Modelling and Validation with VipTool

Jörg Desel, Gabriel Juhás, Robert Lorenz, and Christian Neumair*

Lehrstuhl für Angewandte Informatik
Katholische Universität Eichstätt, 85071 Eichstätt, Germany
{joerg.desel,gabriel.juhas,robert.lorenz,
christian.neumair}@ku-eichstaett.de

Abstract. This paper describes concepts and features of a new version of the VipTool. As for the original VipTool, the main issue of this software package is to generate, analyze and visualize process nets, representing the partial order behavior of business process models given by Petri nets. Whereas the original VipTool was implemented in the scripting language Python, the new VipTool is a completely new and modular implementation in Java that allows to add arbitrary extensions in a more flexible way. In this new version, several drawbacks that had appeared previously where eliminated. Moreover, the new VipTool contains additional features such as a more comfortable editor as well as eps- and XML-interfaces. The main improvement is a better support of step-wise validation of models and specifications and, alternatingly, partial verification (testing) of specification implementations. This paper also presents a small case study explaining how the VipTool supports these design steps.

1 Introduction

VipTool was originally developed at the University of Karlsruhe within the research project VIP[1] as a tool for modelling, simulation, validation and verification of business processes using Petri nets. There exist many software packages, developed at universities or software companies, which support modelling of business processes using different modelling formalisms (e. g. EPKs [8], different variants of Petri nets [1,2] etc.). Most of them, designed to analyze a Petri net model, compute the state space or use linear algebraic methods (e.g. Design/CPN, INA, Renew). The tool Woflan enables to model business processes by workflow nets [2], and to check whether the designed model fulfills the desired properties, such as soundness etc. In comparison, VipTool generates concurrent runs of a given Petri net model of a business process. If a Petri net is not too large and has only finite runs, all runs are generated. There exist some other software packages (e.g. PEP) which use the same theoretical approach of a finite and complete prefix of the unfolding of a Petri net model ([6]). But in the case the Petri net is too complex to generate the complete prefix, VipTool still

* supported by DFG: Project "SPECIMEN"

[1] Verification of Information Systems by evaluation of Partially ordered runs

generates a substantial set of runs, because these runs are computed on the fly. Moreover, VipTool is the first tool which is able to visualize these concurrent runs.

The first version of the VipTool was written in Python [7]. The version of the VipTool presented here is a complete redesign using standard object oriented design. It is re-implemented in Java. The paper is organized as follows: In Section 2 the business process design steps supported by VipTool are described. Section 3 presents a brief description of the VipTool features. A simple case study illustrates the functionality of VipTool in Section 4. Finally, the conclusion outlines the future development.

2 The Business Process Design

VipTool was originally designed as a simulation tool for business process models. Whereas usually simulation creates and visualizes sequences of transition occurrences representing events of the business process, VipTool is based on partially ordered runs, given by process Petri nets. This concept is explained in detail in [3]. The main advantages of this approach compared to sequential simulation are shortly:

- a more efficient representation of the behavior of business processes
  (a single process net represents a set of sequences of transition occurrences which can be quite large in the presence of concurrency),
- a higher degree of expressiveness
  (the flow of objects and information is explicitly given in process nets by paths while in general it is not even implicitly given by sequential runs), and
- more efficient analysis methods for runs
  (relevant properties can be checked by efficient algorithms, exploiting the graphical structure of process nets).

One of the main issues of modelling a business process is its analysis with respect to intended properties. This analysis requires a formalization of both, the business process and the properties to be analyzed. In turn, the value of the analysis depends on the correctness of the model with respect to the actual business process and also on the correctness of the formal representation of the specification with respect to the actual intended property. To avoid confusion with the formal interpretation of the term *correctness* (a model is correct if it satisfies a property formulated by a specification), we call a model *valid* if it faithfully represents the business process. Similarly, a *valid* specification faithfully represents a relevant property. Most approaches to business process design and according tools assume validity of models and specifications and concentrate on analysis and verification issues. However, experience shows that in many cases negative results of analysis or verification are caused by invalid models or invalid specifications rather than by incorrect business processes. Even worse, positive results do not mean much if validity of models and specifications cannot be guaranteed.

The paper [4] discusses how validation of Petri net models in general can be supported by formal means and by Petri net tools. In particular, it is suggested

- to validate models by generation, visualization and inspection of process nets that represent the behavior of each system component by a process net component,
- to formalize specifications graphically within the model representation to avoid error prone new syntactical means,
- to validate specifications of a valid model by presenting separately process nets that satisfy the specification and those that do not satisfy a specification (clearly, this only makes sense if specifications can be interpreted on runs such as linear time temporal logic specifications).

Using this approach, a business process model is designed from constructive specifications, given by a Petri net that represents an early version of a business process or the environment of the business process to be developed, and declarative specifications, representing required properties of the process.

For complex business processes we suggest a step-wise procedure in [4]. The first step is creating an initial model representing the constructed specification explained above. Sometimes this model can be derived by a folding operation from known scenarios that have to be supported by the business process. In any case, the model has to be validated, as mentioned above. Then, iteratively, the following steps are performed:

- A requirement to be implemented is identified and formalized in terms of the graphical language of the model.
- This formal specification is validated by distinction of those process nets that satisfy the specification from all other process nets. This way, the question *what behavior is excluded by the specification?* gets a clear and intuitive answer. The specification is changed until it precisely matches the intended property.
- The valid specification is implemented, i.e., new elements are added to the model such that the extended model matches all previous and the new specification. Obviously, this step requires creativity and cannot be automized. However, again by generation and analysis of process nets it can be tested whether the extended model satisfies the specifications (actually, when all runs can be constructed, this test can be viewed as a verification). At this stage, other verification methods can be applied as well.
- If some requirements are still missing, we start again with the first item, until all specifications are validated and hold for the designed model.

VipTool supports all above mentioned steps. In particular, process nets representing partially ordered runs of business processes are generated from a given Petri net model of the business process. They are visualized, employing particularly adopted graph-drawing algorithms. Specifications can be expressed on the system level by graphical means. Process nets are analyzed w.r.t. these specified properties. The distinction of process nets that satisfy a specification is

supported. For the test phase, simulation stops when an error was detected. At present, the new version of VipTool only supports low-level Petri nets (whereas the previous one dealt with a restricted kind of predicate/transition nets). Also, generation of a model from a given set of runs representing known scenarios has not been implemented yet.

3 Description of the VipTool

The VipTool consists of VipEditor, which enables the user to design the business process Petri net model. The modelling is very intuitive and drawing and painting features can be used analogously as by standard Windows applications. Size, color, fonts, and other usual graphical parameters can be easily set by the user for all draw elements (places, transitions, arcs, labels, etc.). All standard editing features, such as select, move, copy, paste, undo, redo etc. are implemented in the usual way. The user-friendly environment is supported by many other features, for example by automatic alignment and by click-and-drag-points of net arcs. Usual token game simulation is also a part of the VipEditor. Figure 1 shows a screen-shot of the VipEditor with an example of a simple business process model.

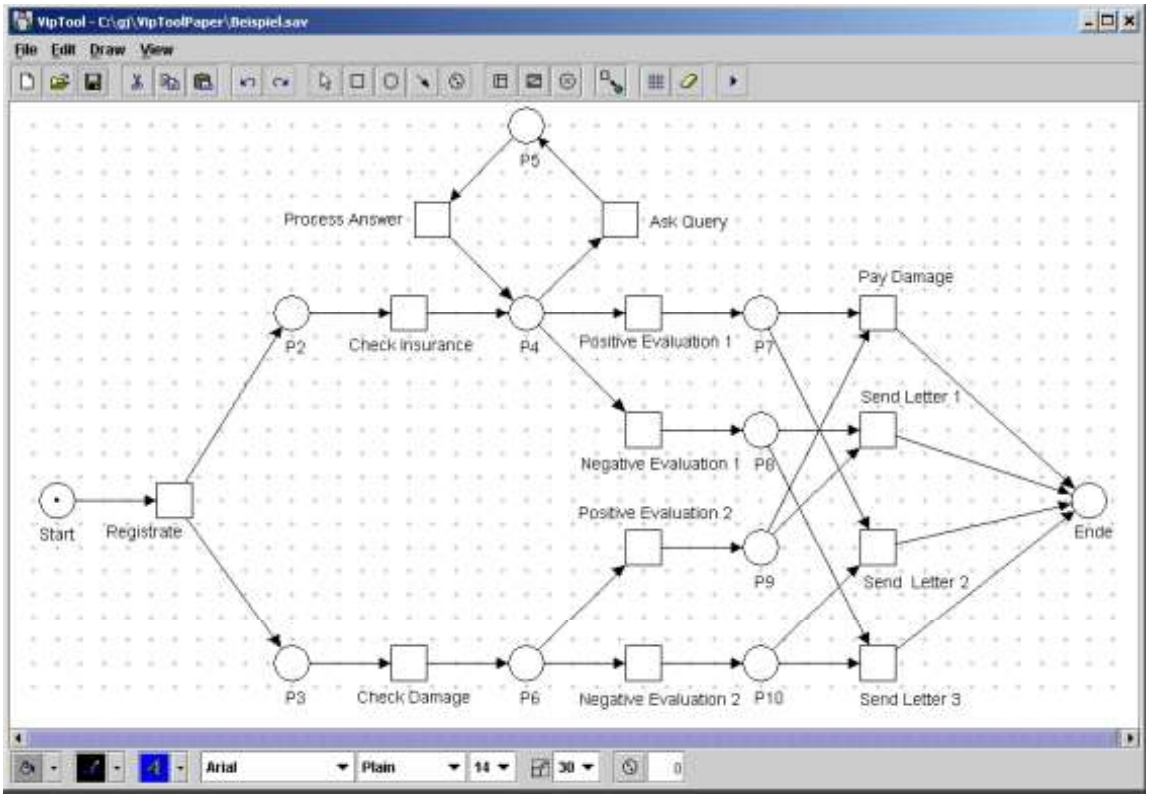

Fig. 1. Screenshot of VipEditor, including an example net, which is explained later in a case study.

An important feature of every graphical editor is the export of the image to a file in an appropriate standard graphic format. The VipTool supports exporting of pictures in Encapsulated Post Script (eps). To support the exchange of Petri nets between different tools, an XML exchange format was developed in [9]. This format is now widely accepted as a standard exchange format for Petri nets. The VipTool allows export of XML files in this format. Figure 2 shows a part of the XML file of the net from Figure 1.

```
- <pnml>
  - <net
      id="VipCanvas[,0,0,1069x686,alignmentX=null,alignmentY=null,b
      order=,flags=9,maximumSize=,minimumSize=,preferredSize=jav
      a.awt.Dimension[width=1005,height=645]]" type="stNet">
    - <name>
        <value>C:\gj\VipToolPaper\Beispiel.xml</value>
      </name>
    - <transition id="VipTransition@1c8f91e">
      - <graphics>
          <position x="580" y="300" />
        </graphics>
      - <name>
          <value>Positive Evaluation</value>
        - <graphics>
            <offset x="-47" y="34" />
          </graphics>
        </name>
      </transition>
    - <place id="VipPlace@1d27069">
      - <graphics>
          <position x="480" y="300" />
        </graphics>
      - <name>
          <value>P4</value>
        - <graphics>
            <offset x="-5" y="35" />
          </graphics>
        </name>
      - <initialMarking>
          <value>0</value>
        </initialMarking>
      </place>
```

Fig. 2. A part of the export of the net in Figure 1 to an XML file

The VipEngine computes the runs of the modelled Petri net. The computation of the runs of the modelled Petri net by VipEngine is based on the construction of the complete prefix of the branching process of the net. In addition to standard cut-off criteria for terminating potentially infinite runs (described in [7]), further termination criteria like bounds for the number of events or for the depth of the branching process (to be specified by the user) are implemented.

To obtain complete runs within the branching process as fast as possible, the branching process is first constructed into depth according to the following strategy: One begins with a certain starting cut (which in the beginning is the cut of conditions representing the initial marking). When trying to add a new event to the branching process, first one searches for an event enabled under the actual cut which uses conditions constructed in the previous step. If this is not possible, one searches for any event enabled under the actual cut. If there are no such events, a possible run is finished and stored, and an event is added, which is enabled under a set of already constructed concurrent conditions. Such possible events are stored in a list which is always updated after adding an event. Now one completes the next run by choosing a cut which includes the preset of that event as the new starting cut and repeating the above strategy. That new starting cut is chosen to be maximal (w.r.t. the flow relation given by the branching process) with the property that it includes the preset of the added event. In such a way, a set of runs is stored on the fly, which cover the whole branching process. Note, that in general not all possible runs are stored on the fly using this procedure, since there can be more than one possible new starting cut. Nevertheless that strategy seems to be a good trade-off between efficiency and completeness. At last the user can decide to compute the whole set of runs from the constructed branching process by applying an appropriate clique-algorithm. To guarantee, that substantial runs are computed on the fly, priority criteria can be employed in the beginning.

The runs are visualized using VipVisualizer, which is based on the Sugiyama graph-drawing algorithm accommodated in [7].

As stated above, VipTool allows to specify graphically certain properties of the business process, like specific forms of forbidden and desired behavior. Namely, the following three types of specifications are implemented (see e.g. [3]):

- Facts specify sets of forbidden markings. Facts are visualized via fact transitions.
- Goals specify two local states which have to satisfy the following property: If the first local state is reached, then the second local state will eventually also be reached. Goals are visualized using goal transitions (also known as Z-transitions, from the German word *Ziel*).
- Causal chains, which specify two transitions that are not allowed to occur causally immediately after each other. The causal chains are visualized using additional places, called common places.

4 Functionality of the VipTool: ACase Study

In this section we briefly illustrate the functionality of the VipTool by a simple case study. All figures in this section are obtained by eps-export from the VipTool.

The Petri net model of Figure 1 represents the workflow caused by a damage report in an insurance company. After the registration (transition *Registrate*) of the loss form submitted by a client, the business process divides into two

concurrent sub-processes. In the upper one, validity of the client's insurance is checked (transition *Check Insurance*), possibly further queries to the client can be answered (transitions *Ask Query* and *Process Answer*), and finally the insurance is positively or negatively evaluated (transitions *Positive Evaluation 1* and *Negative Evaluation 1*). In the lower sub-process, the damage itself is checked (transition *Check Damage*) and subsequently positively or negatively evaluated (transitions *Positive Evaluation 2* and *Negative Evaluation 2*). The two sub-processes join again. Depending on the different evaluations, the damage is payed or different sorts of refusal letters are sent to the client (transitions *Pay Damage* and *Send Letter 1-3*).

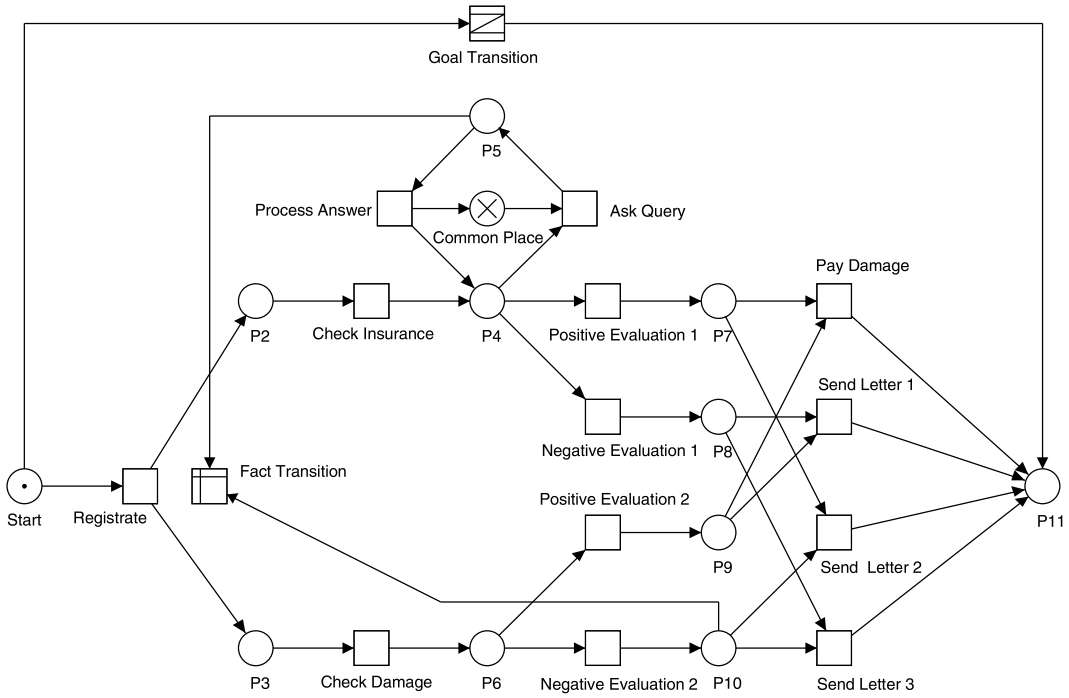

Fig. 3. The net from Figure 1 extended by graphical specifications of three different types: a fact transition, a goal transition and a common place

In Figure 3, the Petri net from Figure 1 is extended by three specifications:

- The fact transition *Fact Transition* ought to specify that we do not want to consider runs where the client is asked for further information, although the damage is negatively evaluated.
- The goal transition *Goal Transition* ought to specify that we only want to consider runs which end by the payment of the damage or the sending of one of the refusal letters. So runs which stop although further activities are possible are excluded (progress assumption) and it is assumed that eventually *Positive or Negative Evaluation* occurs (fairness assumption).

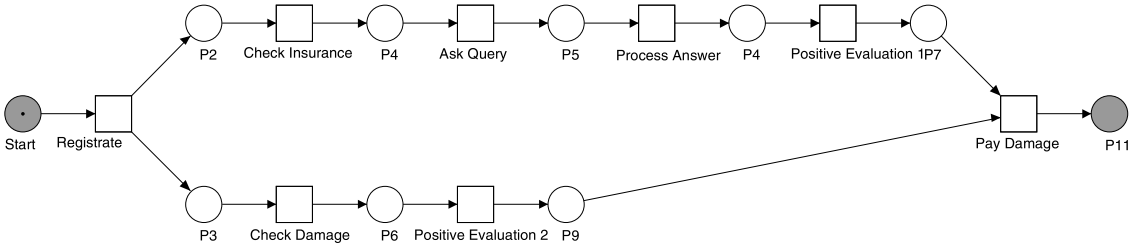

Fig. 4. A legal run of the net of Figure 3, with the places involved in the specification given by the goal transition highlighted.

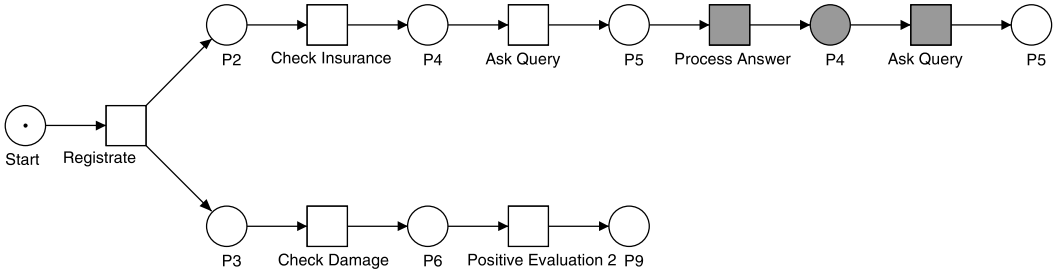

Fig. 5. An illegal run of the net in Figure 3 with the causal chain, involved by the common place, highlighted

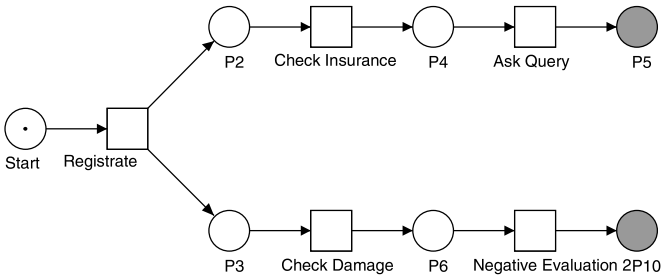

Fig. 6. A run of the net in Figure 3 which indicates that the specification given by the fact transition is wrong.

– The common place *Common Place* ought to specify that we only want to consider runs where the client is asked at most once for more information. Actually this requirement is stronger than the previous fairness assumption.

The set of runs computed by the VipEngine is divided into these runs, which fulfil the above specifications, called *legal runs*, and runs, which do not fulfil at least one of the above specifications, called *illegal runs*. Figure 4 gives an example of a legal run. The places involved by the goal transition of the above specification are highlighted by grey color. Checking the set of illegal runs, we

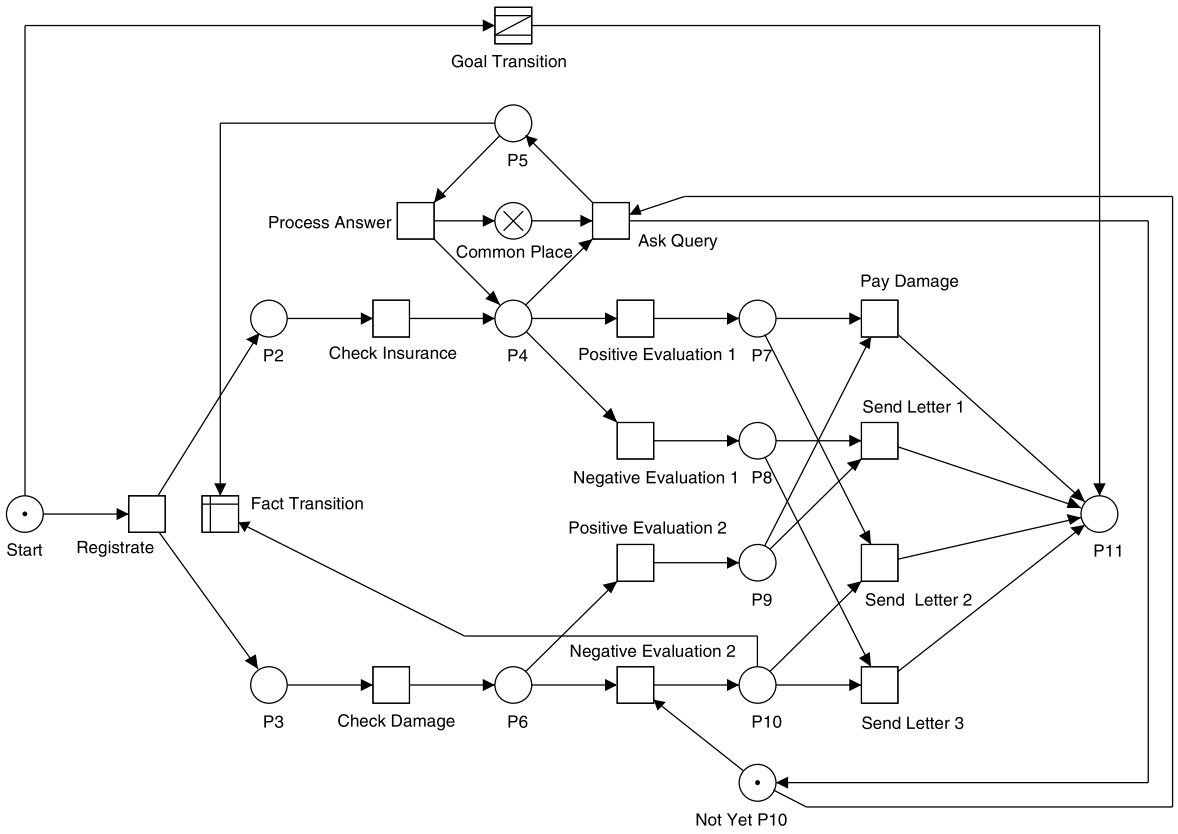

Fig. 7. Improved model of the business process. The connection of the added place *Not Yet P10* with transition *Ask Query* causes that this transition can never follow transition *Negative Evaluation 2*.

can observe whether there are desired runs which are still illegal. The Figures 5 and 6 give examples of illegal runs. In Figure 5, the computation of the run is stopped, because the specification given by the common place is violated. The causal chain (consisting of two transitions and one place) involved by the common place is highlighted by grey color.

The computation of the run in Figure 6 is aborted because the specification given by the fact transition is violated (again the involved places are highlighted). Nevertheless, we observe that a part of the behavior represented by this run should not be forbidden: Assume, the client is asked for more information (place *P5*) before the damage is negatively evaluated (place *P10*). In this case, the run should be completed by sending a refusal letter. So with this run we were able to realize that the specification given by the fact transition was badly formalized. Indeed, it is not possible to model the desired specification by one of the three implemented types of specifications. This leads to the insight that we have to change the model, to obtain the desired behavior. This is done in Figure 7. By applying again the above validation steps, one can observe now that this business process fulfils all desired requirements.

5 Conclusion

We have presented features and implementation of the new VipTool. Emphasis was on the support of step-wise design of business process models, employing validation of specifications and verification of models in each step.

The further development of VipTool includes the following tasks:

- The initial model can be obtained by folding of simpler models representing single scenarios [5]. This folding operation will be supported by the tool
- The supported business process models will be an appropriate class of high-level Petri nets (as it was the case for the original VipTool [7].
- The constructed concurrent runs will be more abstract than usual process nets, partly taking the high-level nature of tokens into account. More precisely, the concept of abstract process nets described in [4] will be implemented.
- The (experienced) user will be enabled to define further graphical specification patterns by graphically specifying classes of legal and/or illegal runs in terms of appropriate process net patterns.

Acknowledgement. We acknowledge the work of all other members of the VipTool development team: Robin Bergenthum, Thomas Liske, Thomas Loidl, Sebastian Mauser, Vesna Milijic and Niko Switek.

References

1. W.M.P. van der Aalst, J. Desel and A. Oberweis (Eds.). *Business Process Management.* Springer, LNCS 1806, 2000.
2. W.M.P. van der Aalst and K. van Hee. *Workflow Management,Models Methods and Systems.* The MIT Press, Cambridge, Massachusetts, 2002.
3. J. Desel. Validation of Process Models by Construction of Process Nets. In [1], pp. 110–128.
4. J. Desel. Model Validation - A Theoretical Issue? In J. Esparza, C. Lakos (Eds.). *Proc. of ICATPN 2002*, LNCS 2360, Springer 2002, pp. 23–43.
5. J. Desel and T. Erwin. Hybrid specifications: looking at workflows from a run-time perspective. *International Journal of Computer System Science & Engineering*, 15 Nr. 5, pp. 291–302 (2000).
6. J. Esparza, S. Römer and W. Vogler: An Improvement of McMillan's Unfolding Algorithm. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems,TACAS '96* , LNCS 1055, pp. 87–106. Berlin, Heidelberg, New York: Springer (1996).
7. T. Freytag. *Softwarevaliedierung durch Auswertung von Petrinetz-Abläufen.* Dissertation, University of Karlsruhe 2001.
8. A.-W. Scheer and M. Nüttgens. ARIS Architecture and Reference Models for Business Process Management. In [1] pp. 376–390.
9. M. Weber, E. Kindler. The Petri Net Markup Language. To appear in H. Ehrig, W. Reisig, G. Rozenberg, H. Weber (Eds.). *Petri Net Technology for Communication Based Systems.* LNCS 2472, Springer 2003.