MuPSi - a multitouch Petri net simulator for transition steps

Thomas Irgang[1], Andreas Harrer[1], Robin Bergenthum[2]

[1]Lehrstuhl für Angewandte Informatik, Kath. Universitaet Eichstaett
`{thomas.irgang,andreas.harrer}@ku-eichstaett.de`

[2]Lehrgebiet Softwaretechnik und Theorie der Programmierung, FernUni Hagen
`robin.bergenthum@fernuni-hagen.de`

Abstract. Petri nets are very useful for modeling systems with concurrent behavior. There are many editors which support the creation of a Petri net and almost all editors offer the possibility of simulation, i.e. firing of transitions. Due to the limitation to possible user interaction using a conventional user interface having only one focus a concurrent firing of several transitions is not supported. In this paper a Petri net simulator (MuPSi) is presented, which supports the execution of transition steps, i.e. multisets of transitions, using a multitouch user interface. MuPSi enables concurrent execution of transitions in a single or multi-user environment. Simulation of transition steps helps to understand the concurrent behavior of a Petri net.

1 Introduction

Petri nets have a clear formal semantic and a simple graphical representation. They are a popular choice for integrated modeling of systems with concurrency in many application areas, such as software engineering, business process modeling, hardware design and controller synthesis. To support the creation of Petri nets there are many editors which differ depending on their field of application. VipTool [1,2] is specialized on partial order semantics of Petri nets. CPN Tools [3] considers an extension of Petri nets, so-called colored Petri nets. WoPeD [4] is developed for teaching purpose. Each of these editors offers the possibility to simulate Petri nets, i.e. they can simulate the firing of a sequence of transitions.

Usually the simulation consists of two steps. The Petri net editor calculates and marks all enabled transitions. Now the user may select and fire one of these transitions. Each firing changes the marking of the Petri net and the set of all enabled transitions is recalculated. This so-called token game is a very simple but also useful way to test and understand a given Petri net.

The token game allows to simulate the sequential behavior of a Petri net. The tool presented in this paper, the MuPSi (Multitouch Petrinetz-Simulator), removes this restriction by enabling the firing of transition steps. A transition step is enabled to fire if all transitions of the step can be executed simultaneously. If a step of transitions is enabled then also each linearization of the transition step is enabled. For a lot of applications the difference between a step and all its linearizations does not matter. But there are applications where testing and understanding the concurrent behavior of a Petri net is of great value. A good example is in education, because understanding the true concurrent behavior of a Petri net is a reasonable learning objective. Therefore MuPSi allows to simulate and visualize concurrent firing of transitions in an intuitive way which is supported by various input mechanisms. MuPSi can be used as a desktop application as well as in a multitouch environment.

In a learning environment an elegant approach to concurrent firing of transition steps is using a multitouch device. Multitouch environments are becoming increasingly popular as a result of the development and propagation of smartphones and tablet computers. The ability to control multiple pointers in a program allows to select several transitions in a Petri net simultaneously and fire the selected transitions concurrently. For this purpose we have built a multitouch table as shown in Figure 1. With the help of this multitouch table and MuPSi multiple users can play the token game concurrently. If a multiset of transitions is selected to be fired concurrently and there are too little tokens MuPSi supports the users to solve this conflict by a reduction of the chosen transition step.

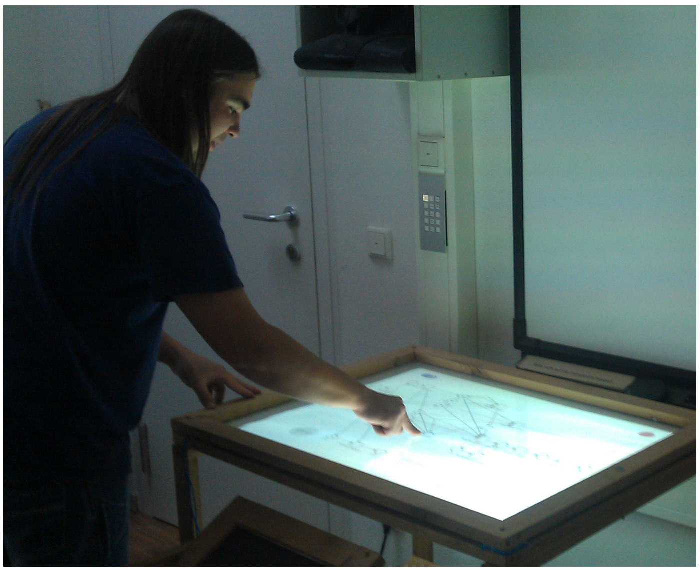


Fig. 1. MuPSi on a multitouch table

The next section discusses the representation of enabled transition steps in Petri nets. At first, in Section 2, different ways to indicate enabled transition steps are developed and evaluated. At second, in Section 3, different ways to allow users the creation of a transition step are discussed. In Section 4 we shortly discuss how MuPSi supports the reduction of a disabled transition step to an enabled sub-step. Section 5 describes the implementation of MuPSi and Section 6 provides a short outlook to further research questions.

2 Representation of enabled transition steps in Petri nets

Petri nets have two different types of nodes: places and transitions. Places can be marked with tokens and a distribution of tokens over all places of the Petri net is called the marking of a Petri net. Transitions and places are connected by directed arcs. MuPSi considers a special class of Petri net so-called place/transition-nets (p/t-nets) [5]. A p/t-net considers arcs with weights and a transition is enabled to fire in a given marking if every place in the preset of the transition carries at least as many tokens as the weight of the arc from the place to the transition indicates. When firing a transition these tokens are consumed and the transition produces new tokens into places in its postset, again according to the weights of the outgoing arcs.

Petri net simulators support the user by highlighting enabled transitions, often by coloring them green. If you allow the firing of transition steps, i.e. a multiset of transitions, the number of tokens in the preset of all transitions given by the transition step must be big enough to enable all transitions concurrently. It happens that one transition is enabled to fire in some transition steps and is not enabled to fire in others. For this reason a strict partition into enabled and not enabled transitions is no longer possible using the step semantic. We extended the color scheme in MuPSi by the additional color yellow. Each yellow colored transition is enabled in the current marking, but there exist other transitions in the Petri net such that if both transitions occure together in a step the step is not enabled. All in all yellow transitions indicate possible conflicts. Green colored transitions symbolize a very low conflict potential, meaning it is possible to combine green transitions with other green transitions in a step of the Petri net. In the following section, we discuss several different approaches on dividing the set of all transitions of a given Petri net and its marking into sets of so-called green, yellow and red transitions. Each approach is implemented in MuPSi and can be useful in different scenarios. In a learning environment it can even be useful to switch between different approaches to get a better understanding of the nature of conflicts in a given Petri net and its concurrent behavior.

Structural conflicts between transitions

Probably the simplest approach to divide the set of all enabled transitions of a Petri net into two sets of transitions, such that green transitions have low conflict potential and yellow a higher conflict potential, is a simple structural analysis of the Petri net. This approach is independent of the current marking of the Petri net. The idea is that an enabled transition can always occur once in a next step if there is no place in its preset which is also in the preset of another transition. If there exists a shared place the concurrent firing of both transitions can lead to a conflict. This is a fairly simple method, since the partition of the set into yellow and green transitions is calculated only once. At runtime this solution requires no additional computational effort, since each transition, when enabled, is always marked with the same color, either yellow or green as computed in the beginning.

This approach can be refined by a small amount of additional computational effort. Assuming that the user cannot add a disabled transitions to a step, a disabled transition can never be in conflict. In the second approach a transition is marked green if it is enabled and there is no place in its preset which is shared with another enabled transition. Figure 2 provides a small example for the first two approaches. The left side of the figure shows the first while the right side of the figure shows the second approach. In this paper colors of the transitions can only be shown as labels. The labels are g(reen) ,y(ellow) or r(ed).

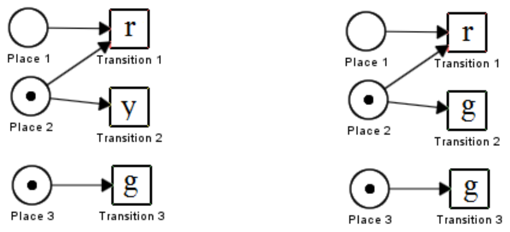


Fig. 2. Coloring of transitions using structural analysis. Left: first approach. Right: second approach.

The number of green colored transitions in the second approach is greater than or equal to the number of green colored transitions in the first approach, but the partition of green and yellow transitions has to be recalulated after each step, since it depends on the marking of the Petri net. Both approaches achieve that a set of green transitions together with only one yellow transition always is enabled to fire. If a step contains more than one yellow transition or some green colored transitions at least twice, it can happen that this step is not enabled.

Conflicts between transitions

To further increase the number of green transitions, the third approach analyses the current marking in more detail. The idea is to color transitions green as long as the marking is sufficient to enable all of them concurrently. For this approach, first we try to fire the set of all enabled transitions in one transition step. If there is a place that causes a conflict each transition in its postset is colored yellow. The remaining enabled transitions are colored green.

Again, each set of green colored transitions together with one yellow colored transition is enabled to fire. The third approach is able to color even more

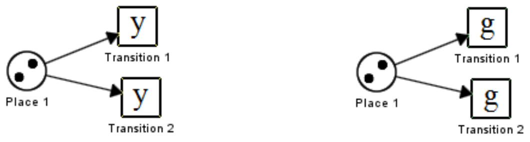


Fig. 3. Coloring of transitions using conflicts between enabled transitions. Left: second approach. Right: third approach.

transitions green. That means that a concurrent firing of more than one yellow colored transition leads to a conflict more probably.

Conflicts between transitions allowing for limited autoconcurrency

The fourth and fifth approach also consider autoconcurrency among transitions. Till now, we only took care of conflicts that arise in sets of transitions. MuPSi also supports a firing of multisets of transitions. In both approaches we have to fix an upper bound to autoconcurrency of transition occurence. Defining such a limit is useful because only a transition with an empty preset can fire autoconcurrently without any limitation. Let n be such an upper bound for the autoconcurrency of all transitions.

In the fourth approach, we test the multiset of transitions which contains each enabled transition n times. As in approach three, transitions which contain places in its presets, which cause conflicts are colored yellow. All other enabled transitions are colored green. With this approach we can ensure that a step of green transitions is enabled if each transition is contained at most n times in the transition step. If we choose one as an upper bound, the fourth approach equals to the third approach. It is easy to see that the number of green colored transitions decreases with increasing n.

The fifth and most complex approach implemented in MuPSi uses the calculation of maximal steps of the Petri net in the given marking. A maximal step of a Petri net is an enabled step which is not included in any other enabled step. If a transition is at least n times contained in all maximal steps, the transition is enabled to fire at least n times independently from any other transition occurence.

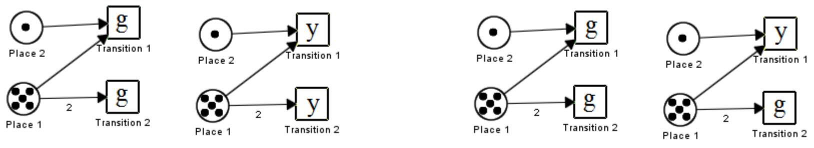


Fig. 4. Coloring of transitions considering autoconcurrency. Left: fourth approach (bound one and two). Right: approach five (bound one and two).

As a result of this approach each multiset of green colored transitions is enabled, if any transition is contained at most n times. As a second result we get that in any enabled multiset (consisting of yellow and green transitions) the number of each green transition can be increased to n without causing any conflicts.

3 Selection of a transition step

While playing the token game it is up to the user to select transition steps and fire them. Depending on the scenario different input methods fit better for the selection of the transition steps. One considered possible scenario is MuPSi running on a desktop computer, another scenario is MuPSi running on a multitouch device. MuPSi offers different input methods to fit both scenarios. Remark that while composing a step the coloring of the transitions is not updated, since we assume that each step is build concurrently by different users.

Manual trigger

The first and simplest way to select a transition step is to use an explicit manual trigger. The user adds transitions to a step by clicking enabled transitions. After building a transition step the user triggers the firing of the transition step by clicking a fire button. A disadvantage of this input method is that it is not well suited in a multi-user environment without choosing a user having a special role, i.e. a moderator, controlling the fire button. The great advantage of a manual tigger in a single user environment is that the user can build and fire transition steps without time pressure.

Fixed time intervals

In some scenarios the use of a fire button is not suitable. In a second approach MuPSi is able to use fixed time intervals. If an interval ends the selected step will be fired if possible. This means that for a fixed time, for example three seconds, transitions are collected and these transitions are fired concurrently at the end of the time interval. This input method is suitable for both: the simulation with a single user using a PC as well as for the collaborative use with a multi-touch device.

Sliding time intervals

A more flexible approach than using a fixed time interval is a so-called sliding time interval. If any user clicks a transition a short countdown starts. If a transition is pressed in this time period the transition is added to the transition step and the countdown restarts. This will continue until the countdown expires. This input method is suitable for a multi-touch table as well as for the usage with a PC. Typically the time interval for a single user should be larger than in multi-user mode. Transition steps might become very large and more likely not executable if several active users continuously add transitions to a transition step and thereby restart the countdown. This way all transitions which are clicked within a short period of time are never split into two different transition steps. Such a splitting could be caused by using a fixed time interval.

Time intervals by overlapping of actions

In the last, most elegant approach MuPSi distinguishes between pressing and releasing a transition. As long as any transition is pressed, transitions are added to a transition step. If no transition is pressed the collected step is fired. This input method is of course only usable with a multitouch device [6] and was the real impulse to develop MuPSi. In a single user environment the user may hold one transition down and then collect other transitions with his second hand to build a transition step. If a user wants to fire a transition autoconcurrently in one transition step he may tap one transition while holding the same transition pressed with his other hand. This approach leads to the feeling of real concurrent live simulation of a Petri net.

4 Conflict solving

In this section we describe the current conflict solving approach implemented in MuPSi. Since MuPSi is intended as a tool for teaching one of its essential features is to support the users if a disabled transition step is selected. Our approach for conflict solving is to offer the users sub-steps of the conflicting step.

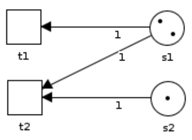
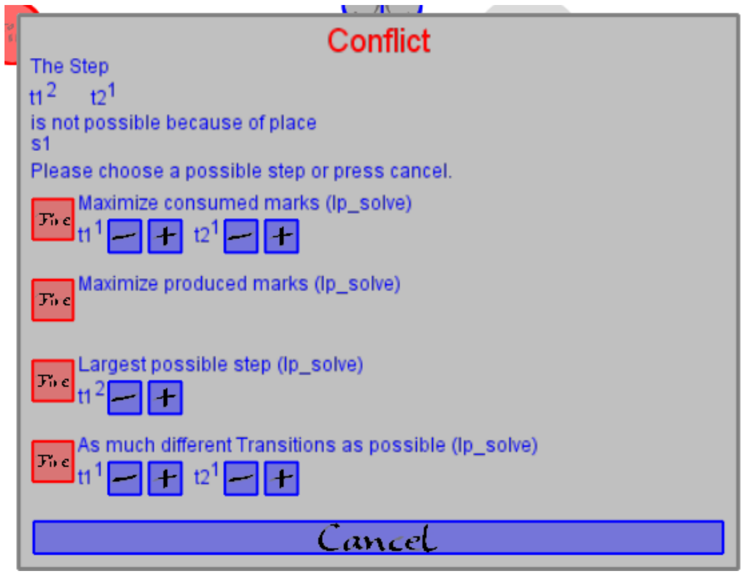


Fig. 5. An example for conflict solving

To calculate possible sub-steps we model the problem as a linear integer optimization problem. A linear optimization problem consists of a target function and side conditions which restrict the set of solutions. For our model the interesting side conditions are defined through the net structure and the conflicting step. If we try to fire the step $\{t_1, t_1, t_2\}$ in the net shown in Figure 5 this leads to a conflict in place s_1. To define the suitable target function we need to know the users intention, in case we have no further information we can only feedback different solutions using different target functions. Currently MuPSi implements four different target functions: maximization of the consumed tokens, maximization of the produced tokens, the maximal number of transition occurrence or a maximal number of occurrence of different transitions.



Conflict

The Step

t1 2 t2 1

is not possible because of place

s1

Please choose a possible step or press cancel.

Maximize consumed marks (lp_solve)

t1 1 t2 1

Maximize produced marks (lp_solve)

Largest possible step (lp_solve)

t1 2

As much different Transitions as possible (lp_solve)

t1 1 t2 1

Fig. 6. The conflict solving dialog.

The solutions of the optimization problem are currently calculated with the help of the LPSolve library. LPSolve is a library which is released under LGPL (GNU Lesser General Public License) and solves different linear problems with a specialization to linear optimization problems. The calculated sub-steps are presented to the users through a dialog, see Figure 6, which allows the user to readjust the transition step. This is not yet a satisfying solution because it is a break in the multi-user operation since a dialog can only be handled by one user.

5 Implementation

MuPSi is currently implemented as part of a thesis of the first author of this paper. It will be available as a plug-in for VipTool and can be used as a stand-alone version at the moment. MuPSi is optimized for the use on a multitouch device and is implemented in Java. MuPSi is available for download on the MuPSi homepage ($www.fernuni-hagen.de/sttp/forschung/mupsi.shtml$). This section describes the implementation and functions of MuPSi in more detail.

There are currently three popular frameworks which provide multi-pointer input. The oldest of these frameworks is the multimouse driver which allows to use multiple mice to control multiple mousepointers. This is not a true multi-touch setting but it has the advantage that multi-touch input can be simulated without special hardware. In the consumer sector the Windows 7 multitouch framework has a very wide dissemination because it is part of the Windows 7 operating system. It supports multi-touch touchpads and inexpensive consumer multi-touch screens. The disadvantage of these monitors is the limited support of only two input points. In the professional area the TUIO protocol is very common. TUIO is a network protocol which supports pointers and tagged objects. Tagged objects are physical objects with a unique ID. If an application wants to professionally use multi-touch tables it has to support this protocol. More information about the TUIO protocol and the usual hardware setup can be found at *www.tuio.org*.

We decided to use the MT4J (multi-touch for Java) library, because it supports all of these input frameworks. The big advantage of this library is that it abstracts the input method and thus MuPSi can support all these input methods without software changes. The MT4J Library is released under GPL (GNU General Public License).

Our homemade multi-touch table, shown in Figure 1, uses a projector for rear projection onto a display output and an infrared camera to detect finger inputs. The table uses a basic FTIR surface as infrared light source. The usage of infrared light for finger detection is necessary to distinguish the input from the output image. For the calculation of the input data from the IR image, we use the Community Core Vision software, briefly CCV, from the NUI Group. CCV is an open source software which is released under LGPL and has several customizable filters in order to ensure a good detection. The detected inputs are sent to a given IP address using the TUIO protocol. The NUI Group is an open source community with the aim to provide natural user interfaces.

For the user interface we also used the MT4J Library. MT4J contains the Java OpenGL library, JOGL, and offers a set of multi-touch optimized GUI elements. The elements are grouped as a 3D scene graph. We use self defined user elements for MuPSi which are built from polygons. The usage of a 3D scene instead of a default Java GUI is useful because of the convenient implementation of scrolling, zooming and rotation which are common operations for multi-touch surfaces. MT4J allows to process the input with the aid of default and self defined gesture processors. MuPSi uses this for example to provide a two-finger-zoom gesture. The usage of hardware OpenGL support is preferred but it is also possible to use software rendering.

Figure 7 shows a screen shot of MuPSi. The manual fire button is labeled with 1.This button is only visible if the manual trigger is selected as fire mode. The undo and redo buttons are labeled with 2. MuPSi supports to undo all fired steps to make simulation more comfortable. All undone steps can be redone as long as no new step was fired. The tools button is labeled with 3. This button allows to move and zoom the loaded net. By clicking the button it is minimized to save screen space.

All of the coloring and step building methods presented in the previous sections are implemented in MuPSi and can be used by selecting them in the configuration file before starting the program. The coloring method can also be changed during the simulation by pressing the key c. The configuration file also allows the change of the resolution, the rendering method and all visual param-

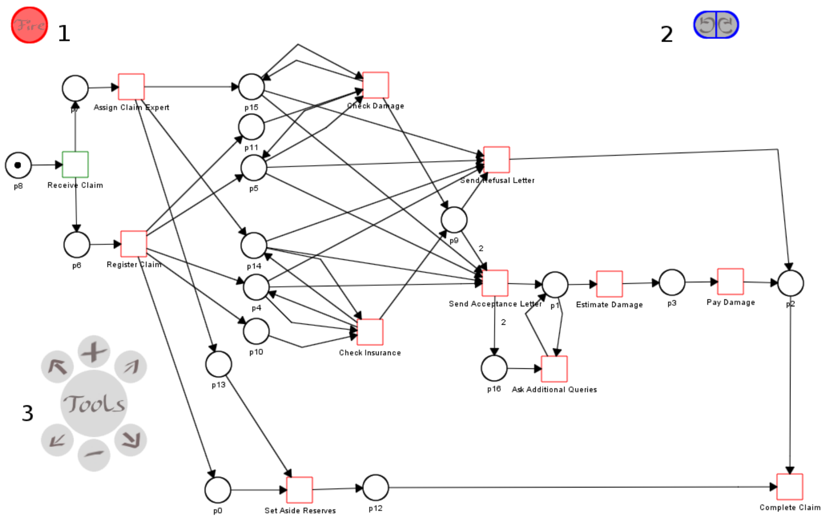


Fig. 7. A screenshot of MuPSi.

eters of the net elements. If visual parameters for the net elements are defined in the config file, the information in the PNML file is replaced with these values.

MuPSi is designed in a modular way to make it easy to implement new firing modes, coloring modes and conflict solvers but we have decided to offer no plugin infrastructure because MuPSi is already designed as a VipTool plugin.

Since MuPSi is not yet available as a VipTool plugin but as a standalone application, it is useful to give a brief description how to use it with various operating systems. MuPSi is, as a Java software, platform independent but some of the used libraries are platform dependent such as the LPSolve library and the OpenGL library contained in MT4J. The available MuPSi zip package contains different startup scripts for Windows x86 and x64 and for Linux x86 and x64. The startup scripts define different platform-dependent Java classpath variables. To support scenario dependent startup scripts a set of command line parameters is defined which sets the display resolution, the used PNML file and a few more settings. A full list of command line parameters is printed with the parameter -h or –help.

6 Conclusion and outlook

In this paper the tool MuPSi was presented. It is a Petri net simulator designed for multi-touch devices which supports simulation of transition steps. This enables a simple concurrent firing of transitions of a Petri net. We discussed several options to visualize enabled transition steps and to discover conflicts in a current marking of a Petri net. We disscused possible input modes for the creation of transition steps. In MuPSi a single user or multiple users may select the most appropriate input mode for each scenario.

After a full implementation of MuPSi we will focus on the aspects of conflict resolution and presentation of these solutions. Another point we miss at the moment is a smooth VipTool integration.

References

1. Desel, J., Juhás, G., Lorenz, R., Neumair, C.: Modelling and Validation with Viptool. In W.M.P. van der Aalst; A.H.M. ter Hofstede; M. Weske, ed.: Business Process Management. Volume 2678 of Lecture Notes in Computer Science., Springer (2003) 380–389
2. Bergenthum, R.: Algorithmen zur Verifikation von halbgeordneten Petrinetz-Abläufen: Implementierung und Anwendungen. Master's thesis, Katholische Universität Eichstätt-Ingolstadt (2006)
3. Jensen, K., Kristensen, L., Wells, L.: Coloured petri nets and cpn tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer (STTT)9(3-4) (2007) 213–254
4. Freytag, T.: WoPeD – Workflow Petri Net Designer. In: ATPN. (2005)
5. Desel, J., W.Reisig: Place/Transition Petri Nets. In W. Reisig; G. Rozenberg, ed.: Petri Nets. Volume 1491 of Lecture Notes in Computer Science., Springer (1998) 123–174
6. Burmester, M., Koller, F., Höflacher, C.: Touch it, move it, scale it - multitouch. Technical report, HDM Stuttgart (2009)