A Simulator for high level Petri Nets: Model based design and implementation

Mindaugas Laganeckas

Kongens Lyngby 2012
IMM-M.Sc.-2012-70

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-M.Sc.: ISSN 0909-3192

Contents

Chapter 1

Introduction

Petri Nets are a powerful mathematical and graphical notation for modeling, analyzing and designing a wide range of discrete-event systems. Traditionally, Petri Nets are distinguished into two major kinds - low level Petri Nets and high level Petri Nets (HLPNs) also known as Colored Petri Nets (CPNs) [5]. There are many differences between the two types of nets such as support of arc inscriptions, transition guards [12]. All these features make HLPNs much more concise then low level Petri Nets. Still the main difference between HLPNs and low level Petri nets is that HLPNs have colored tokens. Each color of a token represents different data object in a model, e.g. a fast or a slow train. Whereas in a case of low level Petri Nets, all (black) tokens correspond to the same data object, e.g. all trains have the same speed. A rich feature set of HLPNs helps Petri Net experts to model a wide variety of complex systems, but it comes at a cost - all of it has to be supported by the Petri Net simulator.

In this Master project we design and implement a Simulator for high level Petri Nets. Our Simulator is similar to one which is already available and well known on the market - "CPN tools" [13]. However, the main difference between our tool and already existing tools is that our Simulator conforms to both ISO/IEC standards [3] and [4]. To our knowledge currently there is no such HLPNs simulator fully supporting both above mentioned standards.

The major part of our project is design and implementation of an effective

algorithm to find legal bindings of transitions which are involved in the transition firing rule (see Legal transition binding management). In the following paragraphs we will list the most challenging issues we faced in this part of the project. After giving a short description of each issue we will briefly explain our solution.

First of all, if a HLPN model has thousands of places and transitions then a computation time needed to find all legal variable bindings can increase significantly. This total time increase directly depends on the running time of the algorithm. Therefore, the transition binding algorithm has to be efficient (fast) since it has to be capable of dealing with Petri Nets where a number of transitions and places is huge. In our solution, the transition binding algorithm running time is $nlog(n)$, where n is a number of incoming arcs. A detailed description of the algorithm will be given in ???.

Secondly, if a Petri Net expert wants to use data types and operations in a HLPN model which are not defined in the standard then there has to be a mechanism to easily plug in new extensions to a simulator. Furthermore, not all data types and operations defined in the standard are currently supported by our Simulator. Therefore, we provide a general infrastructure (extension points) to plug in new or missing data types and operations easily. A complete explanation of already supported data types and operations as well as a mechanism how to plug in new extensions will be given in ???.

As it was mentioned previously, not all data types and operations defined in the standard are currently supported by our Simulator. Furthermore, when a user defines his own data type or an operation due to e.g. a spelling mistake it will not be recognized by our Simulator. In order to avoid such failures during runtime and to warn a user about currently not supported functionality we provide a validation mechanism. Each Petri net which is a target for a simulation is annotated with a special tag. Thus during the validation of the model all functions defined in it are checked if they are supported by the Simulator (see ???).

Another problem which we try to address in our work is variables which cannot be resolved in the transition binding algorithm. In a such case enabled transitions cannot be found and the whole simulation stops. Our proposal to this problem is to ask a user to input a sufficient part of the solution manually. More details of this mechanism will be given in ???.

Finally, the last general problem which we address in this part of our work is a transition firing strategy. Once enabled transitions are found one needs to decide which of them to fire. This decision is completely dependent on the chosen transition firing strategy. This strategy itself depends on the behavior of the parties involved in the model - the existence or absence of their spontaneous activity. In other words, the transition firing strategy is HLPN model dependent. Let us explain our claim by giving two examples. First our example comes from "Minimal distance" algorithm [6]. The algorithm finds the minimal distance of every agent to some distinguished agents in some communication network. It

is quite clear that no matter in which order the agents send messages to (or receive them from) their neighbors - the final result will always be the same for the same network of agents. Since no randomness is involved in this model that implies all enabled transitions can be fired immediately. Now let us take "Consensus in Networks" algorithm [14] as our another example. In this algorithm a set of networked sites try to reach a consensus by spontaneously broadcasting their proposal to their neighbors. The message content as well as the criteria for a site to accept or refuse an offer are not relevant for the algorithm. As a consequence, this acceptance or rejection of a proposal can be easily expressed in a probabilistic manner. Therefore, again next to already supported transition firing strategies we defined a mechanism for users to plug in their own implementations (see ???).

The second important part of the Simulator is a model for HLPNs runtime behavior. By using this model we can clearly distinguish initial and runtime place marking. It is a runtime marking which gets updated in each simulation step. Accordingly, each simulation step corresponds to a concrete runtime marking. Our challenge here is to preserve this relationship between simulation step and runtime marking in an efficient structure with respect to memory usage. As a consequence, this structure enables easy stepping backwards in the simulation history. Moreover, this kind of structure is by definition suitable for state space generation in the future. Currently, our Simulator does not support any kind of state space generation mechanism. More about the model for the Simulator runtime behavior can be found in ???.

The final part of our project is a user friendly graphical user interface for the Simulator. In our solution, the GUI is built on top of ePNK [7] - a model based graphical Petri Net editor providing functionality to create user defined Petri Net extensions. The GUI is a point where a user can actually interact with the simulation. Our Simulator has a list of all supported actions. These actions can be easily found in the application menu provided by ePNK. The supported actions are similar to other simulators from other domains. One of the most important (and time consuming) actions is a support of an interactive (single step) simulation where an expert controls the simulation completely. Usually this mode is good for model debugging purposes. In comparison an automatic simulation mode gives no control to the expert but also does not require his attention. In any case, it is easy for a user to switch from one transition firing strategy to another i.e. from manual to automatic and vice versa. Furthermore, the GUI supports many other features such as active/inactive transition indication, firing mode can be chosen from a pop up menu by clicking on an active transition, runtime information is depicted next to each place. All these features are defined in our presentation management mechanism. Thus any other application can reuse presentation management mechanism with default values or extend and adapt it to its own needs. More about presentation management

framework and its currently supported feature set can be found in ???.

The Simulator comes with two extensions which can serve as examples for future projects.
The first extension deals with a class of network algorithms, in particular "Echo" [10], "Consensus in Networks" [14] and "Minimal distance" [6] algorithms. We will give a brief introduction to each of these algorithms in later chapters. What is important now is that all these Petri nets are not algebraic nets but algebraic net schemes instead [9]. The above given Petri nets are algebraic net schemes since the structure of the network (a number of nodes and a way they are connected) is not directly used in the Petri nets. In other words, a concrete network scheme is an input for these algorithms. For this reason, we made an easy configuration of net schemes in our Simulator extension. To put it simply, we prepared tools to easily create networks and then input them to our first extension. A complete description of this extension can be found in ???.
The second extension was made to help experts where "'playing the token-game' is not enough for understanding the behavior of a complex system". This concept was already presented in [8] where PNVis - a 3D visualization of low level Petri Nets - was introduced. In comparison to PNVis our extension of the Simulator supports high level Petri Nets. And what is most important, this support comes "out of the box" i.e. one does not need to extend the existing high level Petri Net to make it work with the 3D visualization engine. For our project we used an already existing 3D engine [15]. A detailed description of this example can be found in ???.

All software in this work is implemented in JAVA programming language, using Eclipse platform [1] as an underlying infrastructure. Eclipse Modeling Framework (EMF) [11], Eclipse Graphical Modeling Framework (GMF) [2] and ePNK [7] is used in the modeling and GUI part of the project.

A more detailed description of each feature of the Simulator will be given in the following chapters.

Chapter 2

Legal transition binding management

In this chapter we will discuss the major part of our project - legal transition binding management. First we will explain our algorithm to find legal transition bindings (see Finding legal transition bindings).

2.1 Finding legal transition bindings

In this section we will explain in details our transition binding algorithm. The algorithm starts with handling initial marking (see Evaluating initial marking). After initial values are ready a unification (needs citing!!!) process takes place (see Unification). After unification is completed undefined variables are resolved (see Resolve). Finally, the solution is checked if it conforms to a variable definition (see Checking solution).

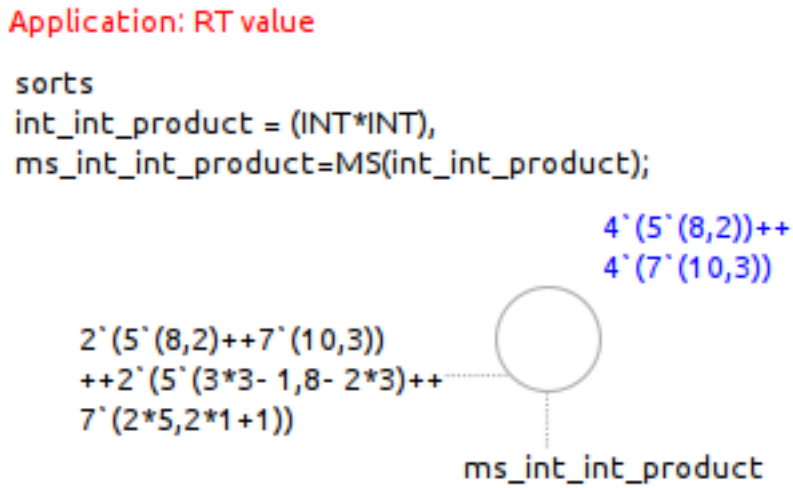

Figure 2.1: Evaluating initial marking value

2.1.1 Evaluating initial marking

2.1.2 Unification

Comparison is performed only on NumberOf pairs participating in multiset Add operation. In a case of subtraction only minuend is considered (subtrahend is ignored). Arc inscriptions are handled independently.

2.1.3 Resolve

2.1.4 Checking solution

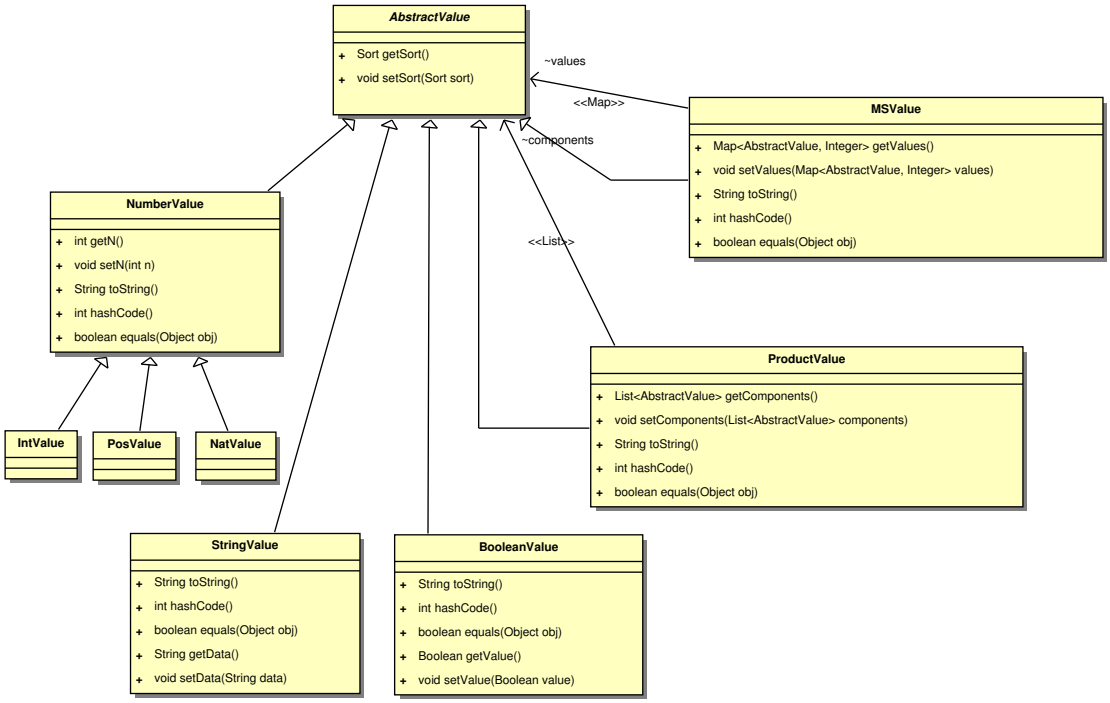

Figure 2.2: Runtime value class diagram

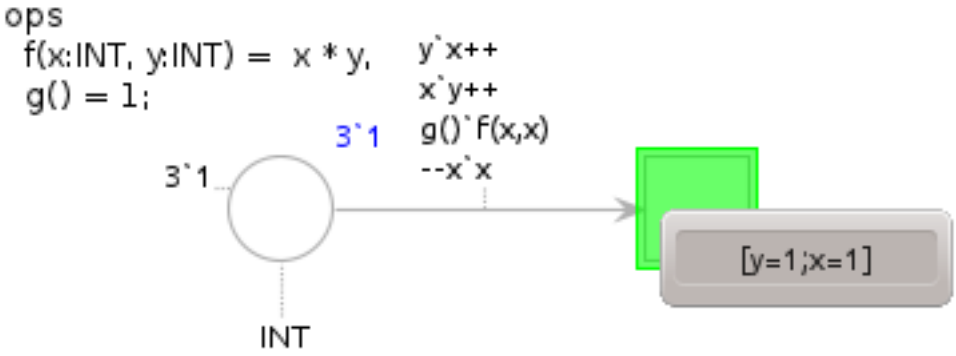

Figure 2.3: Unification: functions and multi set subtraction

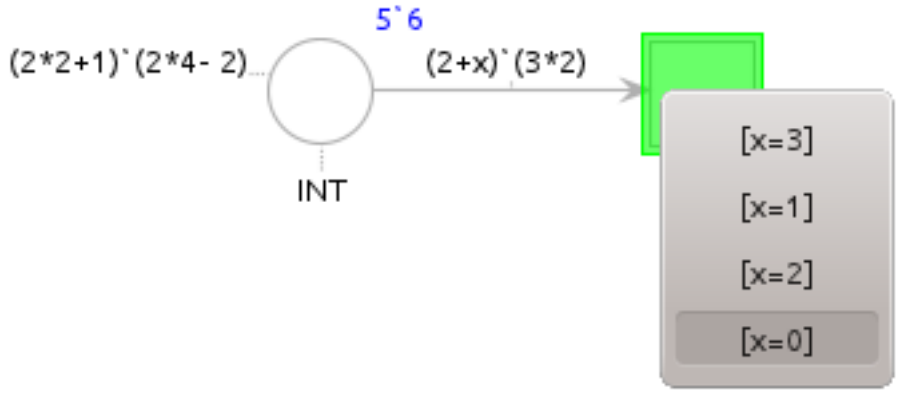

Figure 2.4: Unification: expression assignment

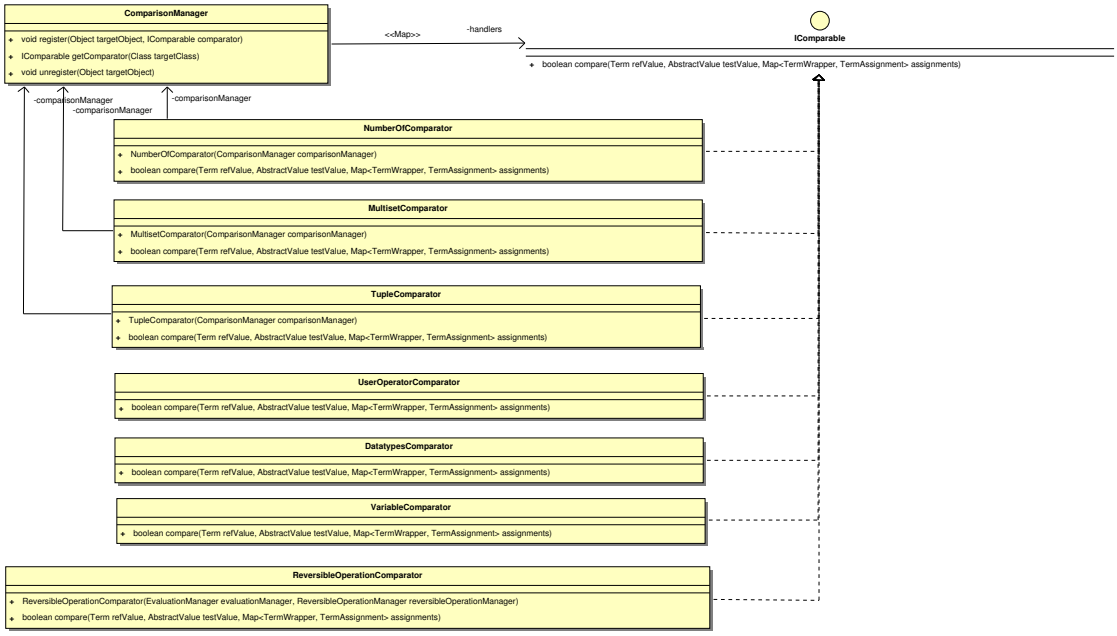

Figure 2.5: Unification: comparators

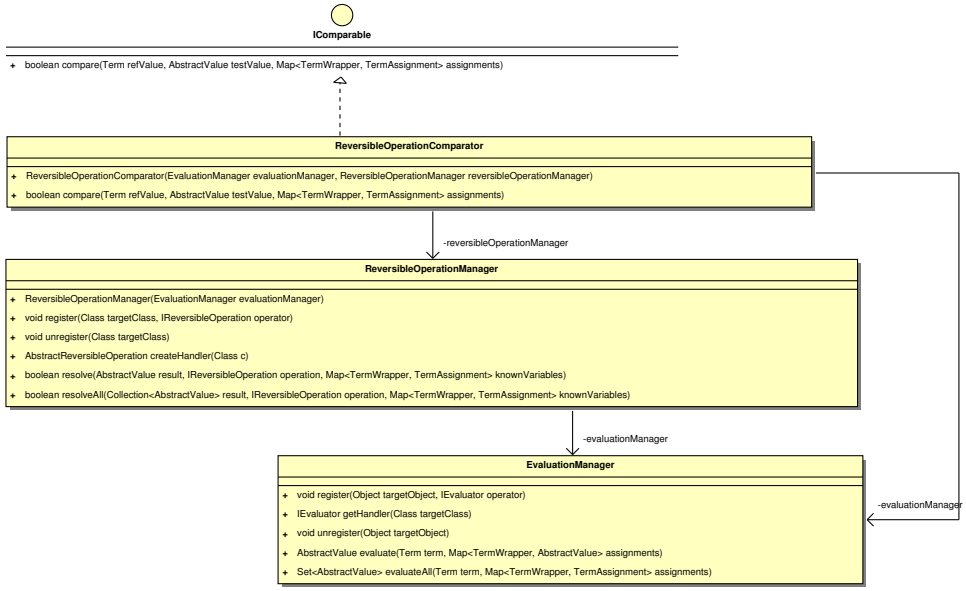

Figure 2.6: Unification: reversible operation comparator

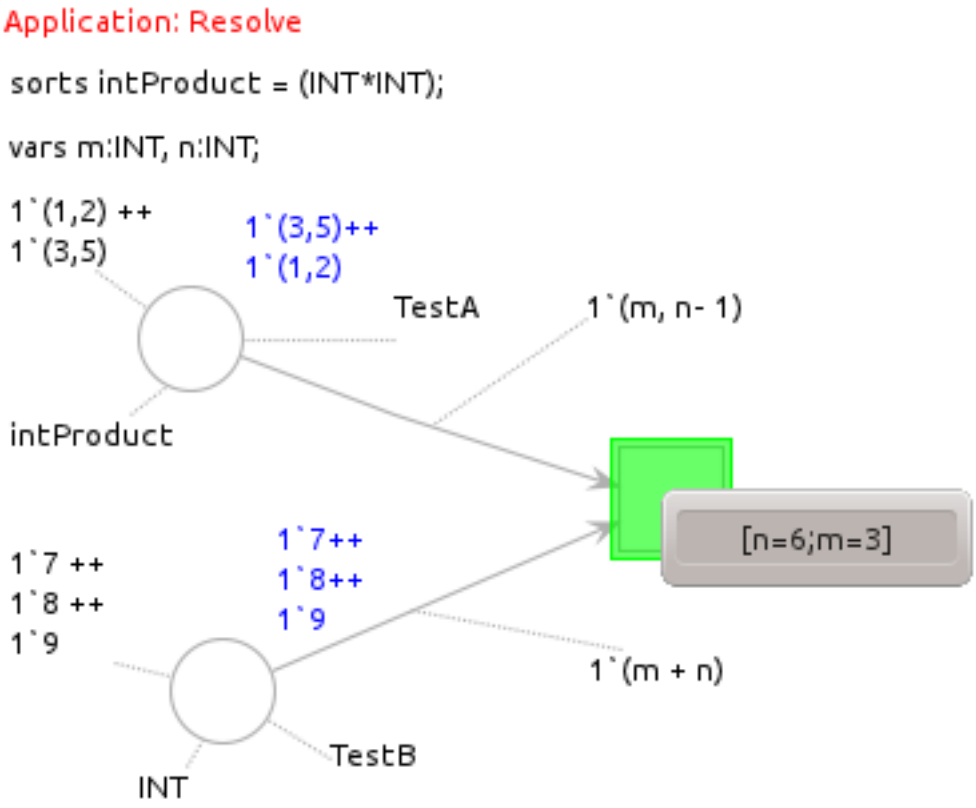

Figure 2.7: Resolve

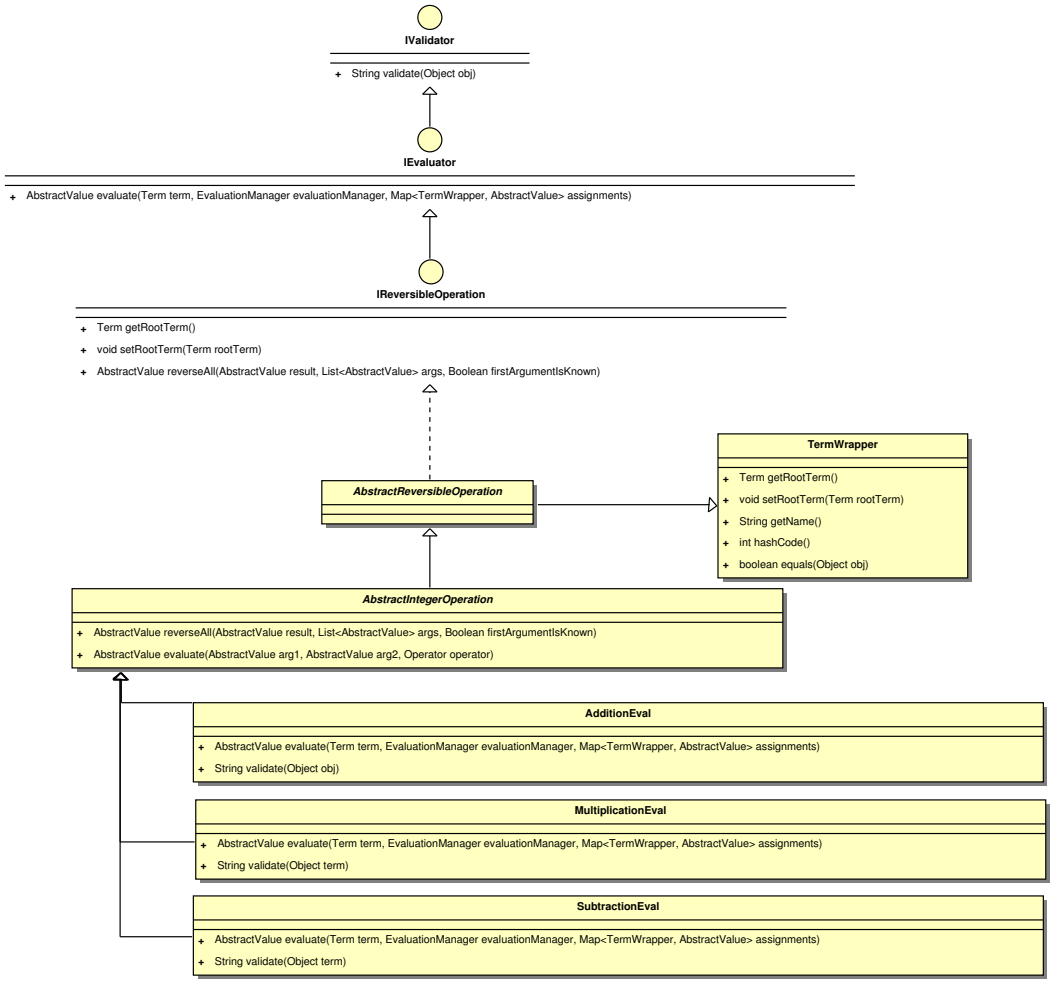

Figure 2.8: Unification: reversible operations

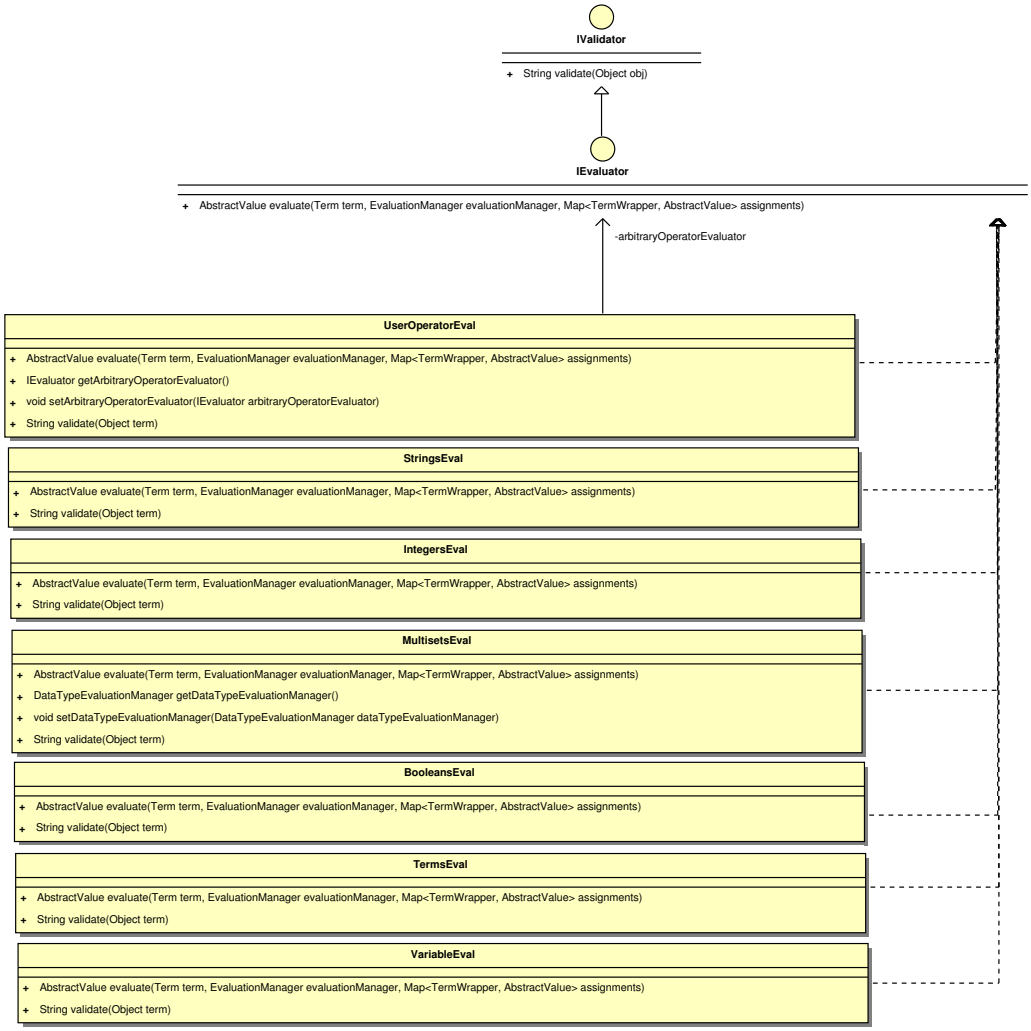

Figure 2.9: Unification: evaluators

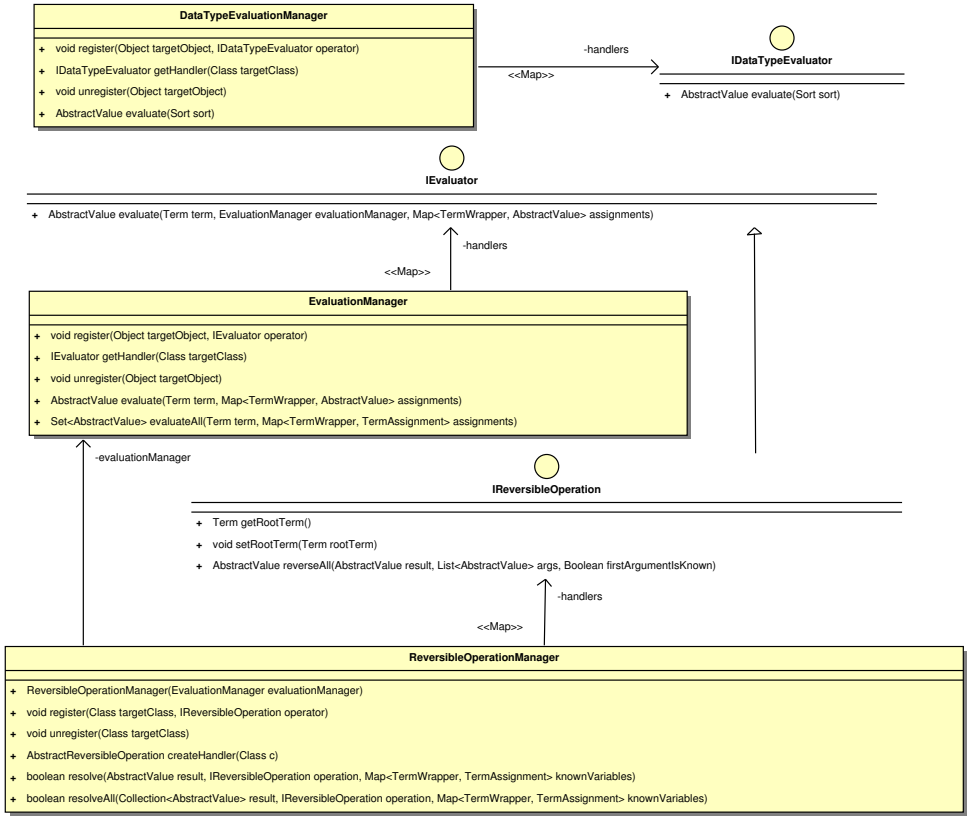

Figure 2.10: Unification: evaluation managers

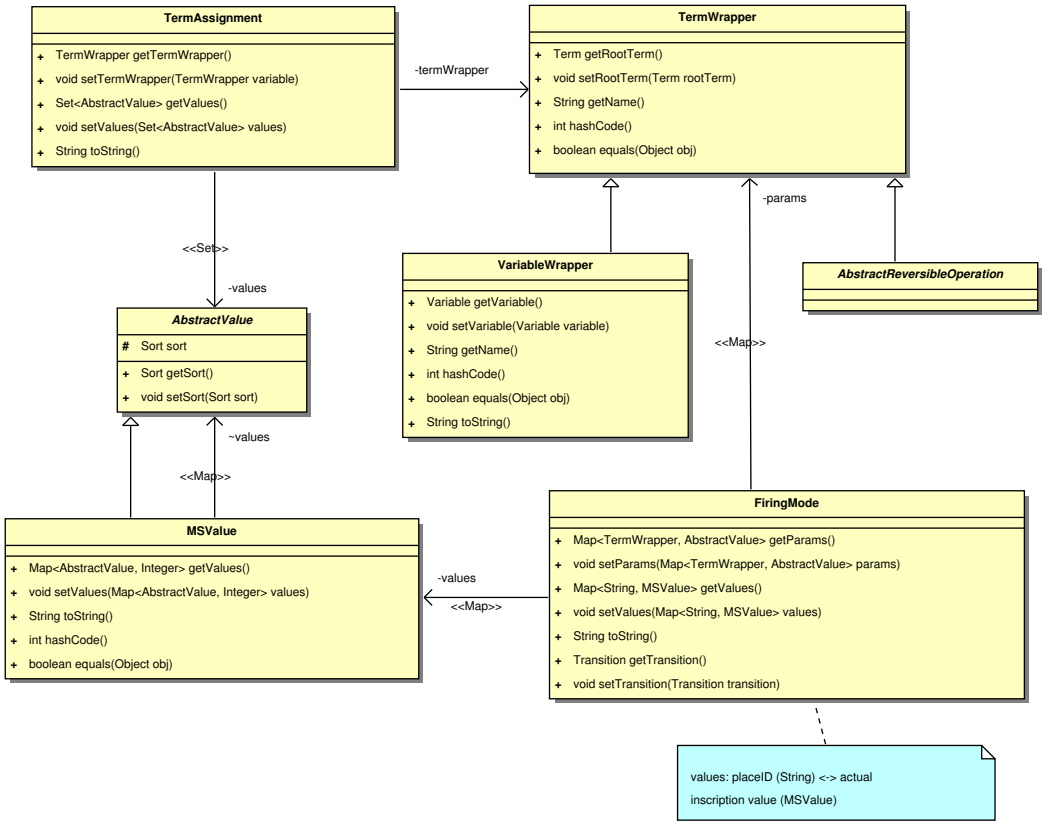

Figure 2.11: Term wrapper

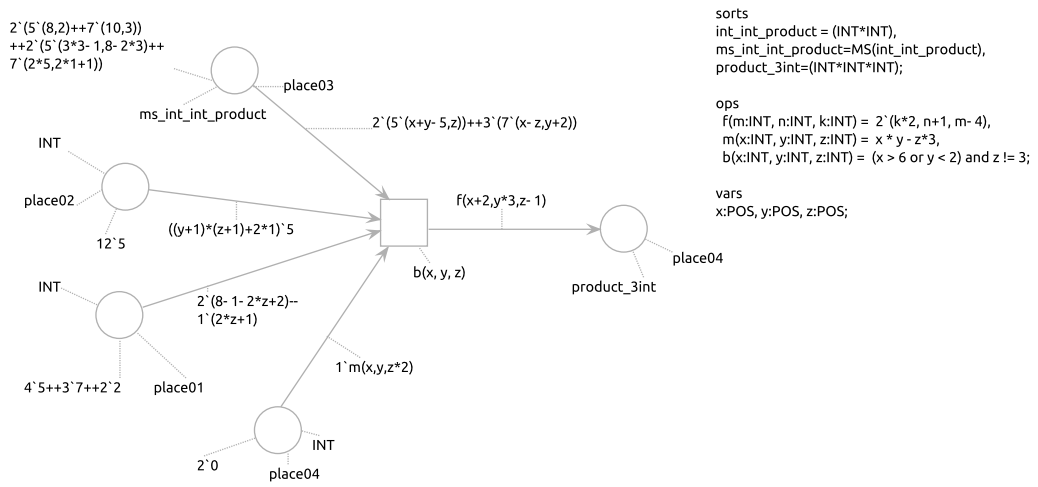

Figure 2.12: A complicated technical example

Bibliography

[1] Eric Clayberg and Dan Rubel. *Eclipse: Building Commercial-Quality Plug-Ins*. Addison-Wesley Professional, 2004.

[2] Richard C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolk*. Addison-Wesley Professional, 2009.

[3] ISO/IEC. Software and system engineering - High-level Petri nets - Part 1: Concepts, definitions and graphical notation. 15909(1), 2004.

[4] ISO/IEC. Systems and software engineering - High-level Petri nets - Part 2: Transfer format. 15909(2), 2011.

[5] K. Jensen and L. M. Kristensen. *Coloured Petri Nets*. Springer, 2009.

[6] E. Kindler and L. Petrucci. A framework for the definition of variants of high-level Petri nets. *Proceedings of the Tenth Workshop and Tutorial on Practical Use of Coloured Petri Nets and CPN Tools (CPN '09)*, 2009.

[7] Ekkart Kindler. ePNK: A generic PNML tool - Users' and Developers' Guide : version 0.9.1. *IMM-Technical Report*, 3, 2011.

[8] Ekkart Kindler and Csaba Páles. 3D-Visualization of Petri Net Models: Concept and realization. In *In Proc. of ICATPN 2004, volume 3099 of LNCS*, pages 464–473. Springer-Verlag, 2004.

[9] Ekkart Kindler and Wolfgang Reisig. Algebraic system nets for modelling distributed algorithms. *Petri Net Newsletter*, 51:51–16, 1996.

[10] Ekkart Kindler, Wolfgang Reisig, Hagen Völzer, and Rolf Walter. Petri net based verification of distributed algorithms: An example. *Formal Aspects of Computing*, 9, 1996.

[11] Dave Steinberg; Frank Budinsky; Marcelo Paternostro; Ed Merks. *EMF: Eclipse Modeling Framework, Second Edition.* Addison-Wesley Professional, 2008.

[12] Vasilis Gerogiannis;Achilles Kameas;Panagiotis Pintelas. Comparative study and categorization of high-level Petri nets. *Systems and Software(JSS)*, 43:133–160, 1998.

[13] Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank, Martin Stig Stissing, Michael Westergaard, Søren Christensen, and Kurt Jensen. CPN Tools for editing, simulating, and analysing coloured Petri nets. In *Applications and Theory of Petri Nets 2003: 24th International Conference, ICATPN 2003*, pages 450–462. Springer Verlag, 2003.

[14] W. Reisig. *Elements of Distributed Algorithms.* Springer, 1998.

[15] Morten Valvik, Du Nguyen, Félix Manuel Rubio Gallego-Preciados, Jesper Jepsen, Mindaugas Laganeckas, Radu Calin Gatej, Johannes Rasmussen, Magnus Felix Tryggvason, and Christian Ejdal Sjøgreen. Petri Net 3D Visualizer. *IMM-Technical Report*, 2011.