http://www-lipn.univ-paris13.fr/~petrucci/PAPERS
In Proc. Workshop on Petri Nets and Distributed Systems (PNDS'08), Xi'an, China, june 2008.

Aggregating views for Petri net model construction

Jörg Desel[1] and Laure Petrucci[2]

[1] Katholische Universität Eichstätt-Ingolstadt, 85071 Eichstätt, Germany
[2] LIPN, CNRS UMR 7030, Université Paris XIII, 93430 Villetaneuse, France

Abstract. When designing a complex system with critical requirements (e.g. for safety issues), formal models are often used for analysis prior to costly hardware/software implementation. However, writing the formal model starting from the textual description is not easy. Approaches to this problem have been presented in the context of Petri nets in [CPR07, CPR08, Des08]. In this paper, since the views adopted previously are both different and complementary, we propose to extract different possible views of the system to be modelled, together with their associated properties. Then, they are aggregated in order to form a complete model. Finally, the properties are checked so as to ensure the consistency of the model w.r.t. the initial requirements.

1 Introduction

When designing a complex system with critical requirements (e.g. for safety issues), formal models are often used for analysis prior to costly hardware/software implementation. However, writing the formal specification starting from the textual description is not easy. It is well known that the first phase of modelling is of particular importance. Only if models that faithfully present the part of reality (or of intended reality) to be modelled are used in system design, the final system can be expected to behave correctly according to the requirements. Conversely, any error in an early stage of modelling will cause very costly redevelopments in later phases.

Different suggestions for a systematic analysis of requirements have been developed in the context of process models. Some are based on extracting pre- and post-conditions of actions from a textual description of the system under consideration, using them as local vicinity of corresponding transitions of Petri net models [FKM03, MKE07, CPR07, CPR08]. Others extract processes specification from the textual description [Des08], indicating when the actions should take place w.r.t. one another.

In this paper, we show that both approaches are complementary. We also introduce a dual approach to processes, that are *lifelines*: instead of describing the successions of actions, as a process does, they represent a succession of possible local states as might e.g. be observed by a user of the system.

The aggregation of different views is also one of the central issues in object oriented modelling. UML offers various diagram types that allow for the modelling of different views of the same system. Our notion of process specification is closely related to specifications with message sequence charts whereas our notion of lifelines has similarities to so-called life cycles of objects. There are several suggestions to transform these different diagrams to Petri nets which can then be integrated to a complete model. Whereas the recent paper [vHSSV06] is inspired by UML, previous work [SS00,WSY98] directly applies to UML diagrams. As we do in our approach, [vHSSV06] emphasizes consistency of the aggregated models. The main difference with our work is that [vHSSV06] suggests to model workflow nets quite directly from scenarios whereas we elaborate this modelling step more deeply.

Section 2 shows that different views of the system coexist within its informal description, that can be interpreted via pre/post-conditions, processes or lifelines. Such views can be individually translated into a Petri net, as shown in section 3. Moreover, some properties — such as invariants — are collected, thus expressing additional details. Applying this approach to all views of the system mentioned in the textual description leads to a collection of nets which should be aggregated, as in section 4, so as to obtain a complete Petri net model. The aggregation must be performed according to specific rules in order to preserve as much as possible consistency between the Petri net behaviour and the initial individual requirements. Finally (section 5), the model behaviour must be consistent with the expected (known) properties collected in the previous phases. This consistency check can be done by standard Petri net techniques such as model checking. However, before that, the properties have to be formalised, i.e. translated in some logic. Hence, the validity check of requirements concerns the three different views (pre/post, processes and lifelines), the expected properties as well as the formalisation of the views and of the properties. Thus, the verification phase does not support correctness of the model but rather supports validity [Des02]. In the positive case, we can hope that the model actually models the system under consideration. In an additional step, desired properties can be formulated, formalised and analysed. Only if we can assume that the formalisation is valid, verification with respect to this additional specification is meaningful.

2 Analysing requirements descriptions

When starting from an informal description, a system may be described in many different ways. [CPR07, CPR08] rely on the identification of *constituent features* of the system, which can be of two kinds: *state observers* and *events*. They correspond to *noun phrases* and *verbal phrases* respectively. Once these constituent features are identified, properties are also extracted through the text analysis. We consider first relations between state observers and events. Events can have *pre-conditions* and/or *post-conditions*. Such an approach gives a local view of the

actions happening in the system: we know that for an event to occur, some pre-condition should be satisfied, and that after it has occurred, some post-condition holds. However, this view may be incomplete, i.e. partial.

Indeed, let us consider a simple example of a system to be designed, where several people are involved in a factory plant. The factory processes and ships glass panes that must be cut to some specific size. Each of the persons handling the glass panes after it has been cut has a different and partial vision, e.g.:

- the *worker* moving the glass to the warehouse knows that for action `stock` to happen, a glass pane is required from the `end of the conveyer belt`, and transports it to the warehouse `depot`. He also knows that the action `cut` takes a glass pane `on the belt` and will move it afterwards to `the end of the belt` ;
- the *stock manager* knows that for the same action `stock`, when the glass pane arrives at the `end of the conveyer belt` he creates an `entry in the database` indicating that the pane is in the stock. But the *stock manager* does not know where the glass pane is stocked in the depot since (s)he does not handle it. The stock manager also informs us that "when `shipping` items, they are `removed from the database` and marked as `sold`".

The examples above (and the method from [CPR07, CPR08]) consider the pre- and post-conditions of events. But a similar analysis can be applied to state observers as well. For example, the description could mention that "a `processed glass pane` is produced by a `cutting` action and then moved by action `stock`". In this case the description concerns the vicinity — i.e. pre- and post-conditions — of a resource (or state observer) instead of the vicinity of an action (or event). Note that in some contexts a resource may be a document or an object.

Other parts of the description may indicate how some *successions of actions* are meant to occur, thus leading to a *process view* of the (partial) order of events, as advocated in [Des08]. In such a case, the steps or pre/post-conditions between events are unknown and cannot be extracted from the informal requirements description. This view of processes can also include multiple branches, indicating that at some stage, several actions may take place simultaneously.

In our glass factory example, a *supervisor* describes the work in the factory: "Glass is `loaded` on a conveyer belt, `moved` to the workstation and then `cut` ; one part is `dumped` and another `stocked`".

In a workflow, a process will start with some action and end later with a final termination action. When modelling other complex systems, such as protocols, loops become inherent to the normal functioning.

In our example, the glass is cut into a pane to be sold and another part. In the previous description, this other part is dumped. But a possibility is also to reuse it, and cut it (for example to produce smaller glass panes).

This alternative scheme will be used in section 3.2 to illustrate loops in processes, and we shall see that handling loops requires the use of rather elaborate techniques.

Similarly, the flow of documents or resources (instead of actions) can be detailed. In that case, we talk about *lifelines*, which can be seen as a dual to the process view. They can express e.g. the circulation of a document within an administration. There, alternatives are allowed, indicating multiple possibilities. As we shall see in section 3.3, even though loops may occur in lifelines, they are easier to handle than those of processes.

Let us now add to our glass factory the description of the *part delivery*: "The glass pane in the `depot` is loaded onto a `fenwick` which brings it either to a `truck` which will directly deliver the pane to a `shop`, or it will be brought to be packaged in a `crate` which will be put on a `train` (e.g. for delivery in a foreign country)".

The textual description also often contains *static properties*. As indicated in [CPR07, CPR08], these properties should be collected during the textual description analysis in order to be checked at a later stage (see section 5).

In the glass pane factory, the stock manager states that there should be exactly one database entry per item in the stock. This is an *invariant property*, where the number of entries in the database is at all times equal to the number of items in the stock.

Once the textual description of the system has been analysed, we have a collection of pre/post-conditions, processes, lifelines and properties. In the next section, we show how the first three can be pictured for facilitating their understanding and how to model them individually with a Petri net.

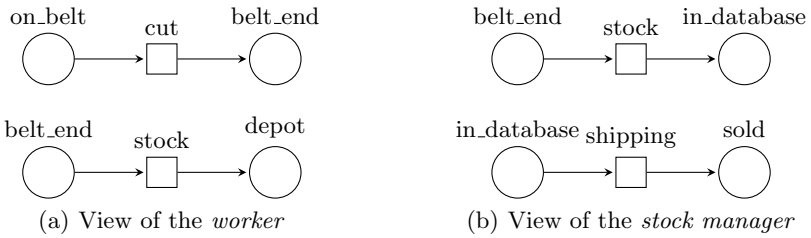
3 Requirements specification schemes

In this section, we shall study the three kinds of views identified in the previous section. For each of these, we provide a graphical representation inspired from Petri nets — actions represented by boxes, state observers by circles — and then show how they can be transformed into a Petri net according to simple specification schemes.

3.1 Pre/post view

The *pre/post view* contains both actions and state observers. Therefore, it maps naturally to a Petri net model. The central element of the view is mapped to a transition if it is an action, to a place otherwise, and its pre-conditions to input places, its post-conditions to output places (to transitions respectively). Places and transitions thus derived are labelled with the associated piece of information (name of the action, ...). With this view, the graphical representation and the Petri net are identical.

Let us model the example of the two persons in section 2. We obtain the Petri nets in figure 1.



(a) View of the *worker* (b) View of the *stock manager*

Fig. 1. Example of the pre/post views

3.2 Process view

The *process view* is exclusively based on actions. Therefore, it can be schematised using *boxes* only. These boxes are linked by arcs indicating the partial order of actions. When several arcs exit the same node, *all subsequent actions occur in parallel*.

The process view of the glass factory example is pictured in figure 2.

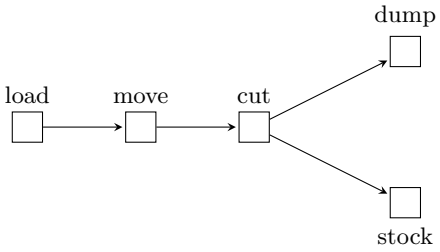


Fig. 2. Example of the process view

In the process view, information on the actions is available, while the intermediate states between actions are unknown. Thus, when translating the graphical process description into a Petri net, the actions obviously map to transitions labelled by their name, while the intermediate states are mapped to places with an as yet unknown meaning. Hence, we introduce, on each arc between two actions, a place labelled with a `?`. These places, called *`?`-places* in the sequel, will be further processed during the aggregation phase described in section 4.

Figure 3 shows the Petri net derived from the process description in figure 2.

Note that when several arcs exit an action `a` in the process description, the Petri net has one `?`-place per original arc. Thus, after firing a transition `a` all of its

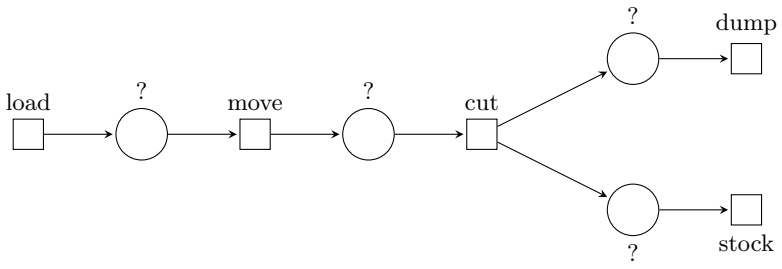


Fig. 3. The Petri net derived from the process view

output ?-places are marked, enabling all subsequent transitions. This behaviour is consistent with the possibility of concurrent actions explained in section 2.

Some series of actions within a process might be repetitive, inducing *loops* in the process view. This is the case in our example if we consider that a part cut can be either dumped or reused, as pictured in figure 4. Several problems arise in such a case:

- an *alternative* between different actions is implicitly introduced. This is not consistent with the semantics of the process view ;
- loops may have side effects if not dealt with very carefully ;
- the values of state observers when the loop is restarted are not clear. This is particularly the case when the start action has multiple pre-conditions.

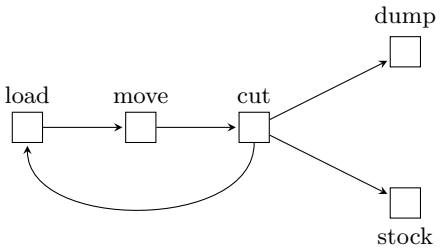


Fig. 4. Example of a process loop view

In order to take loops into account, we propose, inspired by [BDM08], to apply a *synthesis* procedure in order to automatically obtain a Petri net modelling the process. Many works deal with Petri nets synthesis [Dar07, BDLM08b, BDLM08a], and present algorithms. Applied to the process description, they automatically generate a Petri net model (or else the synthesis algorithm fails).

This resulting Petri net is guaranteed to have precisely the specified behaviour, provided any Petri net with this behaviour exists. Otherwise, the generated Petri net has the specified behaviour and minimal additional behaviour. Synthesis approaches developed in the context of process mining [BDLM07] assume that only part of the behaviour was specified. They have the additional goal to generate a comparably small (and easy to understand) Petri net. Synthesis is not the focus of this paper, and will not be detailed any further.

3.3 Lifeline view

The *lifeline view* is the dual of the process view. It deals exclusively with states which are then pictured using circles. When several alternatives are possible, the graphical representation contains as many outgoing arcs as subsequent possibilities. The semantics is then a *choice among the different alternatives*.

Figure 5 depicts the lifeline view in the example.

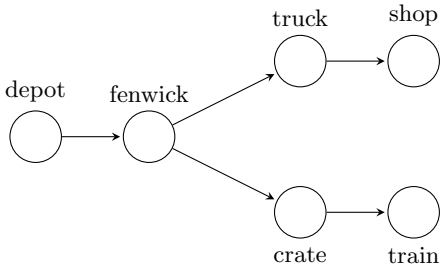


Fig. 5. Example of the lifeline view

Dually to the process view, in the lifeline view information on the states is available, while the intermediate actions are unknown. Thus, when translating the graphical lifeline description into a Petri net, the states obviously map to places labelled by their name, while the intermediate actions are mapped to transitions with an as yet unknown meaning. Hence, we introduce, on each arc between two states, a transition labelled with a `?`, called *`?`-transition* hereafter.

Figure 6 shows the Petri net derived from the lifeline description in figure 5.

Note that when several arcs exit a state `S` in the lifeline description, the Petri net has one `?`-transition per original arc. Thus, once marked, state `S` can be used by one of its output `?`-transitions indicating a choice between these actions as explained in section 2.

Dealing with *loops* in lifeline views is not an issue as it is the case with process views. In lifelines, the loop does not violate the arc semantics since alternatives

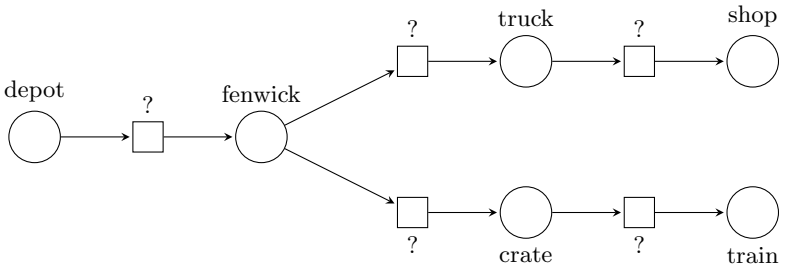


Fig. 6. The Petri net derived from the lifeline view

are allowed. Moreover, the starting state of the loop is known and consistent from one iteration to the other.

3.4 Required properties

The textual description allows for identifying *required properties*. In [CPR07], these properties are categorised into pre/post-conditions and additional properties. In our approach here, pre/post-conditions are already taken into account in the Petri net specification. Most of the other properties can be expressed in terms of either a temporal logic, e.g. LTL or CTL [BBF+01], or Petri nets *invariants*. Transforming the different views into Petri net models has lead to identifying a collection of named places and transitions on which the properties can be expressed.

The example property, indicating that the number of entries in the database is at all times equal to the number of items in the stock, is then expressed by the following P-invariant:

$$\forall M \in [M_0\rangle : M(\texttt{in\_database}) = M(\texttt{depot})$$

where M is a reachable state of the global system, and M_0 its initial state.

4 Aggregating specification schemes

After having analysed the textual description, derived the different views and translated them into individual Petri nets, we have to aggregate these into a single Petri net modelling the complete system. This net will further be analysed (see section 5) to check its properties. However, we shall guarantee that by construction the net will be as consistent with the initial requirements specification as possible.

The *aggregation* of the individual nets should follow the following rules:

– all places (respectively transitions) named with the same label (except `?`) are fused into a single place (respectively transition) with the same name. This is consistent since the name of a net element is derived from the textual description and carries a single informal meaning. In practice, the visions of the different stakeholders often use different terms in their description. The model designer then has the additional task to identify the homonyms and synonyms so as to have a single representation of a same object ;
– some of the `?`-places and `?`-transitions correspond in fact to some already identified place or transition (which might be labelled by `?`). In this case, the place or transition can be deleted because this operation does not change the overall behaviour of the net. A simple operation guaranteeing the same behaviour consists in deleting a `?`-place (or `?`-transition) `x` when there exists another place (respectively transition) `y` with at least the same input and at least the same output:
  • for a `?`-place x, if $\exists y \in P : {}^\bullet x \subseteq {}^\bullet y \wedge x^\bullet \subseteq y^\bullet$, x can be deleted ;
  • for a `?`-transition x, if $\exists y \in T : {}^\bullet x \subseteq {}^\bullet y \wedge x^\bullet \subseteq y^\bullet$, x can be deleted ;
  where P (T respectively) denotes the set of places (transitions) at this stage of the procedure.

The aggregation of the individual Petri nets corresponding to the pre/post, process and lifeline views in the glass factory example lead to the Petri net model of figure 7. The places and transitions fused together to constitute a single one are:

– `cut` in one pre/post view of the worker and the process view ;
– `belt_end` in both pre/post views of the worker and one of the stock manager pre/post views ;
– `stock` in a pre/post view of the worker, a pre/post view of the stock manager and the process view ;
– `depot` in a pre/post view of the worker and the lifeline view.

We also note that the dashed `?`-place has the same input and the same output as place `belt_end`. Therefore, the dashed part of the net in figure 7 can be deleted, without any impact on the Petri net behaviour. Note that the place `on_belt` cannot be deleted although there is another place (below `on_belt`) satisfying the condition that its pre-set is a superset of the pre-set of `on_belt` and that its post-set is a superset of the post-set of `on_belt`. The reason is that we only delete `?`-places.

In the next section, the net properties will be checked so as to validate the Petri net model w.r.t. the requirements.

5 Checking consistency

There exists a wide range of techniques and tools to check Petri nets dynamic and static properties. The main approach consists in generating the system state

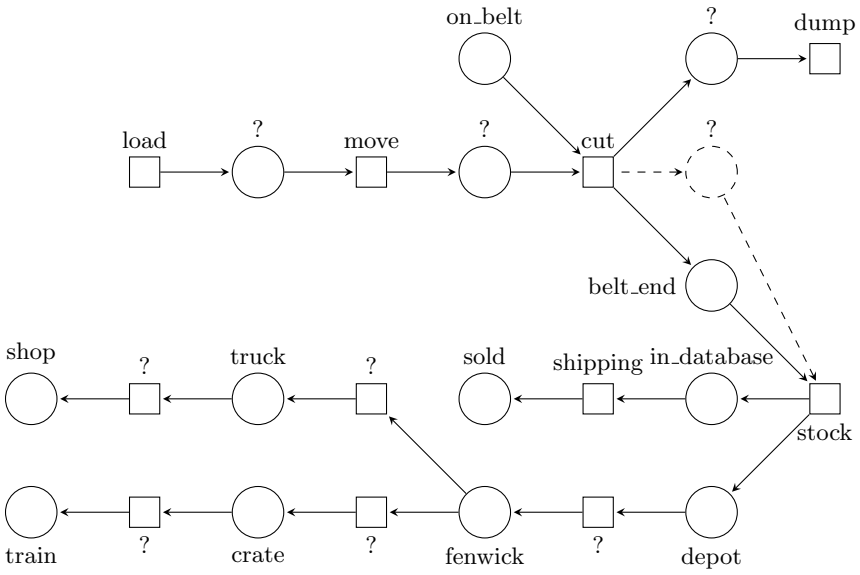


Fig. 7. The aggregated Petri net

space and analysing it afterwards. However, the state space explosion problem occurs with large systems and many tools propose efficient methods to cope with this. Checking that an invariant holds can be done on the state space by checking that the invariant holds for all generated states or by simple flow verification.

All these techniques are implemented in tools, which handle Petri nets [VIP, DJLN03] or high-level nets [LM07, CPN], temporal logics [DLP04, KP08], user-defined properties [CPN], synthesis [BDLM08c], ...

The Petri nets designed should be correct by construction w.r.t. the pre/post, process and lifeline views. However, when using the synthesis procedure, a net is automatically obtained, the behaviour of which should be checked against the requirements. Moreover, even though the individual nets behave as expected, the aggregation step, and thus their composition, may induce other behaviour. For example, some states may never be reachable and parts of the net may become dead. This does of course not respect the requirements of the overall system. Therefore, the behaviour and expected properties of the system must be checked once the Petri net is built.

Let us now analyse the glass factory example. Initially, no glass pane is loaded, none is in the warehouse, and nothing has been sold or shipped. Hence, the initial marking is empty. However, we immediately see that transition `load` can be fired infinitely often, which should obviously not be possible: only a single glass pane can be on the conveyer belt at any time. Note that this was not part of

the initial textual description. Moreover, transition `cut` can never be fired since place `on_belt` is never marked. In a real case study, this would lead to discussing with the domain experts the validity of new assumptions. The proposal could also be that there are two glass panes, one just loaded and one ready to be cut. So, let us consider additional pre/post views where the `conveyer belt is free` before being `loaded` and after the glass has been `cut`, and there is a glass pane `on the belt` after it has been loaded and until it is `cut`. This leads to a modified Petri net (following the same aggregation rules as in section 4). Initially, the conveyer belt is free, thus the initial marking contains one token in place `free_belt`. A new property states that either there is a glass pane on the conveyer belt or the conveyer belt is free.

Another problem occurs in the system. Indeed, the property indicated by the stock manager, stating that there is exactly one entry in the database for each glass pane in the depot does not hold since the `shipping` transition and the `?`-transition exiting the `depot` place can occur independently. This should not be the case, and the database entry should be removed at the same time as the glass pane exits the depot, meaning that the two transitions should be synchronised.

The newly added parts are shown with thick lines in the modified Petri net of figure 8.

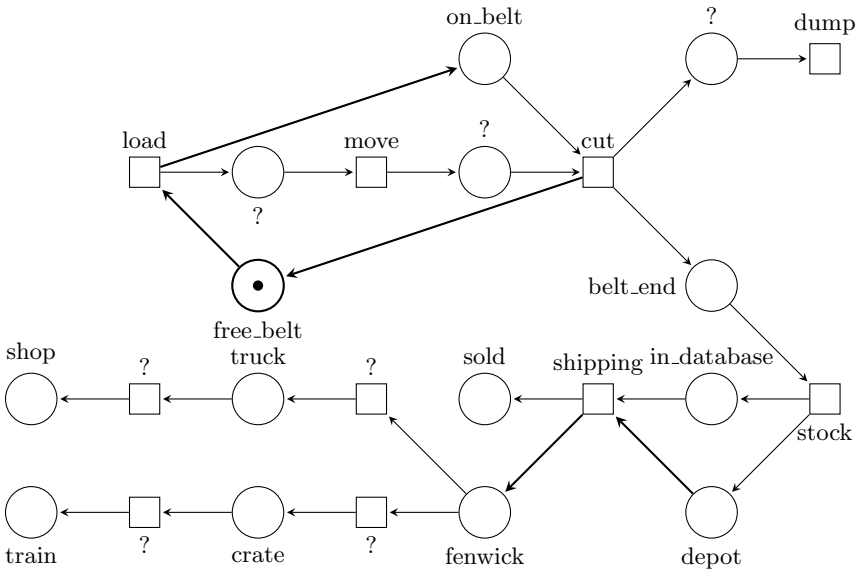


Fig. 8. The modified Petri net

The new property is easily captured by the following invariant, which holds:

$$\forall M \in [M_0\rangle : M(\texttt{on\_belt}) + M(\texttt{free\_belt}) = 1$$

The number of items in the database and depot are now the same:

$$\forall M \in [M_0\rangle : M(\texttt{in\_database}) = M(\texttt{depot})$$

6 Conclusion and future work

Designing complex systems is a difficult task that can be eased using a structured approach. The design of a formal model gives a certain confidence in the correct behaviour of the system under consideration. In this paper, we have shown that the textual description of the system, possibly described by different domain experts, can be considered using several aspects, including partial views and properties. The partial views which can be obtained can focus on pre- and post-conditions, but also on partial orders of events that constitute processes, or on lifelines which represent successions of possible states. We have given a graphical representation for these three types of views and shown how they map to basic Petri net models. This collection of individual nets can be aggregated so as to form a single Petri net modelling the complete system. The textual requirements description also expresses expected properties that the system should satisfy. These properties are also modelled so that we can check whether they hold for the Petri net. In this case, one can have a greater confidence in the correct behaviour of the system — provided, of course, that the design (pre/post, processes and lifelines views as well as formalisation of properties) is consistent with the informal requirements.

As shown in section 3, it is necessary in some cases to apply a synthesis algorithm in order to obtain a Petri net in which the intended behaviour can occur (but maybe more). In the future, we shall adapt the synthesis techniques to our multiple views approach.

The aggregation rules to glue and reduce the individual nets together are for the moment very simple. However, we can identify some cases for which more elaborate rules would apply. All rules should respect the initial specifications, i.e. the original views should actually be views of the aggregated model, but possibly in a modified way. For validation issues, it would be useful to automatically extract these views from the complete model.

The design is generally structured both horizontally and vertically. The horizontal structuring, addressed in [CPR08] consists in constructing the model in a modular fashion: the modules are Petri nets communicating with each other through places and/or transitions. They are extracted from the textual description of the system. Vertical structuring concerns refinement issues. For a large system, the first approach is often rather coarse and events are refined (detailed further) at a later stage. Such a vertical structuring is tackled in [Des08].

Combining horizontal and vertical structuring raises several issues such as synchronisation of refined transitions and termination. The approach in this paper shall be further developed to integrate these structuring concepts.

Various modularisation approaches and refinement approaches for Petri nets have extensively been studied in the literature. One of the main research issues always is preservation of properties, to facilitate modular analysis. This applies in particular to Petri net formalisms based on objects and/or object orientation. However, every compositional modelling approach has to restrict the composition operators severely and begins with simple formal models. As already mentioned in the introduction, the aim of our work is to start with informal, textual descriptions and thus with arbitrary composition of views. So our main focus is on the formalisation of views which are as general as possible and subsequent aggregation rather than on restricted but behaviour preserving composition formalisms.

To validate this work, it will be necessary to process larger case studies. We plan to apply it first to academic examples of greater size, and then an industrial protocol handling distributed databases for mass storage. A tool support automating the Petri net generation and aggregation procedures will then be most helpful. Hence, designing tool support is a major issue to address large case studies.

References

[BBF+01] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[BDLM07] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process mining based on regions of languages. In *Proc. Int. Conf. on Business Process Management, Brisbane, Australia*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer, September 2007.

[BDLM08a] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Synthesis of Petri nets from finite partial languages. *Fundamenta Informaticae*, 2008. To appear.

[BDLM08b] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Synthesis of Petri nets from infinite partial languages. In *Proc. 8th Int. Conf. on Application of Concurrency to System Design (ACSD'08), Xi'an, China*. IEEE Comp. Soc. Press, June 2008. To appear.

[BDLM08c] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Synthesis of Petri nets from scenarios with VipTool. In *Proc. 29th Int. Conf. on Application and Theory of Petri Nets, Xi'an, China*, Lecture Notes in Computer Science. Springer, June 2008. To appear.

[BDM08] R. Bergenthum, J. Desel, and S. Mauser. Synthesis of Petri nets for business process design. In *Proc. Workshop Verhaltensmodellierung: Best Practices und neue Erkenntnisse, Modellierung 2008, Berlin*, March 2008. To appear.

[CPN] *cpntools*. `http://wiki.daimi.au.dk/cpntools/cpntools.wiki`.

[CPR07] C. Choppy, L. Petrucci, and G. Reggio. Designing coloured Petri net models: a method. In *Proc. Workshop on Practical Use of Coloured Petri Nets, Aarhus, Denmark*, October 2007.

[CPR08] C. Choppy, L. Petrucci, and G. Reggio. A modelling approach with coloured Petri nets. In *Proc. 13th Int. Conf. on Reliable Software Technologies — Ada-Europe, Venice, Italy*, Lecture Notes in Computer Science. Springer, June 2008. To appear.

[Dar07] Ph. Darondeau. Synthesis and control of asynchronous and distributed systems. In *Proc. 7th Int. Conf. on Application of Concurrency to System Design (ACSD'07), Bratislava, Slovak Republic*, pages 13–22. IEEE Comp. Soc. Press, July 2007.

[Des02] J. Desel. Model validation — a theoretical issue? In *Proc. 23rd Int. Conf. on Application and Theory of Petri Nets, Adelaide, Australia*, volume 2360 of *Lecture Notes in Computer Science*, pages 1–23. Springer, June 2002.

[Des08] J. Desel. From human knowledge to process models. In *Proc. of UNISCON 2008, Klagenfurt, Austria*, volume 5 of *LNBIP*, pages 84–95. Springer, April 2008.

[DJLN03] J. Desel, G. Juhás, R. Lorenz, and C. Neumair. Modelling and validation with VipTool. In *Proc. Int. Conf. on Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 380–389. Springer, June 2003.

[DLP04] A. Duret-Lutz and D. Poitrenaud. SPOT: an extensible model checking library using transition-based generalized Büchi automata. In *Proceedings of the 12th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'04)*, pages 76–83, Volendam, The Netherlands, October 2004. IEEE Computer Society Press.

[FKM03] G. Fliedl, C. Kop, and H.C. Mayr. From scenarios to KCPM dynamic schemas — aspects of automatic mapping. In *Natural Language Processing and Information Systems, 8th International Conference on Applications of Natural Language to Information Systems*, volume 29 of *LNIP*, pages 91–105. Gesellschaft für Informatik, 2003.

[KP08] K. Klai and D. Poitrenaud. MC-SOG: An LTL model checker based on symbolic observation graphs. In *Proc. 29th Int. Conf. on Application and Theory of Petri Nets, Xi'an, China*, Lecture Notes in Computer Science. Springer, June 2008. To appear.

[LM07] LIP6-MoVe. *The CPN-AMI Home page.* `http://www.lip6.fr/cpn-ami`, 2007.

[MKE07] H.C. Mayr, C. Kop, and D. Esberger. Business process modeling and requirements modeling. In *First International Conference on the Digital Society (ICDS'07). Los Alamitos, CA, USA*, pages 8–11, 2007.

[SS00] J. Saldhana and S. M. Shatz. UML-diagrams to object Petri net models: an approach for modeling and analysis. In *Proc. Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'00)*, pages 103–110, July 2000.

[vHSSV06] K. v. Hee, N. Sidorova, L. Somers, and M. Voorhoeve. Consistency in model integration. *Data & Knowledge Engineering*, 56(1):4–22, 2006.

[VIP] *Viptool*. `http://www.ku-eichstaett.de/Fakultaeten/MGF/Informatik/Projekte/Viptool.de/`.

[WSY98] J. L. Woo, D. C. Sung, and R. K. Yong. Integration and analysis of use casesusing modular Petri nets in requirements enginnering. *IEEE Trans. on Software Engineering*, 24(12):1115–1130, 1998.