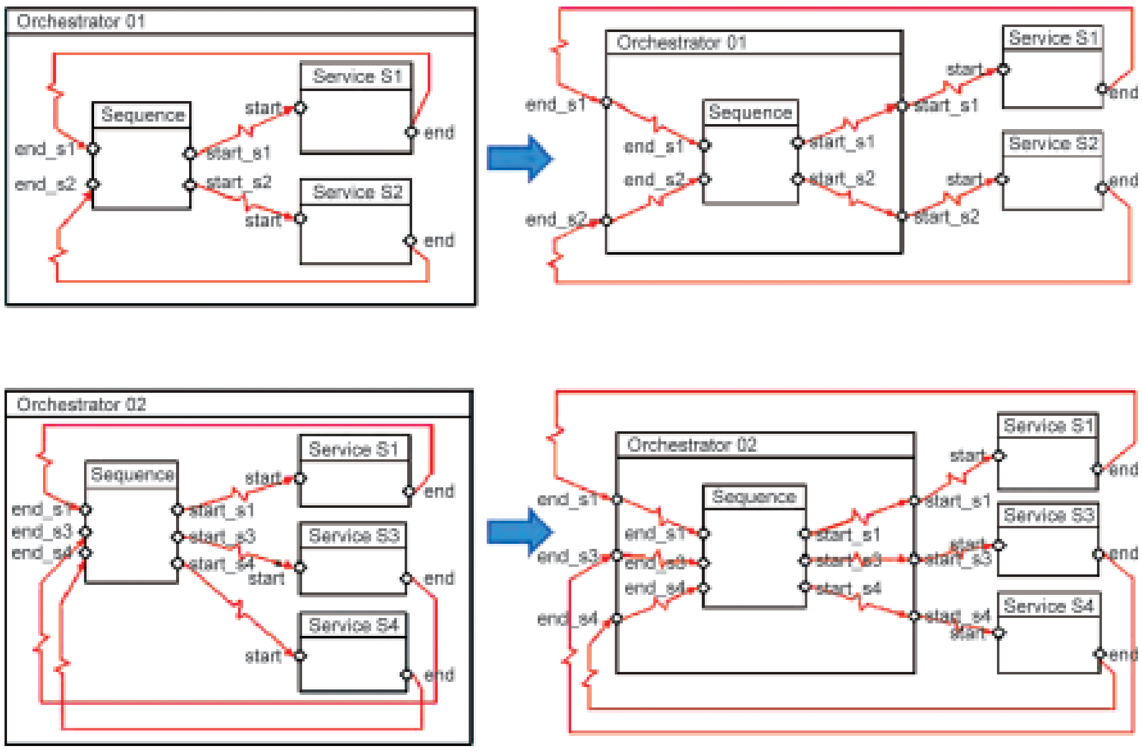


Figure 4: Automatically generated models of standalone orchestrators. Separation of flow-related representation from the atomic services representation.

For each standalone orchestrator, the atomic services are first separated from the rest of the model. An example of the outcome of this procedure is illustrated in Figure 4 (right side). The top-left side of the figure depicts orchestrator O1 as generated automatically in the initial stage. After separation (top-right), orchestrator O1 consists of only the Sequence module. Additional event input and output sets are created to enable the necessary links to the external modules Service S1 and Service S2. To ensure a correct separation, the order in which the atomic services are withdrawn from the initial orchestration module is tracked. This order must be consistent with the generation order of corresponding additional event input –event output pairs.

The TNCES orchestrator mix model obtained by adding the model of O2 to the original *orchestrator mix* model (i.e., O1) is illustrated in Figure 5. The generation procedure considers the orchestrator model to be added at this step [Figure 4, bottom-right side]. A check is performed to see whether the atomic services of the newly introduced model are already part of the existing mix model or not. In this case, service S1 is found to already have been included in the *orchestrator mix* model. Therefore, only the necessary connections are added to the mix module (i.e., the event arcs connecting O2.*start_s1* to S1.*start* and S1.*end* to O2.*end_s1*). The other two atomic services involved in the formal model of O2 - S3 and S4 - are not yet part of the current mix model. Therefore, the corresponding TNCES modules and the necessary connections are added to the main module.

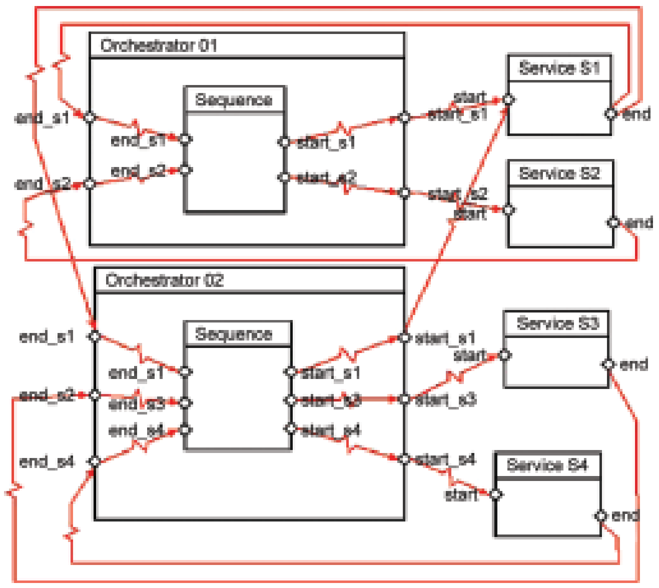
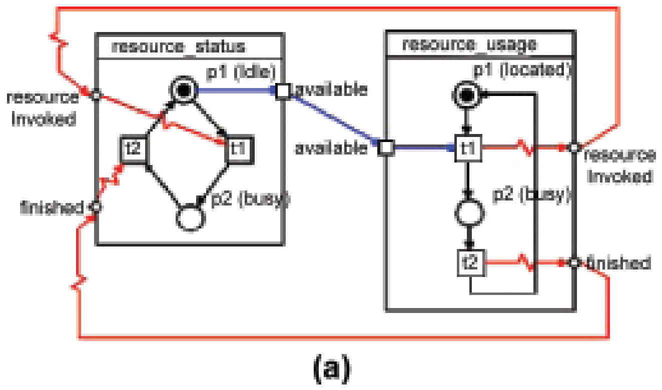


Figure 5: Formal model of the orchestrator mix of the two standalone orchestrators depicted in Figure 4.

The mapping of services to resources is also done at runtime, to account for online equipment modifications or additions. The formal model of the orchestrator mix is dynamically combined and updated with TNCES modules describing the status and, separately, the usage of each existing resource. The separation between status and usage is done to ensure that the possibility to represent the cases in which the same resource is located by more than one requestor (either to provide the same or different services).

A *resource_status* module [Figure 6a, left side] keeps track of whether the depicted resource is busy or idle. A *resource_usage* module [Figure 6b, right side] is a representation of whether the depicted resource is located or invoked by a requestor [Figure 1]. There can be exactly one TNCES *resource_status* module per resource in the mix model. The orchestrator mix model may include as many *resource_usage* modules as necessary, for each device in the line (to correspond to the number of times the device is actually identified as potential provider for a requested service). All modules of both types are to be dynamically updated with real time information from



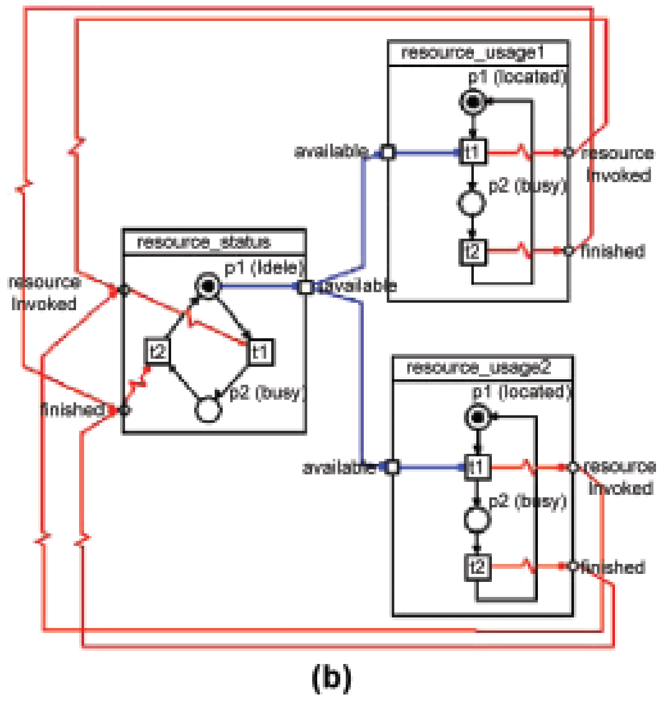


Figure 6: Combining TNCES models of resource status and usage: (a) TNCES model of a resource that is located once; (b) TNCES model of a resource searched for by two different requestors.

the shop floor. This information includes, for instance, various time delays that are associated with the same device performing different processes.

Figure 6a illustrates the model of a resource that is located and possibly invoked once. In case the resource is idle (i.e., there is one token in place resource_status.p1; m(*resource_status.p1*)=1) and identified as potential provider of service for a particular requestor/pallet (m(*resource_usage.p1*)=1), transition *resource_usage.t1* may fire at any time. The information regarding the (un) availability of the resource is carried through the condition arc that links place *resource_status.p1* to transition *resource_usage.t1*. In case *resource_usage.t1* fires, a token is placed in place *resource_usage.p2*. At the same time, the firing of *resource_usage.t1* is announced through the module event arc connecting the *resource_invoked* event output and input. The triggering of this event will cause the firing of transition *resource_status.t1*. Consequently, a token is placed in place *resource_status.p2* as well.

A device may be located more than once. The requests may be for the same service or not – the model accounts for the situation in which a resource has multiple skills in terms of processes (The incoming flowarc of each transition *resource_usage.t2* may be assigned a time interval to represent various processing durations).

Figure 6b illustrates the situation in which the same device is searched for by two different requestors. In this case, the two separate *resource_usage* modules initialized at *m(p1)=1* reflect the case in which two pallets have discovered this particular device to be capable of responding to their current demands. Resource invocation can take place only once. This point should be reflected when searching the reachability graph for scheduling purposes.

Figure 7 shows the formal model of the orchestrator mix depicted in Figure 5, in the case both orchestrators request Service S1. The needed skill may be provided by two different devices: machines R5 and R10. This information is collected in real time from the shop floor and input to the orchestrator mix model. For each considered device, a *resource_status* module is added to the flow model (*resource5_status* and *resource10_status*). Each potential mapping (O1(S1):R5; O1(S1):R10; O2(S1):R5; O2(S1):R10) results in the addition of a corresponding *resource_usage* typed module. The modification of the formal model is done at runtime, based on online collected knowledge about device skills.

4. Scheduling Based on the Runtime Constructed Models

The runtime constructed TNCES model of flow mix is input to search procedures to find feasible deadlock-free schedules. The net marking obtained after each update of

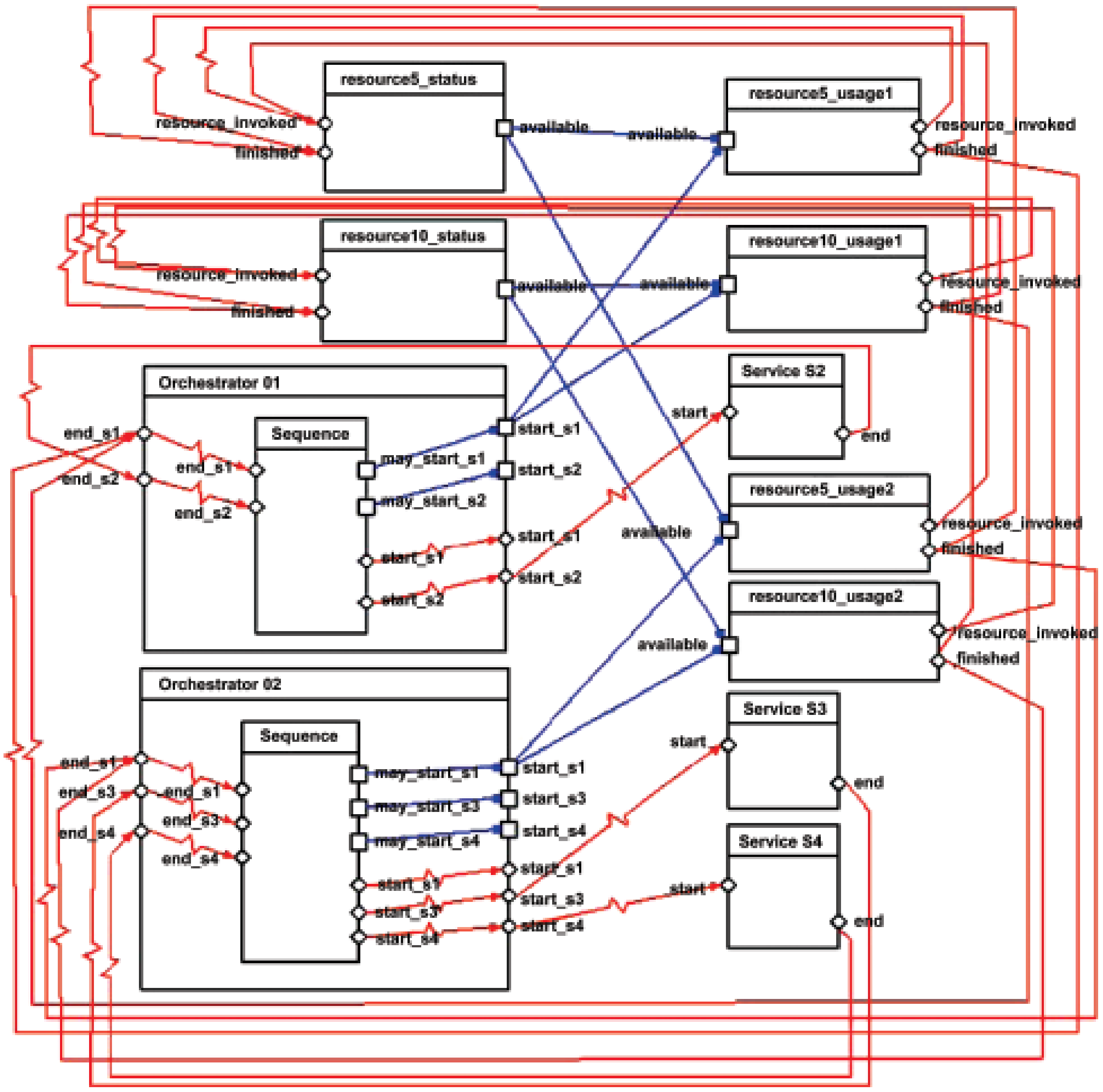


Figure 7: Linking the mix model to real time information collected from the shop floor: O1 and O2 request service S1, which may be provided by either resource 5 or by resource 10.

the entire model is the initial marking of the new search.

The update of the flow mix model and its parameters is event-driven. The entire model changes as a result of machine failure or machine addition. Model parameters are updated in case of machine replacement (e.g. time intervals associated with the inner flowarcs of the *resource_usage* typed modules change).

The goal marking is automatically obtainable, as the marking corresponding to termination of execution of each main calling module (in the example of Figure 7, such modules are the *Sequence* typed modules within each orchestrator).

Both backtracking (BT) and best-first (BF) search [21] can be used to find feasible schedules based on the runtime-constructed TNCES models of flow.

Backtracking maintains in storage a single path leading to the current marking, without considering optimality. Therefore there is no need to unnecessarily increase the size of the model input to the search procedure: for each service requested by each orchestrator, it is enough to incorporate only one of the found resource possibilities in the model. For backtracking purposes, in the model shown in Figure 7, it is enough to incorporate either one of the two resources available to perform Service S1 for orchestrator O1 (*resource5* or *resource10*); the orchestrator mix model is thus enhanced with either the *resource5_usage1* or the *resource10_usage1* module, and the corresponding connection arcs. The filtering of unnecessary paths in the state space of the complete flow model can thus be assisted at modeling stage already. Complementary resource allocation policies can be used during model construction to guide a feasible modeling phase.

Best-first search examines before each decision the entire set of available alternatives (both newly generated and suspended in the past). Therefore, if optimality is desired, all resource possibilities must be included in the overall flow model to be input to the search procedure (as illustrated in Figure 7). This imposes a set of rules to govern the on-the-fly construction of the BF search space:

First, a mutual exclusion policy must be followed if multiple resources are feasible to perform the same service on an orchestrator (multiple *resource_usage* typed modules are connected to the same *start_s_j* condition output of a calling module). In the example of Figure 7, service S1 needed by orchestrator O1 can be performed by either resource 5 or resource 10; in case both are available only the execution of one of the corresponding *resource_usage* typed modules must be investigated per search path (mutual exclusion condition-enabling imposed on the (O1.*start_s1*, *resource5_usage1.available*) and (O1.*start_s1*, *resource10_usage1.available*) condition arcs).

Second, if multiple *resource_usage* typed modules are connected to the same *resource_status* module, a mutual exclusion relation must be imposed on them (the same resource cannot be used for two different purposes at the same time). In the example of Figure 7, either one of the two condition arcs {(*resource5_status.available*, *resource5_usage1.available*), (*resource5_status.available*, *resource5_usage2. available*) must be allowed to condition-enable the firing of the corresponding transitions, when building the BF search space, per search path.

Module types dictate the type of internal transition firing, when building the search space. In *Split* or *Split+Join* modules, all transitions that are enabled at the same time should fire concurrently. In *Choice* typed modules, the same scenario imposes that only one of the eligible transitions fires.

The search for a schedule does not have to start all over again each time a new orchestrator is input to the line. The model update preserves the current situation of the orchestrators already involved in the mix model, while adding a new module.

5. Conclusion

Fast and optimal adaptation of production to changes in market demand and dynamic modifications of the manufacturing line is needed. Service encapsulation of devices has been repeatedly identified to respond to these needs.

However, online changes must reflect automatically in the formal model of the flow mix in a line, to input relevant information to the decision taking mechanisms immediately.

This paper proposes a modular and composable procedure to automatically build at runtime the flow model and to dynamically link this model to real time information collected from the shop floor. The resulting representation assists deadlock-free scheduling.

Many static PN flow modeling approaches exist in the literature. The main contribution of this particular work is the possibility to model flow at runtime, automatically, solely based on the standalone flow descriptions of each newly added pallet in the line. Traditional PN-based search procedures can subsequently be utilized for finding (non)optimal schedules taking into consideration the overall flow mix.

A tool has been programmed in JAVA as a plugin to an already existing tool chain (available online at http://www.pe.tut.fi/movida3/tools/). Currently the tool can be utilized for automatic generation of TNCES models of standalone flow descriptors, manual interconnection of these to generate orchestrator formal models, and for dynamic construction of the overall flow model each time a new orchestrator is added (to search for feasible schedules). Future work will focus on linking the existing tool set to the real time information coming from an industrial demonstrator. Pallet routing optimization based on the generated formal model of the flow mix is another upcoming topic of interest.

Acknowledgment

This work is partly supported by the Nokia Foundation Corporation.

References

1. J.L. Martinez Lastra, and I.M. Delamer, “Semantic Web Services in Factory Automation: Fundamental Insights and Research Roadmap”, *IEEE Transactions on Industrial Informatics*, vol. 2, no. 1., pp. 1-11, Feb. 2006.
2. I.M. Delamer, and J.L. Martinez Lastra, “Loosely-coupled Automation Systems using Device-level SOA”, in *Proceedings of the 5th IEEE International Conference on Industrial Informatics*, 2007, June 2007, pp. 743-48.

3. C. Popescu, A. Lobov, J.L. Martinez Lastra, and M. Cavia Soto, "A modeling approach to formally represent service orchestration", *International Journal of Computer Aided Engineering and technology*, vol. 1, no. 1, 2008, pp 1-30.
4. M. Rausch, and H.-M. Hanisch, "Net condition/event systems with multiple condition output", in *Proceedings of the Symposium on Emerging Technologies and Factory Automation*, vol. 1, Oct. 1995, pp. 592-600.
5. H.-M. Hanisch, J. Thieme, A. Luder, and A. Wienhold, "Modeling of PLC Behavior by Means of Timed Net Condition/Event Systems", in *Proceedings of 6th International Conference on Emerging Technologies and Factory Automation*, Sep. 1997, pp. 391-6.
6. S.S. Panwalkar, and W. Iskander, "A survey of scheduling rules", *Operations Research*, vol. 25, no. 1, Jan.-Feb. 1977, pp. 45-61.
7. C. Rajendran, and O. Holthaus, "A comparative study of dispatching rules in dynamic flow shops and job shops", *European Journal of Operational Research*, vol. 116(1), 1999, pp. 156-70.
8. R.J. Abumaizar, and J.A. Svetska, "Rescheduling job shops under random disruptions", *International Journal of Production Research*, vol. 35(7), 1997, pp. 2065-82.
9. A.K. Jain, and H.A. ElMaraghy, "Production scheduling/ rescheduling in flexible manufacturing", *International Journal of Production Research*, vol. 35, no. 1, 1997, pp. 281-309.
10. L.K. Church, and R. Uszoy, "Analysis of periodic and event-driven rescheduling policies in dynamic shops", *International Journal of Computer Integrated Manufacturing*, vol. 5(3), 1992, pp. 153-63.
11. B.L. MacCarthy, and J. Liu, "Addressing the gap in scheduling research: A review of optimization and heuristic methods in production scheduling", *International Journal of Production Research*, vol. 31(1), 1993, pp. 59-79.
12. C.S. Shukla, and F.F. Chen, "The state of the art in intelligent real-time FMS control: A comprehensive survey", *Journal of Intelligent Manufacturing*, vol. 7, 1996, pp. 441-55.
13. K. McKay, and V.C.S. Wiers, "Unifying the theory and practice of production scheduling", *Journal of Manufacturing Systems*, vol. 18(4), 1999, pp. 241-54.
14. P. Cowling, and M. Johansson, "Using real time information for effective dynamic scheduling", *European Journal of Operational Research* vol. 139, 2002, pp. 230-44.
15. G.E. Vieira, J.W. Herrman, and E. Lin, "Rescheduling manufacturing systems: A framework of strategies, policies and methods", *Journal of Scheduling*, 2003, vol. 6, pp. 39-62.
16. M. Zhou, Petri Nets in Flexible and Agile Automation. *Kluwer Academic Publishers*, 1995.
17. M. Zhou, and K. Venkatesh: Modeling, Simulation and Control of Flexible Manufacturing Systems – A Petri Net Approach, *World Scientific Publishing*, 1999.
18. D. Ouelhadj, and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems", *Journal of Scheduling*, 2008.
19. J.H. Lee, and C.O. Kim, "Multi-agent systems applications in manufacturing systems and supply chain management: A review paper", *International Journal of Production Research*, vol. 46(1), 2007, pp. 233-65.
20. C. Wang, H. Ghenniwa, and W. Shen, 'Real time distributed shop floor scheduling using an agent-based service-oriented architecture', *International Journal of Production Research*, vol. 46(9), 2008, pp. 2433-52.
21. D.Y. Lee, and F. DiCesare, "Scheduling Flexible Manufacturing Systems Using Petri Nets and Heuristic Search", *IEEE Transactions on Robotics and Automation*, vol. 10, no. 2, April 1994, pp. 123-32.
22. M. Zhou, and M.D. Jeng, "Modeling, Analysis, Simulation, Scheduling and Control of Semiconductor Manufacturing Systems: A Petri Net Approach", *IEEE Transactions on Semiconductor Manufacturing*, vol. 11, no. 3, August 1998, pp. 333-57.
23. T. Murata, "Petri nets: Properties, analysis and applications", *Proceedings of the IEEE*, vol. 77, no. 4, April 1989, pp. 541-80.
24. S. Narayanan, and S.A. McIlraith, "Simulation, Verification and Automated Composition of Web Services", in *Proceedings of WWW*, 2002, May 7-11, pp. 77-88.
25. N. Luo, J. Yan, and M. Liu, "Towards Efficient Verification for Process Composition of Semantic Web Services", in *Proceedings of Services Computing*, 2007, pp. 220-7.
26. R. Hamadi, and B. Benatallah, "A Petri Net-based Model for Web Service Composition", in *Proceedings of Australasian Database Conference*, 2003, vol. 17.
27. The OWL Services Coalition, 'OWL-S: Semantic Markup for Web Services', *Available from: http://www.ai.sri.com/daml/services/owl-s/1.2/overview, 2006.*
28. Web Services Business Process Execution Language. *Available from: http://docs.oasis-open.org/wsbpel/2.0/varprop*, 2007.
29. S. Hinz, K´. Schmidt, and C. Stahl, "Transforming BPEL to Petri Nets", in BPM 2005, LNCS 3649, 2005, pp. 220–5.
30. N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg, "Analyzing Interacting BPEL Processes", *Lecture Notes in Computer Science*, vol. 4102, 2006, pp. 17-32.
31. W.M.P. Van der Aalst, "Interorganisational workflows: An approach based on message sequence charts and Petri nets", in *Systems Analysis, Modelling, Simulation* 34(3), 1999.
32. The Workflow Patterns Initiative. Available from: http://www.workflowpatterns.com/., 2007
33. M. Voorhoeve, "Compositional Modelling and Verification of Workflow Processes", LNCS 1806, 2000, pp. 184-200.
34. C. Ouyang, E. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas, and A.H.M. ter Hofstede, "Formal semantics and analysis of control flow in WS-BPEL", in Science of Computer Programming, 2007.

AUTHORS

Corina Popescu received the Control Engineer degree from the Politehnica University of Bucharest, Romania, in 2004. She is currently pursuing the degree of Dr. Tech. at the Institute of Production Engineering, Tampere University of Technology. Her main research interests are in semantic web technologies and in formal methods in factory automation.

E-mail: corina.popescu@ieee.org

María de los Angeles Cavia Soto received her Licentiate degree in Physics and her Dr. Ing. degree in Electrical Engineering from the Universidad de Cantabria, Spain. Dr. Cavia has been a tenured University Reader since 1987, and is the Vice-Dean of the ETS de Ingenieros Industriales y de Telecomunicacion of the Universidad de Cantabria. Her main research interest is in the advancement of engineering education by developing new methodologies and tools.

E-mail: caviama@unican.es

Jose L. Martinez Lastra joined the Tampere University of Technology in 1997, and became Full Professor of Factory Automation in 2006. Prof. Lastra earned his advanced degrees (MS "with distinction" and Dr. Tech. "with commendation") in Automation Engineering from the Tampere University of Technology (Tampere, Finland). His undergraduate degree in Electrical Engineering is from the Universidad de Cantabria (Santander, Spain). Previous to his current position at the Department of Production Engineering, Prof. Lastra carried research at the Departamento de Ingeniería Eléctrica y Energética (Santander, Spain), the Mathematics Department (Tampere, Finland), the Institute of Hydraulics and Automation (Tampere, Finland) and the Mechatronics Research Laboratory of the Massachusetts Institute of Technology (Cambridge, MA). He has published over 175 original papers in international journals and conference proceedings and holds a number of patents in the field of Industrial Automation. His main research interest is in the application of information and communication technologies in the field of factory automation.

E-mail: lastra@ieee.org

DOI: 10.4103/0256-4602.50705; Paper No TR 63_09; Copyright © 2009 by the IETE