Scenario-based Modeling in Industrial Information Systems

Ricardo J. Machado[1], João M. Fernandes[2], João P. Barros[3], and Luís Gomes[4]

[1]Dep. Sist. Informação, Universidade do Minho, Portugal
[2]Dep. Informática / CCTC, Universidade do Minho, Portugal
[3]Instituto Politécnico de Beja / UNINOVA-CTS, Portugal
[4]Universidade Nova de Lisboa / UNINOVA-CTS, Portugal

Abstract. This manuscript addresses the creation of scenario-based models to reason about the behavior of existing industrial information systems. In our approach the system behavior is modeled in two steps that gradually introduce detail and formality. This manuscript addresses the first step, where text-based descriptions, in the form of structured rules, are used to specify how the system is or should be regulated. Those rules can be used to create behavioral snapshots, which are collections of scenario-based descriptions that represent different instances of the system behavior. Snapshots are specified in an intuitive and graphical notation that considers the elements from the problem domain and permit designers to discuss and validate the externally observable behavior, together with the domain experts. In the second step (not fully covered in this manuscript), the system behavior is formalized with an executable model. This formal model, which in our approach is specified using the Colored Petri Net (CP-nets) language, allows the system internal behavior to be animated, simulated, and optimized. The insights gained by experimenting with the formal model can be subsequently used for reengineering the existing system.

1 Introduction

In industrial environments, reengineering an existing industrial information system, to support significant changes in the process or to improve its performance, is usually an extremely sensitive operation. In industrial environments, modifying directly the system and testing the impact of those changes on the number and quality of the produced goods is simply prohibitive, because this would imply vast losses. Additionally, some industrial information systems are intrinsically complex, since they are expected to orchestrate control, data, and communication in distributed environments, where their operation is both business- and safety-critical. Monitoring and supervision of industrial processes require huge investments in technical solutions based on real-time embedded technologies, especially developed to interconnect the production equipments with the MIS (Management Information Systems) applications [8]. Complex systems are, by their nature, hard to master and reason about. In engineering, one classical solution to this problem is to create a model, since for the specific purpose in consideration, it is simpler, safer or cheaper than the considered system. For industrial information systems, which are typically control intensive [9], this implies that we essentially need to have a model of the behavior, since this is the most critical

view to take into account. This contrasts with data-centric systems, like databases or information systems, where the information and the relation among entities are the most important issues to consider.
For the majority of the existing industrial information systems in operation, there is no model with which one can immediately reason about those systems. If it does exist, typically the model does not completely reflect the system, since maintenance procedures that resulted in modifications in the system structure and behavior, were not reflected in changes in the model. This implies that techniques to obtain models for systems in use are most-needed in industrial organizations.
This manuscript presents an approach that was devised for a particular problem (i.e., an existing industrial information system), in order to obtain a behavioral model of that already existent system. This model, obtained after a careful description of the perceived behavior, permits industrial engineers (here, considered the domain experts) to reason about the system, evaluate which parts can be improved, change the model accordingly, analyze the improvements in relation to the initial version, and decide if the changes could be reflected in the industrial information system. In summary, the devised approach adopts three different artifacts:

1. Rules describe, in a textual form (written with natural language), how the system is (in an 'as-is' approach) or should be (in a 'to-be' approach) regulated, and thus implicitly specify the requirements the system is supposed to accomplish;
2. Snapshots present, in a pictorial format (by means of an intuitive and graphical notation), scenarios of the interactions among the system and the environment, illustrating application cases of the defined rules;
3. CP-nets are used to give a formal and executable nature to the snapshots, which are essential characteristics to allow reasoning capabilities.

Within a concrete reengineering problem of an existing industrial information system, the proposed approach supports the characterization of both the baseline situation (the 'as-is' system) and the future or end-state situation (the 'to-be' system). This is extremely important to allow the construction of the sequencing plan, where the strategy for changing the system from the current baseline to the target architecture is defined. It schedules multiple, concurrent, interdependent activities, and incremental builds that will evolve the industrial organization.
In this sense, the overall goal of the presented work is to simultaneously capture requirements and support animation of behavioral snapshots through Petri nets (PNs) based modeling. This manuscript focuses on the integrated usage of the first two artifacts for the considered industrial information system in an 'as-is' approach and is structured as next described. For details about the generation of CP-nets (from scenario models), please refer to [3, 11, 13] In Section 2, the running case study is briefly described. Section 3 presents the structuring of rules by using text-based descriptions. Section 4 illustrates the construction of snapshots by means of scenario-based descriptions. Section 5 briefly describes how CP-nets must be obtained to support reasoning activities. Section 6 is devoted to the final considerations.

2 Case study

All artifacts presented in this manuscript are related to the production lines that manufacture car radios (Fig. 1). Each car radio is placed on top of a palette,

whose track along the lines is automatically controlled. The transport system is composed of several rolling carpets that conduct the radios to the processing sites.

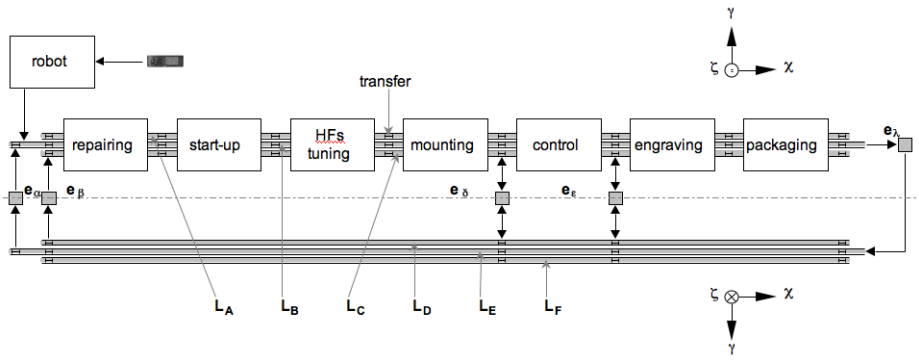


Fig. 1. The production lines of the case study.

The radios are processed in pipeline by the production lines. The processing sites are geographically distributed in a sequential way, along the production lines. Each production line is composed of 6 transport tracks (that can be simply called "lines"): three on the upper level (L_A, L_B, L_C) and three on the lower level (L_D, L_E, L_F). The upper level tracks transport palettes from left to right and the lower level tracks transport palettes from right to left.
The track L_B is used to transport radios between non sequential sites. The upper tracks L_A and L_C are preferably utilized for sending the radios to the buffers of the sites (FIFOs that start at the sites). The lower tracks are used for: (1) routing malfunctioning radios to the repairing sites; (2) feed backing the sites that did not accept radios because their buffers were full; (3) transporting empty palettes to the beginning of the line. There is also a robot that receives radios from the previous production sub processes (component insertion) and puts them on track L_B. The transfers allow the change of palettes between two neighbor tracks at the same level or between a track and an elevator. The five elevators (e_α, e_β, ...) establish the linkage between the upper and the lower tracks.

3 Text-based Descriptions

Text-based descriptions in the form of structured rules are used to specify how the system is or should be regulated. These rules constitute, from the external point of view, the functionalities of the control parts of the industrial information system.

The usage of rules at the beginning of the (re-)design phase is crucial to characterize the system, since domain experts can thus be involved to discuss, with the designers, the expected behavior for the environment elements (that constitute the plant). The option for natural language allows domain experts (frequently, persons with no scientific knowledge about specification formalisms) to effectively get involved in the definition of the rules.

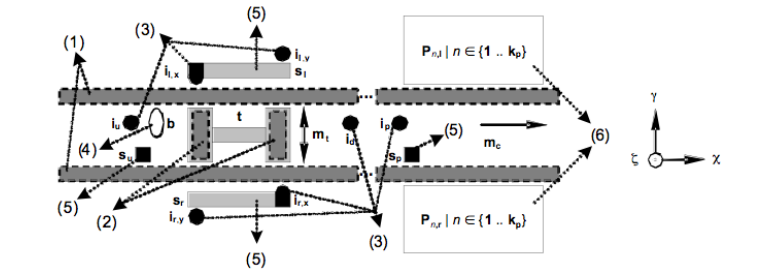
Typically, the rules make reference to the elements of the environment. Taking into account the domain concepts, it is crucial to normalize the vocabulary, the notation and the graphical elements. For the case study, the graphical notation depicted in Fig. 2 was adopted, where all the basic elements of the environment (in this case, sensors and actuators), that must be sensed and controlled by the system, possess a precise graphical representation and a textual notation.

Fig. 2.a shows (1) rolling carpets that transport the palettes along the O_χ axis, whose movement is activated by actuator m_c; (2) transfers that shift palettes between transport tracks along the O_γ axis, whose movement is activated by actuator t; (3) sensors that detect palettes in a specific (x,y) point of the transport tracks, identified as i_u, $i_{l,x}$, $i_{l,y}$, i_d, i_p, $i_{r,y}$, and $i_{r,x}$; (4) bar code readers that identify the car radio that is placed on top of a palette, identified as b; (5) stoppers that block the movement of palettes in a specific (x,y) point of the transport tracks, whose state is activated by actuators s_c, s_p, s_l, and s_r; (6) processing sites, identified as $P_{n,l}$ and $P_{n,r}$.

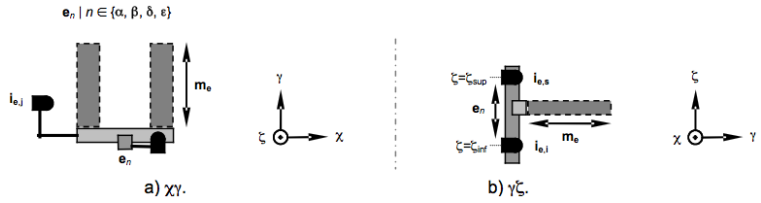
Additionally, for each basic element of the environment, there is a tabular description that fully characterizes its functionality and its logic interface (output for sensors and input for actuators). Fig. 2.c is an example of one of these tables for one inductive sensor. The tables for the other elements in Fig. 2.a are not shown here, due to space limitations. To specify the concrete production lines, this textual notation was used to instantiate each one of the existing elements of the environment package (Fig. 2). See [10] for details, not covered in this manuscript, on how to obtain the system's components.

The notation should take into account the elements usually adopted in the problem domain, so that designers can validate the behavior with the domain experts when animating the rules with behavioral snapshots. The effort to use only elements from the problem domain (in these rule-based representations) and to avoid any reference to elements of the solution domain (in what concerns the system parts) is not enough to obtain models that can be fully understood by common domain experts. This difficulty is especially noticeable in the comprehension of the dynamic properties of the system when interacting with the environment. This means that, even with the referred efforts, those static representations should not be used to directly base the validation of the elicited requirements by the domain experts. Instead, those static representations are used to derivate behavioral snapshots.

The purpose is not to formally reason about the mathematical properties of the obtained system models, in a typical verification approach. The usage of intuitive representations of the expected system behavior, from the external point of view and in a usability driven approach, is rather preferred. The adopted tables for static characterization and pictorial representation of the plant have proven to be quite effective to accomplish the goal of simultaneously capturing requirements and supporting the animation of behavioral snapshots.



(a)



(b)

Graphical notation	Textual notation	Type	State
● *vertical inductive sensor*	$i_{line,n,site}$ *line* ∈ {**A**, **B**, **C**} *n* ∈ [**0**, K_p**+1**[*site* ∈ {**u**, **d**, **p**}	(1) inductive sensor (2) without trigger (3) two state output (**ON** e **OFF**) (4) output normally **OFF** (5) located in the upper tracks	(1) **ON**, when a kit is detected (2) **OFF**, else
	$i_{line,n,site}$ *line* ∈ {**D**, **E**, **F**} *n* ∈ [**0**, K_p**+1**[*site* ∈ {**u**, **d**}	(1), (2), (3), (4) the same as $i_{line,n,site} \mid line \in \{A, B, C\} \wedge site \in \{u, d, p\}$ (5) located in the lower tracks	(1), (2) the same as $i_{line,n,site} \mid line \in \{A, B, C\} \wedge site \in \{u, d, p\}$
	$i_{line,n,site,dir}$ *line* ∈ {**A** .. **F**} *n* ∈ [**0**, K_p**+1**[*site* ∈ {**l**, **r**} *dir* = **y**	(1), (2), (3), (4) the same as $i_{line,n,site} \mid line \in \{A, B, C\} \wedge site \in \{u, d, p\}$	(1), (2) the same as $i_{line,n,site} \mid line \in \{A, B, C\} \wedge site \in \{u, d, p\}$
	$i_{line,n,site}$ *line* = **e** *n* = λ *site* ∈ {**B**, **E**}	(1), (2), (3), (4) the same as $i_{line,n,site} \mid line \in \{A, B, C\} \wedge site \in \{u, d, p\}$	(1), (2) the same as $i_{line,n,site} \mid line \in \{A, B, C\} \wedge site \in \{u, d, p\}$
	i_{site} *site* = **h**	(1), (2), (3), (4) the same as $i_{line,n,site} \mid line \in \{A, B, C\} \wedge site \in \{u, d, p\}$ (5) located at $\chi = \chi_{trt}$	(1) **ON**, when a kit is detected at $\chi = \chi_{trt}$ (2) **OFF**, else

(c)

Fig. 2. (a) Graphical notation of the case study environment; (b) Graphical notation of an elevator node; (c) Characterization of one basic element of the environment.

3.1 High-Level Rules

A set of generic rules (named high-level rules), that characterize the global objectives of the plant, must be defined. The concrete rules (just called rules) must contribute, either directly or indirectly, to the accomplishment of the high-level rules. For the running case study, the following high-level rule is an example:

> **[hlr 3]** Transfers and elevators must be managed as scarce resources of the environment. This implies that the time they are allocated to a given palette must be minimized and that the simultaneous accesses must be efficiently controlled.

This high-level rule of the plant is very generic and does not impose any design or implementation decision to the system. It also leaves open the way it will assure the exclusive access to the critical resources of the environment. However, although the high level rule is generic in its nature, it constitutes a proper requirement of the system, namely the need to control multiple accesses.

3.2 Rules

Due to the great complexity of the system (illustrated in the case study), it was decided to impose a functional partition that gave rise to two hierarchical levels to define the (low-level) rules: (1) level 1, where the strategic management decisions about the flows along the lines are considered; (2) level 2, where the concrete movement decisions for the palettes along the lines are taken. This 2-level partitioning guides the elicitation of the system requirements, since, for each level, a specific set of rules must be defined to specialize and refine the high-level rules.
For level 1, four sets of rules were defined: computation of the next production area (rna), site processing (rsp), buffers management (rbm), and strategic routing (rsr). In total, 15 rules of level 1 were characterized. As an example, consider one of the rules related to the site processing:

> **[rsp-2]** A car radio can be processed in a site, if the latter belongs to its processing sequence, if the task to be processed in the site was not yet accomplished over the car radio, if it is guaranteed that all the previous processing tasks were successfully executed over the car radio, and if the car radio physically arrived to the given site under coordination of the system.

For level 2, other four sets of rules were defined: transfers access (rta), elevators access (rea), fault tolerance (rft), and performance optimization (rpo). In total, 16 rules of level 2 were identified. As an example, consider one of the rules related to the elevators access:

> **[rea-2]** The routing of a palette that requires the usage of an elevator must be executed in two distinct steps; in the first one, the final destination is the transfer that is inside the elevator; in the second step, the destination is the real one and the start is the transfer inside the elevator.

This rule directly contributes to the fulfillment of high-level rule hlr-3. Nevertheless, not all high-level rules must be refined, since they are supposed to be very high-level directives to guide the development of the system. Thus, it is possible that some of them are not taken into account, especially in the early stages of system design, when functional prototyping gathers the main design effort. Typically, high-level directives that are concerned with non-functional requirements, such as fault tolerance and performance optimization, are postponed due to the need to adopt requirements prioritization techniques.

To fully characterize the interaction with the environment, all the possible rules must be elicited and documented. Thus, the system behavior is correctly and completely inferred. If this task is not properly executed, the behavioral description of the system can become incomplete and some inconsistencies may also occur.

4 Behavioral Snapshots

Scenarios are almost unanimously considered a powerful technique to capture the requirements of a given system. They are especially useful to describe the system requirements, which are typically more detailed than the user requirements. Additionally, scenarios are easier to discuss than the textual descriptions of the systems requirements, since these are inevitably interpreted in different ways by the various stakeholders, due to the usage of natural language.

UML 2.0 has several types of interaction diagrams: communication diagrams (designated collaboration diagrams in UML 1.x), sequence diagrams, interaction overview diagrams, and timing diagrams. Each type of diagram provides slightly different capabilities that make it more appropriate for certain situations. All interaction diagrams are useful for describing how a group of objects collaborate to accomplish some behavior in a given scenario. However, these diagrams are considered too technical for domain experts not able to read UML models.

In some situations, to allow a better communication with the domain experts, it is important to use a different notation, for modeling the interaction between the environment elements and the system. That notation should be based on the vocabulary of the problem domain. In the case study, the environment elements are sensors, actuators and the palettes for the car radios. If carefully selected to be as powerful and expressive as the sequence diagrams, the usage of behavioral snapshots is a proper choice, especially if the system is complex in behavioral terms and the need to discuss the system with the domain experts is paramount.

In our approach, an **instantaneous snapshot** is a static configuration of the environment elements in a sufficiently short timeframe, which assures the atomicity of the external observable system state from a behavioral point of view. A **behavioral snapshot** is a chronologically ordered collection of instantaneous snapshots that shows how elements of a system behave and react, within a given scenario. A **scenario** is a coherent sequence of actions that illustrates behaviors, starting from a well defined system configuration and in response to external stimulus. A behavioral snapshot is intended to convey the same behavior as a sequence diagram, and thus can be seen as a domain specific visual representation of a sequence diagram. Fig. 3 depicts one behavioral snapshot with four instant snapshots ($a \rightarrow b \rightarrow c \rightarrow d$), for the following rule of performance optimization:

[rpo 3] If a palette, during a movement through the transfers, is in a transfer of a middle line (lines B and E, for the upper and lower nodes), it must be verified, during a pre-defined period (parameter TIME_BL), if the exit at the destination is free; if this is not the case, the palette must follow for a middle line.

In this behavioral snapshot, between instants t_1 and t_2, palette #2 is put just after the transfer C, which makes impossible for palette #1 to reach its destination. The unexpected positioning of palette #2 just after the transfer C may occur without its explicit transportation by the system, since line operators sometimes put palettes in the tracks. At instant t_3, after time TIME_BL is elapsed, the destination for palette #1 is changed to track L_B, since palette #2 is still placed just after the transfer C. In this case, track L_B is used as an alternative route, since the initial destination (track L_C) can not be reached. With this strategy, the permanence of stopped car radios at the transfers is avoided, which increases the availability of resources. This behavior maximizes the probability of car radios to have a destination to exit the node, even in situations where the initial path becomes blocked for some reason. If the track L_B is also blocked, the node is blocked until the track becomes free. At the lower tracks, the behavior is similar and track L_E is used as the alternative one.

Only for those rules that present some critical behavior requirements it is recommended to construct the corresponding behavioral snapshots. Rule rpo-3 corresponds to a critical situation. The arrows depicted in behavioral snapshots represent the final destination of palettes. Whenever the destination of a palette must be redefined, a new arrow must be drawn to represent that new destination.

The behavioral snapshots can also illustrate the application of the rules that present alternative or optional scenarios. Rule rsr-7 presents two alternative behavioral snapshots.

[rsr-7] Under the request of level 2 control, level 1 control should authorize one palette to mount into one transfer, if the palette path does not present any crossing point with any other palette that is already executing its path along the same node and if the exit at the destination is free (the place just after the transfer) to receive the palette.

Fig. 4 depicts one behavioral snapshot for rule rsr-7. In this scenario, palette #2 has track L_B as its destination. At time t_1, it is possible to check that the path to track L_B is free, even though one palette (#3) is located in a transfer, while being conducted to its destination (track L_A). The movement of palette #2 can be started at time t_1, since the paths of the palettes #2, #3 and #4 do not overlap and the destination of palette #2 is free. Instants $t_2 - t_4$ show the elementary movements made simultaneously by palettes #2 and #3 to reach their destinations (palette #4 remains stopped during all the scenario).

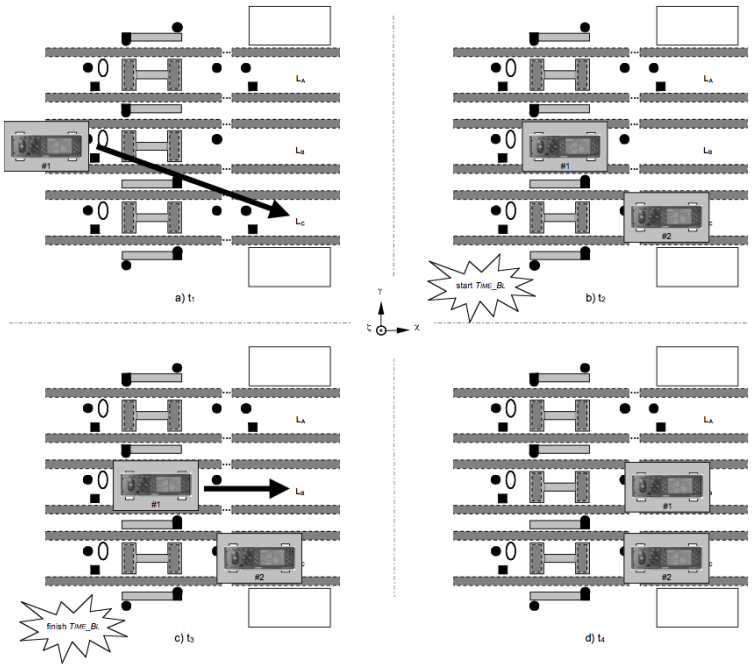


Fig. 3. A behavioral snapshot for rule rpo-3.

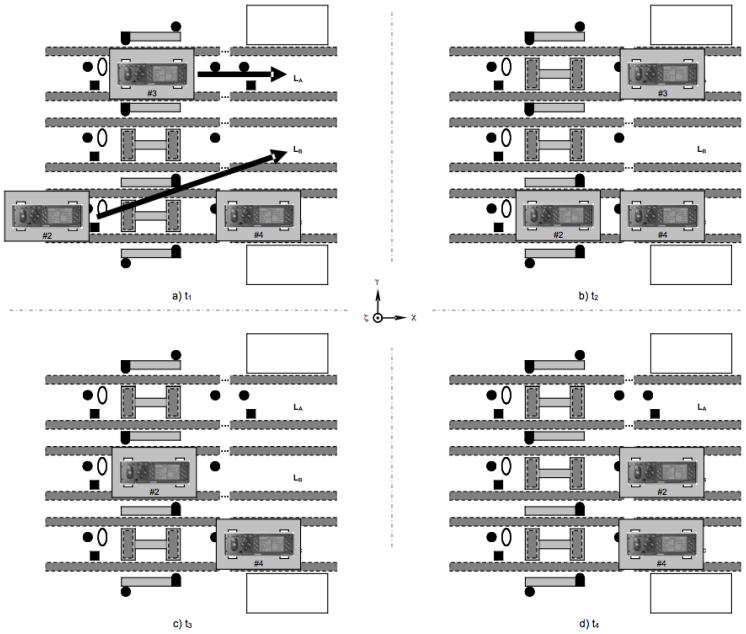


Fig. 4. First behavioral snapshot for rule rsr-7.

Behavioral snapshots are a good technique for requirements elicitation. However, since they are based on scenario identification, they do not assure a complete behavior characterization and they lack semantic formalization. These characteristics justify the usage of a more formal behavioral specification to support the system detailed design, namely those based on state oriented models.

5 Specification with Colored Petri Nets

As already said, the ultimate goal of the approach partially presented here is to allow the generation of CP-nets from scenario models, in order to allow validation of the system under consideration.
The application of PNs to the specification of the behavioral view of controllers can benefit from several research results. PNs constitute a mathematical meta-model that can be animated/simulated, formally analyzed, and for which several implementation techniques are available. The designer can choose, among several PN meta-models, a specific one intentionally created to deal with the particularities of the system under consideration, like the ones referred in [4, 7, 15, 16].
In the last years, research in scenario-based modeling is receiving a considerable attention. In this manuscript, the main general goal is to devise scenario-based modeling techniques that can be translatable to a PN model, so here we focus on previous works that address the (more generic) transformation of scenario-based models into state-based models.
Campos and Merseguer integrate performance modeling within software development process, based on the translation of almost all UML behavioral models into Generalized Stochastic PNs [1]. They explain how to obtain from sequence diagrams and statecharts a performance model representing an execution of the system.
Shatz and other colleagues propose a mapping from UML statecharts and collaboration diagrams into CP-nets [14, 5]. Firstly, statecharts are converted to flat state machines, which are next translated into Object PNs (OPNs). Collaboration diagrams are used to connect these OPN models and to derive a CP-net model for the considered system, which can be analysed by rigorous techniques or simulated to infer properties some of its behavioral properties.
Pettit and Gomaa describe how CP-nets can be integrated with object-oriented designs captured by UML communication diagrams [12]. Their method translates a UML software architecture design into a CP-net model, using pre-defined CP-net templates based on object behavioral roles.
Eichner at al. introduce a formal semantics for the majority of the concepts of UML 2.0 sequence diagrams by means of PNs [2]. The approach concentrates on capturing, simulating and visualizing behavior. An animation environment is reported to be under development, to allow the objects to be animated, using the PN as the main driver. Their work has some similarities with ours, namely on the usage of sequence diagrams, but uses a different PN language (M-nets) and is oriented towards sequence diagrams that describe the behavior of a set of objects.
It is important to note that the choice of which state based model to use must be made consciously, taking into account the characteristics of the system. If they have simple sequential behavior, FSMs or Statecharts are enough, but if they present several parallel activities and synchronization points, a high-level PN may be the most adequate choice to cope with the system's complexity.

In our approach, behavioral snapshots are translated into sequence diagrams to allow the application of the techniques described in [13, 11, 3] to allow the rigorous generation of CP-nets [6]. The transitions of these CP-nets present a strict one-to-one relationship with the messages in the sequence diagrams. So, for each message in a sequence diagram, one transition, in the corresponding CP-net, is created. In order to make that correspondence more evident, the name of each transition matches exactly the name of the corresponding message in the sequence diagram.

6 Conclusions and Future Work

In this manuscript, we present an approach that uses scenario-based descriptions and CP-net for modeling the behavior of an industrial information system. Further research is needed to investigate how the approach can be generalized, namely because the usage of an informal and intuitive notation, based on concepts and elements borrowed from the problem domain, may not have the same degree of readability.
A behavioral snapshot is an ordered collection of instant snapshots and shows how elements of a system behave and react, within a given scenario. Since the notation for the snapshots should consider the vocabulary of the problem domain, designers and domain experts can cooperate in the validation of the system behavior. The presented approach offers a client friendly scenario notation, which eases the discussion with non technical stakeholders.
Based on the sequence diagrams equivalent to the behavioral snapshots, controllers can be incrementally formalized with a state based model. CP-nets are adopted, since they are able to explicitly support the management of the environment resources in a conservative way. This incremental approach allows the completion and early correction of CP-nets by functional validation and performance optimization.
Currently, the domain concepts used in the snapshots have to be produced for each application. As a way to bridge the current gap between sequence diagrams and snapshots, the development of a domain specific meta model to describe the terms used on the sequence diagrams is under consideration.
It is also planned to incorporate into the tool workbench a mechanism to achieve the automatic generation of the animated sequence diagrams. This will allow the automatic reproduction of the very same set of scenarios that were initially described using behavioral snapshots and sequences, if the state based model is correct and complete.

References

1. J. Campos and J. Merseguer. On the integration of UML and Petri nets in software development. In *27th Int. Conf. on Applications and Theory of Petri Nets and Other Models of Concurrency (Petri Nets 2006)*, LNCS 4024, pages 19–36. Springer, 2006.
2. C. Eichner, H. Fleischhack, R. Meyer, U. Schrimpf, and C. Stehno. Compositional semantics for UML 2.0 sequence diagrams using Petri nets. In *Model Driven Systems Design (SDL 2005)*, LNCS 3530, pages 133–48. Springer, 2005.

3. J.M. Fernandes, S. Tjell, J.B. Jørgensen, and O. Ribeiro. Designing tool support for translating use cases and UML 2.0 sequence diagrams into a coloured Petri net. In *6th Int. Workshop on Scenarios and State Machines (SCESM 2007), at ICSE 2007*. IEEE CS Press, 2007.
4. L. Gomes and J. P. Barros. Structuring and composability issues in petri nets modeling. *IEEE Trans. Industrial Informatics*, 1(2):112–123, 2005.
5. Z. Hu and S. M. Shatz. Mapping UML diagrams to a Petri net notation for system simulation. In *Int. Conf. on Software Engineering and Knowledge Engineering (SEKE 2004)*, pages 213–9, 2004.
6. K. Jensen, L. M. Kristensen, and L. Wells. Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. *Int J. on Software Tools for Technology Transfer*, 9(3–4):213–254, 2007.
7. B. Kleinjohann, J. Tacken, and C. Tahedl. Towards a complete design method for embedded systems using predicate/transition-nets. In *Int. Conf. on Hardware Description Languages and Their Applications (CHDL '97)*, pages 4–23. Chapman & Hall, 1997.
8. R. J. Machado and J. M. Fernandes. Heterogeneous information systems integration: organizations and tools. In *4th Int. Conf. on Product Focused Software Process Improvement (PROFES 2002)*, LNCS 2559, pages 629–43. Springer-Verlag, 2002.
9. R. J. Machado and J. M. Fernandes. Integration of embedded software with corporate information systems. In *1st IFIP Int. Embedded Systems Symposium (IESS 2005)*, pages 169–78. Springer, 2005.
10. R. J. Machado and J. M. Fernandes, H. Rodrigues, and P. Monteiro. A demonstration case on the transformation of software architectures for mobile applications. In *5th IFIP TC10 Working Conf. on Distributed and Parallel Embedded Systems (DIPES 2006)*, pages 235–44. Springer, 2006.
11. R. J. Machado, K. B. Lassen, S. Oliveira, M. Couto, and P. Pinto. Requirements validation: execution of UML models with CPN Tools. *International Journal on Software Tools for Technology Transfer*, 9(3–4):353–369, 2007.
12. R. G. Pettit and H. Gomaa. Modeling behavioral design patterns of concurrent objects. In *28th Int. Conf. on Software Engineering (ICSE 2006)*, pages 202–11. ACM Press, 2006.
13. O. Ribeiro and J. M. Fernandes. Some Rules to Transform Sequence Diagrams into Coloured Petri Nets. In K. Jensen, editor, *7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN 2006)*, pages 237–256, 2006.
14. J. Saldhana and S. M. Shatz. UML diagrams to object Petri net models: an approach for modeling and analysis. In *Int. Conf. on Software Engineering and Knowledge Engineering (SEKE 2000)*, pages 103–10, 2000.
15. A. Semenov, A. M. Koelmans, L. Lloyd, and A. Yakovlev. Designing an asynchronous processor using Petri nets. *IEEE Micro*, 17(2):54–64, 1997.
16. M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli. Synthesis of embedded software using free-choice Petri nets. In *36th annual ACM/IEEE Design Automation Conf (DAC '99)*, pages 805–810. ACM, 1999.