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

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 divided into low level Petri Nets and high level Petri Nets (HLPNs) also known as Colored Petri Nets (CPNs) [6]. Colored Petri Nets are much more concise then low level Petri Nets. There are many differences between the two nets. Still the main difference lies in a fact that HLPNs have colored tokens i.e. each color of a token represents different data type where in a case of low level Petri Nets, all (black) tokens correspond to the same data type. 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" [11]. Still the main difference between our tool and already existing tools is that our Simulator conforms to both ISO/IEC standards [1] and [2]. To our knowledge currently there is no such HLPNs simulator supporting both above mentioned standards.

The major part of our project is a design and implementation of an effective algorithm to find legal bindings of transitions which are involved in the transition firing rule. On one hand the transition binding algorithm has to be fast since it has to be capable of dealing with Petri Nets where a number of transitions and places is huge. On the other hand, it has to provide a general infrastructure where new data types and operations can be plugged in easily. Our Simulator has clearly defined extension points where users could plug in their own extensions. A part of the data types and operations which are described in the standard is already supported by the Simulator. Furthermore, another general problem which we address in this work is a transition firing rule. Once enabled transitions are found one needs to know which of them to fire. By experimenting with different Petri Net algorithms it became clear that a transition firing rule is an application dependent. This dependency is reflected in the behavior of the participating parties, the existence or absence of their spontaneous activity. For example, in ”Minimal distance” algorithm [7] all enabled transitions can be fired immediately: each agent is reporting its distance from the closest root agent to its neighbors. It is quite clear that no randomness is involved in this algorithm - the final result will always be the same no matter in which order the agents send a message to their neighbors. Let us take ”Consensus in Networks” algorithm [12] 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. Each neighbor can accept or reject the offer. It is obvious that acceptance or rejection of a proposal can be easily expressed in a probabilistic manner. Based on two examples which were given above we defined an extension point for users to plug in their own implementations of firing rule algorithm or the predefined methods can be used instead. Finally, sometimes the enabled transitions cannot be found due to a number of variables which cannot be resolved. In such cases a user is asked to input a sufficient part of the solution manually in our Simulator in order to continue the simulation.

The second important part of the Simulator is a model for HLPNs runtime behavior. Two main issues were addressed during the design and implementation of a model for HLPNs runtime behavior. The first challenge was to make our model suitable for supporting state space generation in the future. The second, our model has to be efficient with respect to memory usage.

Finally, a user friendly graphical user interface for the Simulator concludes the project. The GUI is built on top of ePNK [8] - a model based graphical Petri Net editor providing functionality to create user defined Petri Net extensions. The Simulator can be run in interactive (single step) mode where an expert can control a simulation completely. Sometimes controlling a simulation in a single step mode can be a time consuming task. Thus the Simulator can be run in automatic mode as well with a chosen firing rule. Furthermore, the GUI supports active/inactive transition indication. Firing mode can be chosen from a pop up menu by clicking on an active transition. Runtime information is depicted at top-right corner of each place using different color with respect to initial marking.

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” [12] and ”Minimal distance” [7] algorithms. To run these algorithms an input graph representing a network is needed where nodes correspond to participating parties such as agents, sites etc. and edges reflect the the general structure of the network. The second extension was inspired by a 3D visualization of Petri Net models described in [9]. The idea behind 3D visualization is that sometimes "‘playing the token-game‘ is not enough for understanding the behavior of a complex system". As an example a physics law about rigid objects can be considered: an abstraction of a token can easily hide the fact no two rigid objects can be at the same place in time. Furthermore, each moving physical object has a finite speed. On the other hand, when a transition fires a token from one place to another is transfered immediately. All these problems are addressed in an extension of the Simulator called Visual Simulator. The Visual Simulator controls a simple 3D train traffic example using existing 3D visualization engine [1]. We use an asynchronous communication between the Visual Simulator and the 3D engine to reflect the speed difference problem mentioned above. 3D engine comes with a support for collision detection which helps to detect rigid object problem.

All software in this work is implemented in JAVA programming language, using Eclipse platform [3] as an underlying infrastructure. Eclipse Modeling Framework (EMF) [4] and Eclipse Graphical Modeling Framework (GMF) [5] is used in the modeling and GUI part of the project.

To sum up, in this wolrk we present a highly extensible simulator for high level Petri Nets conforming to both ISO/IEC standards [1] and [2]. A more thorough description of each feature of the Simulator will be given in the following chapters.

[1]The 3D visualization engine was developed in the course 02162 Software Engineering 2 by group F, 2011

Contents

Bibliography

[1] Software and system engineering - high-level petri nets - part 1: Concepts, definitions and graphical notation. *ISO/IEC*, 15909(1), 2004.

[2] Systems and software engineering – high-level petri nets – part 2: Transfer format. *ISO/IEC*, 15909(2), 2011.

[3] The Eclipse Foundation. Eclipse, 2012.

[4] The Eclipse Foundation. Eclipse modeling framework, 2012.

[5] The Eclipse Foundation. Graphical modeling project, 2012.

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

[7] 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.

[8] Ekkart Kindler. epnk: A generic pnml tool - users' and developers' guide : version 0.9.1. *IMM-Technical Report*, 3, 2011.

[9] Ekkart Kindler and Csaba Pales. 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.

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

[11] Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank, Martin Stig Stissing, Michael Westergaard, SÃ¸ren 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.

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