2009 Fourth International Conference on Software Engineering Advances

Validation of Scenario-based Business Requirements with Coloured Petri Nets

Óscar R. Ribeiro João M. Fernandes
Dep. Informática / CCTC, Universidade do Minho, Braga, Portugal
{*orribeiro, jmf*}*@di.uminho.pt*

Abstract

A scenario can be used to describe a possible instantiation of a given business use case and can be expressed for example as a list of steps written in natural language, or by an interaction diagram. This paper discusses how a collection of scenarios, all expressed as UML2 sequence diagrams, can be described for validation purposes by a single model, written in the Coloured Petri Nets (CPN) modelling language. Due to the support for parallelism given by the CPN language, the obtained CPN model can: (1) simultaneously execute several scenarios; and (2) elegantly represent the parallel activities inside a scenario. This two-level parallelism is crucial during validation, since it allows one to detect problems that are only evident when several scenarios are in simultaneous execution and may affect each other. We exemplify our approach in a system that has a rich set of interactions with its users.

Keywords – *Business Requirements, Scenarios, Validation, Coloured Petri Nets*.

1. Introduction

Use cases are a well-known technique used to specify the set of functionalities presented by a system as seen by its users. They facilitate the dialogue between clients and developers, due to their simplicity and informal nature. Scenarios can be used to describe possible instantiations of a given use case. Each scenario describes a specific sequence of actions and interactions between the users and the system. Among other alternatives, scenarios can be expressed either as a list of steps written in natural language, or by a UML interaction diagram.

CPNs [8] are a graphical modelling language adequate to describe the behaviour of systems with characteristics like concurrency, resource sharing, and synchronization. The CPN modelling language is supported by *CPN Tools* [8] which is a tool that allows the execution of animations in accordance with the CPN model.

In this paper, we propose the description of the behaviour of each business use case to be detailed by a collection of UML2 sequence diagrams. Those sequence diagrams describe the interactions among the actors and the system. Each sequence diagram can describe more than one elementary scenario, because it can use high-level operators. To complement the initial steps of the analysis phase, it is critical to validate that the elicited requirements are indeed those needed by the stakeholders [12]. Our approach consists of expressing a set of sequence diagrams by a CPN model that can be used to animate and simulate, and consequently validate the behaviour of the system under consideration. Since the CPN modelling language naturally supports parallelism and concurrency, the obtained CPN model is able to: (1) execute in an interleaved way several scenarios (either instances of the same or different use cases); and (2) directly represent the parallel activities inside a given scenario. This parallelism at two different levels (intra-scenario and inter-scenario) is essential during validation, since it allows the analysts and the users to detect problems that emerge only when several scenarios that may affect each other are executed simultaneously. We exemplify our approach with a revised version of a system taken from [16].

This paper is structured as follows. Section 2 introduces the system used in this paper to exemplify our approach. Section 3 describes our approach to express a set of sequence diagrams by a CPN model. We describe, in Section 4, how the obtained CPN model could be used in the validation of requirements. Section 5 discusses related work. Conclusions and future work are presented in Section 6.

2. Check-in System

This section describes a check-in system in an international airport [16], which is used in this paper to exemplify the application of our approach.

The main stakeholder in this system is the so called "check-in agent" that interacts with the passenger in order to execute the check-in of the passenger. The sequence of steps performed by the check-in agent during the main scenario of the "check-in passenger" business use case is the following [16]:

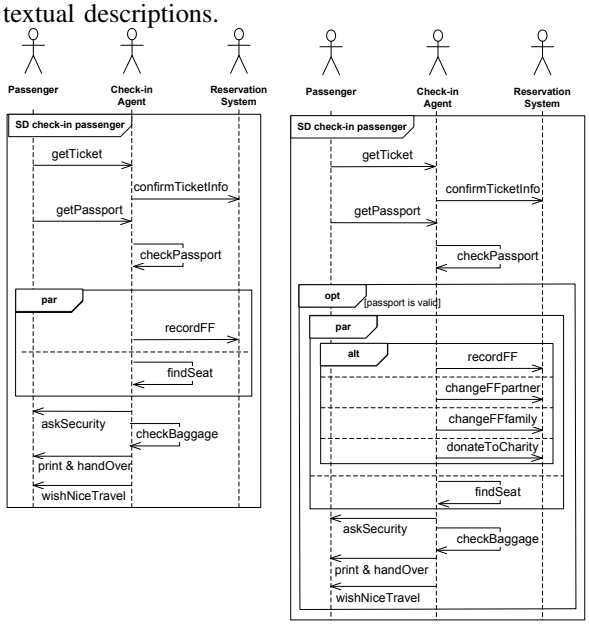
1) Get the passenger's ticket or record locator;
2) Confirm passenger, flight, and destination;
3) Check the passport is valid;
4) Record the frequent-flyer (FF) number;
5) Find a seat;

978-0-7695-3777-1/09 $26.00 © 2009 IEEE
DOI 10.1109/ICSEA.2009.45

6) Ask security questions;
7) Check the baggage onto the flight;
8) Print and hand over the boarding pass;
9) Wish the passenger a pleasant flight.

The main scenario is modelled by the UML2 sequence diagram illustrated in Fig. 1(a), where we can observe a parallel operator that represents the fact that steps 4 and 5 ("record the FF number" and "find a seat" respectively) can be accomplished in any order.

Each step in the scenario is represented by a message in the sequence diagram. The sender and the receiver of a message are extracted from the step's description, and the order between these messages is the same as the order introduced by the numbers of the corresponding steps in the textual descriptions.



(a) (b)

Figure 1. Sequence diagrams for the "check-in passenger" business use case: (a) the main scenario; (b) some alternatives and exceptions.

Some alternative and exception scenarios can be added to the main scenario. For example, three alternatives for step 4 were identified:

A4.1 Allow the FF number to be changed to that of a partner airline;

A4.2 Allow the FF number to be changed to that of a family member; or

A4.3 Allow the mileage of the flight to be donated to a charity of the passenger's choice.

The "A4" prefix in the previous items is used to indicate that the items constitute an alternative for the step 4 of the main scenario, and each alternative item is also enumerated.

In the case the passenger has an invalid passport, an exception to the main scenario is introduced and the execution of the scenario must be ended.

The behaviour present in the main scenario, with the alternatives listed above together with the exception for the invalid passport can be expressed by the sequence diagram in Fig. 1(b). This sequence diagram describes a set of elementary scenarios, since three high-level operators are used.

3. Expressing Scenarios by a CPN Model

To illustrate our approach we show how it can be applied to the "check-in passenger" business use case. This section explains the manual transformation of UML2 sequence diagrams, describing scenarios, into a CPN model. After explaining how to obtain the CPN model for the main scenario, we show how to add alternative and exception scenarios, and how to enrich the CPN model with some possible behaviours performed by the passengers. The interested reader is referred to [15], for details on how to obtain a CPN model from a set of sequence diagrams.

3.1. Expressing the Main Scenario

The main scenario of the use case described by the sequence diagram in Fig. 1(a) gives rise to the part of the CPN model in Fig. 2 that is inside the dashed line.

The initial state for the considered scenario is modelled by the tokens inside the two places at the top of Fig. 2. The place ready to check-in passengers contains the tokens that represent the passengers that are ready to proceed with the check-in, and the place available agents contains the tokens that represent the available check-in agents. These two places are used as input places to the transition getTicket, which represents the step in the scenario where the passenger gives the ticket to the agent. After this trigger, the passenger and the agent must proceed together to complete the steps of the scenario. There are places between transitions to save the information related to the considered passenger and agent along the scenario execution and to preserve the order between the transitions (according to the order given in the sequence diagram). The input arcs for getTicket have the inscriptions p and a that are variables representing a passenger and an agent, respectively.

The execution of each instance of the scenario terminates when the considered passenger is checked-out, which implies that the agent is available again to start a new check-in procedure. The place checked-out passengers at the bottom of Fig. 2 is used to contain the tokens of the passengers that were successfully checked-out. In Subsection 3.4 the details about the colour sets used in the CPN model are presented.

3.2. Adding Alternatives and Exceptions

This subsection details how alternatives and exceptions can be integrated in the CPN model obtained for the main scenario. As explained in Section 2, in the case study being considered in this paper there is one exception introduced

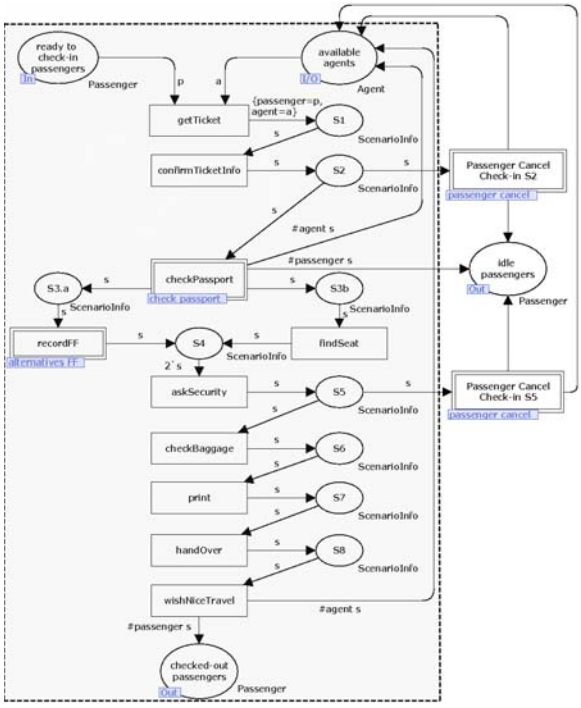


Figure 2. CPN module to express the "check-in passenger" business use case.

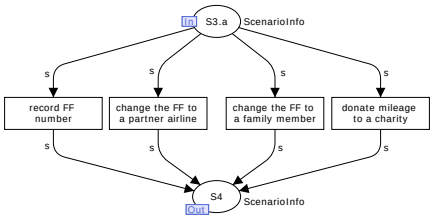


Figure 3. altenativesFF CPN module.

by the usage of an invalid passport; and there are two alternatives introduced by the possibility to either save the flight in a FF number of a partner airline, save the flight in a FF number of a family member or donate the mileage. The resulting behaviour is described by the sequence diagram in Fig. 1(b), where we can find *opt* and *alt* operators.

The transitions recordFF and checkPassport are substitution transitions that represent the behaviour introduced by these high-level operators. The substitution transition recordFF is connected to the CPN module presented in Fig. 3, where there is an alternative execution among the four presented transitions. In the common input place S3.a there is only one token for each scenario instance, which in this case is represented by a passenger and a check-in agent.

The substitution transition checkPassport is connected to the module in Fig. 4, where there are two alternative transitions: one for the case when the passport is valid, and another one when the passport is invalid. The use of an

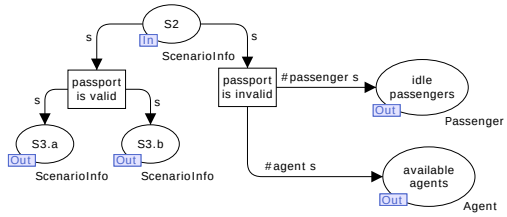


Figure 4. check passport CPN module.

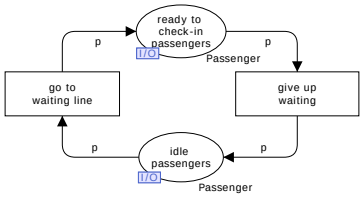


Figure 5. The CPN module to capture some behaviours of the passenger.

invalid passport is considered an exception, implying that the execution of the scenario must terminate. The end of the scenario execution moves the token representing the passenger participating in the scenario to the place idle passengers, and the token representing the agent to the place available agents.

3.3. Adding the Behaviour of Actors

We assume, as explained in [7], that passengers have free will and might behave in unexpected ways. Check-in agents also have free will, but they are more biddable, so they are expected to behave in a more constrained way. Therefore, in this paper we enrich our CPN models with alternative scenarios triggered by possible abnormal or unexpected behaviours of the passengers, which may include, for example, ignorance, insubordination or malevolence. This is an important issue to make the CPN models more useful for animation and validation purposes, since they address a richer set of scenarios of usage.

In the context of the "check-in passenger" business use case, three explicit states were identified for passengers. In the main scenario we have identified the state (to begin the scenario) when the passenger is ready to check-in, and another state to finish the scenario when the passenger is checked-out. These two states are represented in the obtained CPN module in fig. 2 by the places ready to check-in passengers (at the top of the figure) and checked-out passengers (at the bottom of the figure). When considering alternatives and exceptions to the main scenario a third state was identified to capture the fact that the passenger is idle. For example, when the agent verifies that the passenger's passport is invalid the check-in scenario ends and this passenger is considered to be idle. The CPN module presented in Fig. 2 has a place idle passengers to contain the tokens corresponding to passengers that are idle.

To capture some of the abnormal behaviours introduced during the execution of a scenario we can enrich the

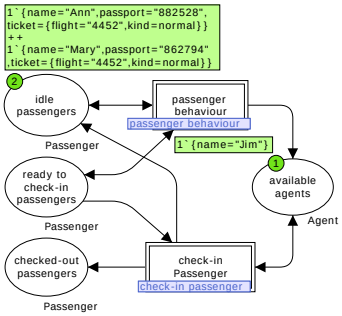


Figure 6. The top-most CPN module.

CPN model with a specific module. The CPN module that expresses the considered behaviours of the passenger is presented in Fig. 5. We consider that an idle passenger can go for the waiting line for proceeding with the check-in (transition go to waiting line in Fig. 5). A passenger in the waiting line can decide to abandon the area, becoming idle again (transition give up waiting in Fig. 5).

We model the cancellation of the check-in process by the passenger after the confirmation of the ticket information, or after the passenger being asked about the security issues. These two execution points are represented by the places S2 and S5 in Fig. 2. The substitution transitions "Passenger Cancel Check-in S2" and "Passenger Cancel Check-in S5" in Fig. 2 introduce the cancellation in the two corresponding execution points. These substitution transitions stand for the module "cancel passenger" that consists on a transition taking a ScenarioInfo token and puts a token with the part of the passenger information into the place "idle passengers", and a token with the part of the agent information into the place "available agents". After the cancellation, the passenger can start again from the beginning of check-in process.

3.4. Generalizing the CPN Model

We explain here how a CPN model can be generalized in order to allow more use cases and their scenarios to be executed in parallel. Therefore we describe the role of colour sets used in the places of the CPN models and how the usage of multiple tokens in the places of a CPN model allows the parallel and concurrent execution of scenarios.

The top-most CPN module is presented in Fig. 6. This module integrates, using substitution transitions, all the CPN modules presented previously, namely those for the main scenario of the "check-in passenger" business use case in Fig. 2 and its exceptions and alternatives, and the ones for the behaviour of passengers in Fig. 5.

We have defined a colour set to represent each actor of the system that appears in the use case. The passengers and the check-in agents are the actors for this system, and their colour sets are listed in Fig. 7.

For the passenger, we consider that it is relevant to have her name, her passport number, and her ticket information. For the agents, only the name is used. As explained above

```
1 colset Passenger =
2          record name: STRING*
3                 passport: INT*
4                 ticket: TICKET;
5 colset Agent =
6          record name: STRING;
7 colset ScenarioInfo =
8          record passenger: Passenger*
9                 agent: Agent;
```

Figure 7. Declaration of the colour sets.

the check-in of a given passenger is triggered when she shows her ticket to one of the available check-in agents.

To save the information about the scenario being executed, we use a colour set based on a record with the information of a passenger and an agent, called ScenarioInfo (see Fig. 7).

The CPN module in Fig. 6 has two tokens in the place idle passengers and one token in the place available agents. Near to each place, there is a rectangle where the values of its tokens are specified.

In the specific case shown in Fig. 6, no more than one passenger can simultaneously do the check-in, because there is only one available agent (called Jim). The introduction of an additional agent (adding one token with her identification to the place available agents) potentially allows two passengers to concurrently perform the check-in operation.

4. Validation of Scenario

This section explains how the obtained CPN model can be used to help on the validation of the initial scenarios for the system under consideration. We consider that the validation task is based on the animation of the obtained CPN model, following the ideas to animate formal specifications [5], [11], and to improve the quality of requirements using animations [18]

To build this animation, developers and users must share a common understanding of the concepts and terms used in the problem domain. The usage of a CPN model for animation purposes implies that it must include elements and constructs that are animation-specific. Since we envision the CPN model to be also useful in later development phases, it is important to clearly separate which parts of the CPN model are animation-specific, and which ones are related to the problem logic.

The CPN models are enriched with animation-specific characteristics to connect the execution of the CPN model in the *CPN Tools* with an animation layer. The connection is guaranteed by the *BRITNeY suite* animation tool [20], which allows some animation-specific code to be added to a CPN model in order to be interpreted by an animation running separately. For the construction of the animation layer, the *SceneBeans* tool [11] was considered, which allows the specification of an animation in a specific XML-file format. We use the *SceneBeans* plug-in available in the *BRITNeY suite* to display and interact with a *SceneBeans* animation. The development of an animation layer using these tools is detailed in [14].

The usage of CPNs for validating requirements through animation was already reported to be successful for workflow systems [9] and for pervasive systems [10].

5. Related Work

We have already presented and discussed some ideas on how to transform use cases and their associated scenarios into a CPN model in [3], [4]. These previous works address the development of embedded systems, and focus on modelling the behaviour associated with the interaction between the system's controller and the elements in the environment (actuators and sensors). This technological perspective implies that the possible behaviours of the human users were only considered, in a very limited and indirect way, in the environment.

In the present work, we tackle the systems at the business level and therefore we can directly reflect the behaviour of the human users in the CPN model, after analysing all the considered scenarios. This allows us to take a broader vision and to potentially consider some erratic or unpredictable situations, where the user behaves in unexpected ways. This proves to be crucial in aiding the analyst on the requirements validation process, since it allows an executable model at the business-level to be created and discussed with the relevant stakeholders.

Research in scenario-based modelling is receiving a considerable attention in the last years. For example, Somé presents an approach to requirements engineering where use cases and scenarios are used to complement each other, namely it uses scenarios to validate use cases [19]. This differs from our approach because we intend that for each use case there is a set of scenarios (that can have alternative branches), and based on these scenarios a global state-based behaviour model is created, where the concurrency between steps are explicitly specified. They consider use cases are related to the system (contrasting with our view in terms of business), together with a more complex scenario's structure than the one considered by us.

Campos and Merseguer integrate performance modelling within software development process, based on the translation of almost all UML behavioural models into Generalised Stochastic PNs [1]. In particular 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 CPNs [17], [6]. 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 CPN model for the considered system, which can be analysed by rigorous techniques or simulated to infer properties some of its behavioural properties.

Pettit and Gomaa describe how CPNs can be integrated with object-oriented designs captured by UML communication diagrams [13]. Their method translates a UML software architecture design into a CPN model, using pre-defined CPN templates based on object behavioural 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 behaviour. 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 behaviour of a set of objects.

6. Conclusion and Future Work

In this paper, we have presented an approach that allows a CPN model to be obtained from a set of scenarios, expressed as UML2 sequence diagrams. The natural support to parallelism and concurrency given by the CPN modelling language permits the CPN model to be considered for simultaneous execution of several scenarios (either of the same use case or of different use cases) and also to represent the parallel activities inside a scenario. Thus, the CPN models support two types of parallelism (intra-scenario and inter-scenario), which means that they can be made rich enough to explore, during animation, situations where several scenarios that might affect each other are executed simultaneously. Our approach also supports the modelling of the human users, namely some possible unexpected behaviours. This feature is important to address more cases in the animation, and thus better validate the requirements with the stakeholders.

Our approach scales up since a CPN model distributes the system's complexity among its graphical and textual parts. For example, in the considered system, the introduction of more passengers and check-in agents does not change the structure of the CPN model; we only need to add extra tokens to some well-identified places. Additionally, we propose scenarios to be reflected in the CPN model in an iterative way. This implies that when, for example, the first alternative scenario is being considered, we should update the CPN model obtained for the main scenario with the extra behaviour introduced by the alternative scenario. Typically, one expects the extra features to be much smaller than all behaviour, since there are some overlapping parts in the scenarios (of the same use case).

As future work, we plan to use our approach in industrial contexts and to evaluate how useful it proves to be for practitioners. Additionally, we expect to provide some tool support for the automatic transformation from scenarios into CPN models.

Acknowledgments

This work was partially supported by Fundação para a Ciência e Tecnologia (FCT), under grant SFRH/BD/19718/2004, and by "AMADEUS: Aspects and Compiler Optimizations for Matlab System Development", Programa FCT (PTDC/EIA/70271/2006).

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)*, ser. LNCS 4024. Springer, 2006, pp. 19–36.

[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)*, ser. LNCS 3530. Springer, 2005, pp. 133–48.

[3] J. M. Fernandes, J. B. Jørgensen, and S. Tjell, "Requirements Engineering for Reactive Systems: Coloured Petri Nets for an Elevator Controller," in *14th Asia-Pacific Software Engineering Conference (APSEC 2007)*. IEEE CS Press, 2007, pp. 294–301.

[4] J. M. Fernandes, S. Tjell, and J. B. Jørgensen, and O. R. Ribeiro, "Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net," in *6th International Workshop on Scenarios and State Machines (SCESM 2007: ICSE Workshops 2007)*, 2007.

[5] A. Gravell and P. Henderson, "Executing Formal Specifications need not be Harmful," *Software Engineering Journal, IEEE*, vol. 11, no. 2, pp. 104–10, 1996.

[6] 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)*, 2004, pp. 213–9.

[7] M. Jackson, *Problem Frames: Analysing & Structuring Software Development Problems*. ACM Press, 2001.

[8] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems," *Int. Journal on Software Tools for Technology Transfer, Springer*, vol. 9, no. 3-4, pp. 213–54, 2007.

[9] J. Jørgensen and C. Bossen, "Requirements Engineering for a Pervasive Health Care System," in *11th IEEE Int. Requirements Engineering Conf. (RE 2003)*. IEEE CS Press, 2003, pp. 55–64.

[10] 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, Springer*, vol. 9, no. 3-4, pp. 353–69, 2007.

[11] J. Magee, N. Pryce, D. Giannakopoulou, and J. Kramer, "Graphical Animation of Behavior Models," in *22nd Int. Conf. on Software Engineering (ICSE 2000)*. ACM Press, 2000, pp. 499–508.

[12] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: a Roadmap," in *Proceedings of the 2000 Future of Software Engineering (FOSE 2000) at Int. Conf. on Software Engineering*. ACM Press, 2000, pp. 35–46.

[13] R. G. Pettit and H. Gomaa, "Modeling Behavioral Design Patterns of Concurrent Objects," in *28th Int. Conf. on Software Engineering (ICSE 2006)*. ACM Press, 2006, pp. 202–11.

[14] O. R. Ribeiro and J. M. Fernandes, "On the Use of Coloured Petri Nets for Visual Animation," in *8th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN 2007)*, 2007, pp. 237–56.

[15] O. R. Ribeiro and J. M. Fernandes, "Some Rules to Transform Sequence Diagrams into Coloured Petri Nets," in *7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN 2006)*, 2006, pp. 227–41.

[16] S. Robertson and J. Robertson, *Mastering the Requirements Process*, 2nd ed. Addison Wesley, 2006.

[17] 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)*, 2000, pp. 103–10.

[18] J. I. Siddiqi, I. C. Morrey, C. R. Roast, and M. B. Ozcan, "Towards Quality Requirements via Animated Formal Specifications," *Annals of Software Engineering*, vol. 3, pp. 131–55, 1997.

[19] S. S. Somé, "Use Cases Based Requirements Validation with Scenarios," in *13th IEEE Int. Requirements Engineering Conf. (RE 2005)*. IEEE CS Press, 2005, pp. 465–6.

[20] M. Westergaard and K. B. Lassen, "The BRITNeY Suite Animation Tool," in *Proceedings of the 27th Int. Conf. on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2006)*, ser. LNCS 4024. Springer, 2006.