BUILDING PARALLEL RAYTRACING SIMULATION MODEL WITH PETRI NETS AND B-METHOD

Štefan Korečko, Branislav Sobota

Technical University of Košice, Faculty of Electrical Engineering and Informatics,
041 20 Košice, Letná 9, Slovakia

stefan.korecko@tuke.sk (Štefan Korečko)

Abstract

In this paper we deal with a process of building a Coloured Petri net model of a parallel raytracing implementation, developed at the home institution of authors. The model has been used to evaluate possible parallel raytracing strategy improvements by means of simulation-based performance analysis. Here we present a basic Place-Transition net model, which captures all crucial properties of the raytracing implementation. We demonstrate how structural properties of Petri nets, namely place and transition invariants, and proof obligations of B-Method can be used to verify that the basic model has desired properties and is deadlock free. The verification is performed using available tools, including our original ones and uses original theoretical results regarding transformations between Petri nets and specifications in B-Method. Finally, we show how the CPN model can be designed on the basis of the Place-Transition net model.

Keywords: Petri nets, B-Method, deadlock freeness, simulation-based performance analysis, raytracing

Presenting Author's biography

Štefan Korečko was born on July 13, 1978. In 2001 he graduated (MSc.) with honours at the department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University (DCI FEEI TU) in Košice. He defended his PhD thesis in the field of computer devices and systems in 2006. The title of his thesis was "Integration of Petri Nets and B-Method for the mFDT Environment". Since 2004 he is working as an assistant professor at the DCI FEEI TU in Košice. His scientific research is focused on formal methods, Petri nets and B-Method in particular, their integration and use in software development and modelling and simulation.

1 Introduction

3D scene raytracing belongs to the set of very time-consuming tasks. Fortunately the task can be relatively easily decomposed and computed in a parallel environment. In principal, there are two methods of decomposing a raytracing computation: demand-driven and data-driven. Our computer graphics research group also developed a parallel raytracing solution [14], which implements the demand-driven method and uses a multicomputer (cluster) environment. For the raytracing itself, a freeware raytracer Pov-ray [18] is used.

There are two types of computational nodes in our implementation: master (one) and slave nodes. All the nodes perform raytracing jobs, but the master also manages the whole process, allocates jobs and interacts with the user. The scene raytracing proceeds as follows: First the master node sends a 3D scene data to each slave node. Then the master divides the final image area into rectangular tiles and orders each node to process (raytrace) a tile. When a slave node finishes tile raytracing, it notifies the master. Then the master downloads the raytraced tile and orders the slave to process another one. The communication with slaves has a higher priority for the master than its raytracing job. After processing of all tiles the master assembles the final image.

To improve an effectiveness of our implementation we proposed several improvements aimed at a reduction of master/slave communication delays. As we feel that it would be very time and money consuming to test intended improvements using real software and hardware, we evaluated the improvements using simulation-based performance analysis on a Coloured Petri net (CPN) model. We opted for CPN and its supporting tool, called CPN tools [7], [17], because they provide a sufficient modelling power, probability functions and time concept.

In this paper we focus on the process of building the simulation model itself. We start with a basic Place-Transition (PT) net model, which captures all crucial properties of our raytracing implementation and intended improvements. Then we verify the PT net model using structural properties of PT nets and adequate proof obligations of an equivalent model in the language of B-Method. Finally, the PT net model is "updated" to a timed CPN model, where all details of the raytracing process, including timing, are added.

2 PT net model

The Place-Transitions (PT) nets, sometimes called simply Petri nets, can be regarded as the best-known class of low-level Petri nets. They are able to naturally express non-determinism, parallelism and concurrency and offer easy to understand graphical notation and possibility of structural analysis. Basic information about Petri nets and their properties can be found in [5], [12].

The basic PT net model of our raytracing implementation is shown in Figure 1. It represents a scenario with 10 computational nodes and the image area divided into 750 tiles. Its initial marking means that a new scene is ready to be processed (a token in the place *newScene*). A number of tokens in *nodesM* and *nodesS* is a number of free master and slave nodes. A firing of *sendScene* represents a sending of the 3D scene data to each slave node. Tokens in *tiles* are scene tiles that have not yet been processed. Processing of a new tile starts by firing of *selectNewTileS* (on a slave) or *selectNewTileM* (on the master). The firing also reserves a node by removing a token from *nodesS* or *nodesM*. The fact that the master is rendering is indicated by a token in *masterRend*. Raytracing on a slave can be successful (firing of *sucRtrStartS*) or unsuccessful (*unsucRtrStart*). On the master it can be only successful (*sucRtrStartM*). Firing of *freeMrNode* finishes tile raytracing on the master - it sends the tile to *computedTiles* and releases the master node. There are two possible cases after a tile is processed on a slave node: If the master is free, then it starts downloading the computed tile (firing of *sendTileSl_Mf*). A firing of *freeSlNode_Mf* finishes the downloading. In the second case master interrupts rendering and downloads the tile (*sendTileSl_Mr*, *freeSlNode_Mr*). An unsuccessful tile processing (started by *unsucRtrStart*) is caused by a slave node failure. Then firing of *returnTile* occurs when the master notes that the slave is not responding (the response of nodes is checked regularly). After this the tile is marked as unprocessed (i.e. moves to the place *returnedTiles*) and can be raytraced again by firing of *selectRetTileM* or *selectRetTileS*. The slave node then recovers from failure by firing of *recoverNode*.

2.1 Invariant analysis

While PT nets don't offer great modelling power, they have nice analytical properties. S-invariants express invariant properties of modelled system state (net marking) and T-invariants represent transition sequences whose firings have no effect on the net marking. For the invariant analysis of the PN net model we used the TIme petri Net Analyzer (TINA) [3], [16] toolbox and our own mFDTE/PNtool [6].

From the T-invariants computed we get that there are fireable sequences, consisting of all transitions, which lead us from the initial marking back to the initial marking.

S-invariants found allowed us to verify important properties of the model. Some of equations based on them are listed in Tab. 1.

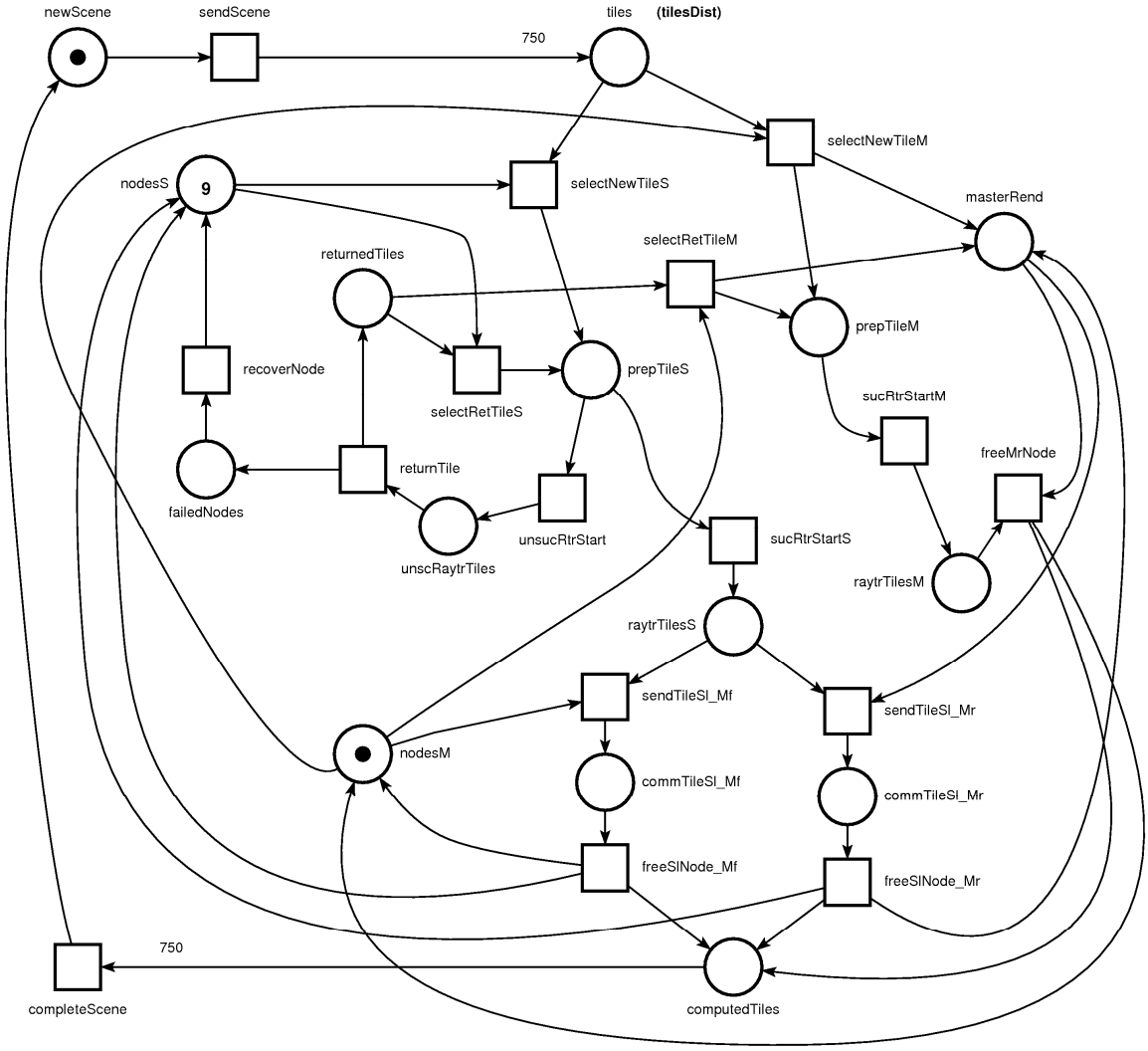

Fig. 1 Basic PT net model of parallel raytracing.

Tab. 1 Equations derived from S-invariants

I1	commTileSl_Mf + commTileSl_Mr + failedNodes + nodesS + prepTileS + raytrTilesS + unscRaytrTiles =9
I2	commTileSl_Mf + commTileSl_Mr + computedTiles + 750*newScene + prepTileM + prepTileS + raytrTilesM + raytrTilesS + returnedTiles + tiles + unscRaytrTiles = 750
I3	commTileSl_Mf + nodesM + prepTileM + raytrTilesM = 1
I4	commTileSl_Mf + commTileSl_Mr + masterRend + nodesM = 1

Names of the places in the equations stand for the markings of corresponding places. The equation *I1* proves correct distribution of slave nodes, *I3* and *I4* proves the same for the master. *I2*, among others, states that tiles are processed only after the scene data are send to slave nodes: If (marking of) *newScene=1*, marking of all other involved places is 0. It also states that while there is some tile processed (i.e. marking of some other place than *newScene* is greater than *0*), we cannot start to process a new scene (because *newScene=0*).

S-invariants obtained also provided indispensable aid for deadlock freeness proof.

2.2 Deadlock freeness proof with B-Method

The B-Method [1], with its B-Abstract Machine Notation (B-AMN) specification language, is a state-based model-oriented formal method. It offers a well-defined development process, which allows to specify a software system as a collection of so-called B-machines and to refine such an abstract specification to a concrete one. A consistency of the abstract specification and correctness of refinement are

verified by means of proof obligations (PObs). There is an industrial tool, called Atelier B [15], which supports the whole development process and includes prover for PObs. The tool is available for free.

In general, the B-machine consists of a set of state variables (clause VARIABLES), an invariant to restrict the variables (clause INVARIANT), an initial operation to establish an initial state (INITIALISATION) and a set of operations to modify the variables (OPERATIONS). There are also other clauses intended for additional assertions and data components (parameters, sets and constants). The PObs of the machine are used to prove that the initial operation establishes the invariant and operations of the machine maintain the invariant.

In [9] a theory of translations between B-AMN and Petri nets has been introduced which makes it possible to transform any PT net or Evaulative Petri net (a Turing-powerful class of Petri nets) into the computationally equivalent B-machine and almost any B-machine into the equivalent CPN. A basic idea of the translations is to link together similar behavioural concepts of both methods. Therefore places of PN are transformed to state variables of B-machine, initial marking to initialisation operation and transitions and adjacent arcs to operations.

Here we used the translation of PT net to a bisimilar B-machine to prove a deadlock freeness of our basic model. The machine, named *parRaytrBModel*, has been generated by mFDTE/PNtool and a part of it is shown in Fig. 2. To make the machine more general we replaced number of slave nodes and tiles by corresponding constants *tilesNo* and *sLnodesNo*.

Values of machine variables are naturals (N) and correspond to markings of the PT model (same names as places). Similarly, operations correspond to the transitions. The operations consists of a guarded command "SELECT P THEN S END", which means "do S, if P holds". If P doesn't hold, then the command is not feasible. Operator "||" stands for parallel composition, so "S1|| S2" means "do S1 and S2 simultaneously". To use the B-machine obtained to check a deadlock freedom of the PT model, we add a predicate saying "there must be at least one feasible operation in each state of the machine" and prove the PObs. The predicate has the form (1).

```
(newScene>=1) or (nodesM>=1 & tiles>=1)
or (nodesS>=1 & tiles>=1) or …                    (1)
or (computedTiles>=tilesNo)
```

We managed to prove the deadlock freeness in Atelier B but only after addition of other invariant conditions (Fig. 2), equivalent to the S-invariant equations for the PT model and utilizing the predicate prover of the tool.

It should be noted that we are not the only ones who came with an idea of Petri net analysis by means of B-

```
MACHINE parRaytrBModel

ABSTRACT_CONSTANTS
  tilesNo, sLnodesNo

PROPERTIES
  tilesNo:N & tilesNo>0 & sLnodesNo: N & sLnodesNo>0

VARIABLES
   newScene, nodesM, nodesS, tiles, ..., computedTiles

INVARIANT
   newScene:N & nodesM:N &   nodesS:N & tiles:N & ...
   & computedTiles:N
  & /*I1 S-invariant*/
   commTileSl_Mf + commTileSl_Mr + failedNodes +
   nodesS + prepTileS + raytrTilesS + unscRaytrTiles
   =sLnodesNo
  & /*I2 S-invariant*/
   commTileSl_Mf + commTileSl_Mr + computedTiles +
   newScene*tilesNo +… + unscRaytrTiles = tilesNo
  & /*I3 S-invariant*/
   commTileSl_Mf + nodesM + prepTileM + raytrTilesM=1
  & /*I4 S-invariant*/
   commTileSl_Mf + commTileSl_Mr + masterRend +
   nodesM = 1
  & /*I5 S-invariant*/
   commTileSl_Mf + commTileSl_Mr*2 + computedTiles +
   masterRend+newScene*tilesNo+prepTileS+raytrTilesS +
   returnedTiles + tiles + unscRaytrTiles = tilesNo
  &  /*Deadlock freeness condition */
   (newScene>=1) or (nodesM>=1 & tiles>=1) or
   (nodesS>=1 & tiles>=1) or…
   or (computedTiles>=tilesNo))

INITIALISATION
  newScene:=1 || nodesM:=1 || nodesS:=sLnodesNo ||
  tiles:=0 || ... || computedTiles:=0

OPERATIONS
op_sendScene=
      SELECT newScene>=1 THEN
       newScene:=newScene - 1 || tiles:=tiles + tilesNo
      END;
op_selectNewTileM=
      SELECT nodesM>=1 & tiles>=1 THEN
       masterRend:=masterRend + 1 || nodesM:=nodesM
       - 1 || prepTileM:=prepTileM + 1 || tiles:=tiles - 1
      END;
op_selectNewTileS=
      SELECT nodesS>=1 & tiles>=1 THEN
       nodesS:=nodesS - 1 || prepTileS:=prepTileS + 1 ||
        tiles:=tiles - 1
      END;
…
op_completeScene=
      SELECT computedTiles>=tilesNo THEN
       computedTiles:=computedTiles - tilesNo ||
       newScene:=newScene + 1
      END;
END
```

Fig. 2 Fragment of B-machine generated from the basic PT net model, with extended invariant clause.

Method. There is also another approach [2], where not only the system modelled but also the whole definition of Petri net is captured in B-Method. We opted for a more natural translation, where a link between PT net and B-AMN models remains clear and it is, for example, easy to import S-invariant equations from PT net to a corresponding B-Machine. There is also no evidence that the transformation proposed in [2] has been implemented.

A similar deadlock freeness condition is also included in Event-B, an evolution of B-Method, as one of the proof obligations [11].

3 Building up CPN model

After verification of the basic model its "transformation" to CPN can be performed. The CPN model is shown in Fig. 3 and its detailed description can be found in the paper [10], available online. The transformation has been done by hand and we tried to keep it as "conservative" as possible to retain the properties proved for the basic net. For the most of the model we just transformed undistinguishable tokens of the basic net to structured ones. These structured tokens carry all the information needed for precise simulation, such as complexity of a tile and type of

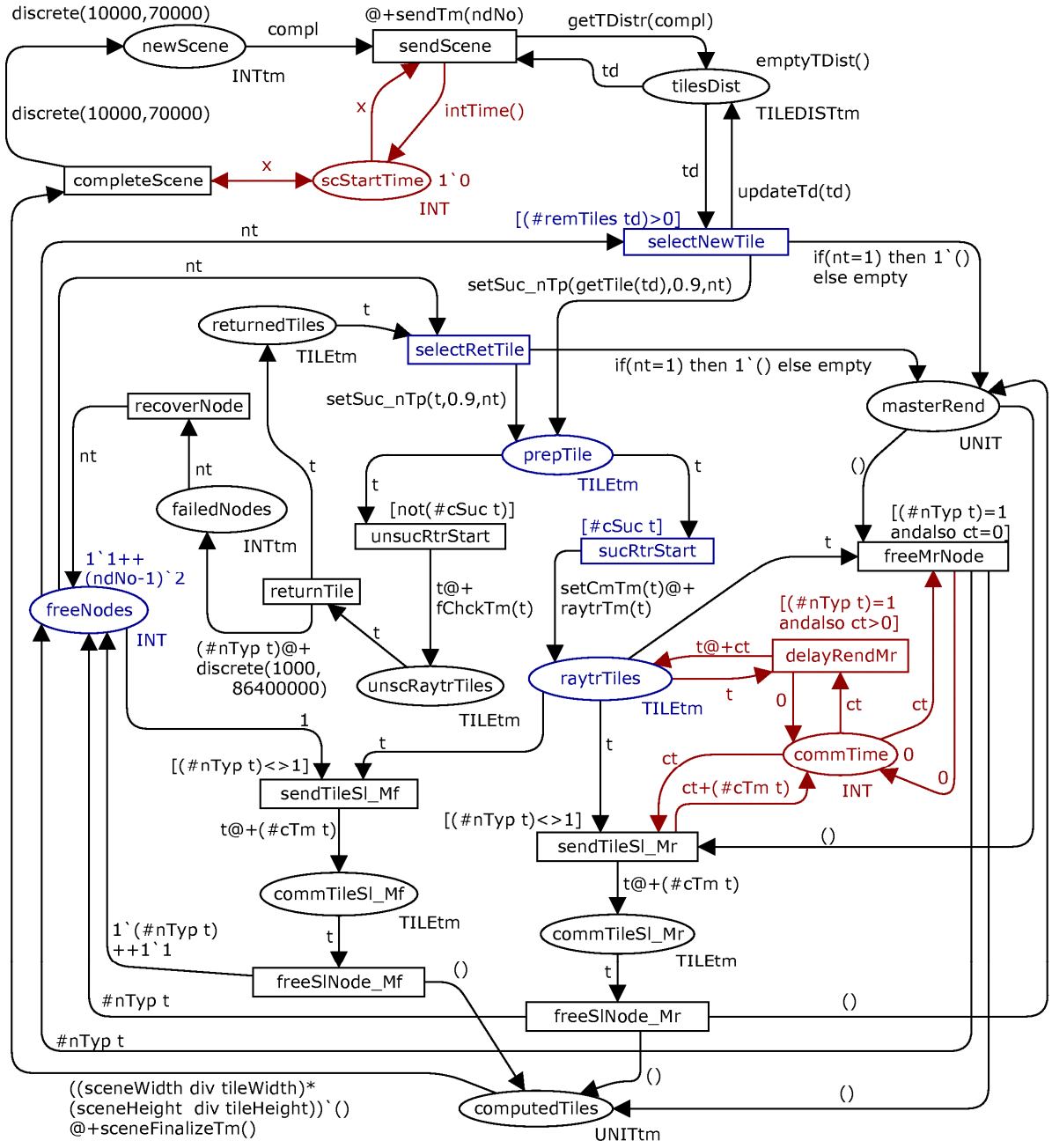

Fig. 3 Timed CPN model of parallel raytracing.

the node where the tile will be raytraced, raytracing job duration and communication delays. The corresponding arc expressions have been added to compute data carried by tokens. High-level nature of CPN also allowed a partial folding and thus simplification of the net structure: all places and transitions of the net that differ only in suffix (*S* or *M*) merged to one node (i.e. *sucRtrStartS* and *sucRtrStartM* to *sucRtrStart* or *nodesM* and *nodesS* to *freeNodes*). Of course, this folding resulted in addition of new arc expressions and transition guards. Folded places and transitions are rendered in blue in Fig 3.

We illustrate these changes on a part responsible for the start of tile raytracing: In the basic model the choice between successful and unsuccessful raytracing start is simply non-deterministic. But in the CPN model the choice depends on the value of the field *cSuc* of a data structure carried by a token *t* that represents a tile to be raytraced. A value of *cSuc* is set by the function *setSuc_nTp* when the tile is selected. Because the token data structure also carries information about type of the node where the tile will be processed, we no longer need separate places and transitions for slave and master (*setSuc_nTp* ensures that *cSuc* will be true in the case o master node). We can say that the transformation to CPN limited the non-determinism by taking simulation-related details into consideration, but the "token flow" is essentially the same. A number of not-yet raytraced tiles is also represented differently: In the PT net it is simply the marking of *tiles* (*tilesDist*). In the CPN it is the value of the field *remTiles* of a record-type token *td*, stored in *tilesDist*. The token *td* also carries information about a distribution of the scene complexity among tiles.

We can also see two new parts (rendered in brown) added to the CPN model:

1. *delayRendMr*, *commTime* and adjacent arcs: This part ensures that delay, caused by a communication with slaves, will be added to a raytracing job duration on the master node.
2. *scStartTime* and adjacent arcs: The place *scStartTime* stores the time of raytracing job start. The time stored is then subtracted from the simulation time when *completeScene* fires and stored to a text file for further processing.

The CPN model has been used to evaluate possible improvements of our current parallel raytracing implementation. The improvements aimed at reduction of the communication delay by adding some memory buffers to the master or slave nodes. Obtained simulation results, presented in [10], shown that the improvements are not worth of an implementation. But to be sure in this case, we plan to perform other "large-scale" simulation experiments, utilizing our newly developed simulation automation tool for CPN and CPN tools.

4 Conclusion

In this paper we demonstrated how Petri nets and B-Method can be used to design and verify a parallel raytracing strategy model. We shown the verification of a PT net model that served as a basis for building a high-level timed CPN model, used for simulation-based performance analysis.

While the verification of the PT net model has been performed using automated tools and formal theory, the transformation, or refinement, to the CPN model has been done by hand. Therefore we intend to prepare a tool which will partially automate the transformation by utilizing known approaches to transformation of ordinary Petri net to CPN (including folding) [4] and refinement of Petri nets [13].

We also intent to extend our Petri net to B-machine translation to CPN. This can be done by adopting a method similar to that of [8], where markings are translated to set variables and transition guards and expressions on adjacent arcs to preconditions and generalised substitutions of corresponding B-machine operations. Here a translation of B-AMN expressions to CPN expressions, defined as a part of [9] can be helpful.

The work presented is supported by VEGA grant project No. 1/0646/09: "Tasks solution for large graphical data processing in the environment of parallel, distributed and network computer systems".

5 References

[1] J.R. Abrial. *The B-book: assigning programs to meanings*. Cambridge University Press, 1996.

[2] J. Ch. Attiogbé. Semantic Embedding of Petri Nets into Event-B. Technical Report. LINA - FRE CNRS. http://arxiv.org/abs/cs/0510073

[3] B. Berthomieu, F. Vernadat. Time Petri Nets Analysis with TINA. In: *Proc. of the Third International Conference on the Quantitative Evaluation of Systems*, pages 123-124, 2006.

[4] H. Darabi, M.A. Jafari. A zero-one programming of Petri nets to colored Petri net transformation. In: *Proc. of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, pages 25-31, Troy, NY, USA, October 1994.

[5] J. Desel, W. Reisig. Place/Transition Petri Nets. In: G. Goos, J. Hartmanis, J. van Leeuwen. (editors) *Advances in Petri Nets*. LNCS, vol. 1491, pages 122-173. Springer, Heidelberg, 1997.

[6] Š. Hudák, Š. Korečko and S. Šimoňák. ¸A Support Tool for the Reachability and Other Petri Nets-Related Problems and Formal Design and Analysis of Discrete Systems. *Problems in Programming*, 20, 2-3:613-621, 2008.

[7] K. Jensen, L.M. Kristensen and L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer*, 9, 3-4:213-254, 2007.

[8] L.A. Kalinichenko, S.A. Stupnikov, N.A. Zemtsov. Extensible Canonical Process Model Synthesis Applying Formal Interpretation. In *Proc. of the East-European Conference ADBIS'05*, pages 183-198, Talin, Estonia, September 12-15 2005,.

[9] Š. Korečko. Integration of Petri Nets and B-Method for the mFDT Environment. PhD thesis. DCI FEEI TU Košice, Slovakia, 2006 (in Slovak).

[10] Š. Korečko, B. Sobota and R. Janošo. Evaluation of Parallel Raytracing Strategy Improvements by Petri Nets. *Journal of Computer Science and Control Systems*, 3, 1:87-92, 2010. http://electroinf.uoradea.ro/reviste%20CSCS/volumes/volumes.htm

[11] C. Métayer, J.R. Abrial, L. Voisin. Event-B Language. RODIN Deliverable 3.2, 2005. rodin.cs.ncl.ac.uk/deliverables/D7.pdf

[12] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77, 4: 541-580, 1989.

[13] J. Padberg, M. Urbášek. Rule-Based Refinement of Petri Nets: A Survey, LNCS vol. 2472, pages 161-196, Springer Berlin-Heidelberg, 2003.

[14] B. Sobota, J. Perháč, Cs. Szabó and Š. Schrötter. High-resolution visualisation in cluster environment, In *Proc. of Grid Computing for Complex Problem 2008 - GCCP 2008*, pages 62-69, Bratislava, October 27-29 2008, Institute of Informatics of Slovak Academy of Sciences.

[15] http://www.atelierb.eu

[16] http://homepages.laas.fr/bernard/tina/

[17] http://wiki.daimi.au.dk/cpntools

[18] http://www.povray.org/