ImageNetDiff: A Visual Aid to Support the Discovery of Differences in Petri Nets

Lawrence Cabac, Jan Schlüter

University of Hamburg, Department of Computer Science,
Vogt-Kölln-Str. 30, 22527 Hamburg
`http://www.informatik.uni-hamburg.de/TGI`

Abstract In this paper we propose a method and present a tool as plugin for Renew that supports the process of discovery of differences in possibly conflicting versions of Petri net code. The method uses the image representaion of the net graph and compares the pixels of the exported Petri nets. The tool uses the image processing of ImageMagick. An open source graphical tool kit, which is available on all common operating systems.

Keywords: high-level Petri nets, nets-within-nets, reference nets, net components, Renew, modeling, agents, multi-agent systems,

1 Introduction

During development of large applications or models with Petri nets developers frequently encounter different and/or conflicting versions of the code base artifacts. Especially in shared projects where Petri net code is shared through source code management systems (SCM) such as the Concurrent Versions System (CVS) or Subversion conflicts frequently appear and have to be resolved manually by the developer. In the evaluation of the code (Petri nets) the main problem is the identification of the syntactical differences or equalities. However, on the one hand formally it is very hard to verify graph equality and even harder to determine the minimum of parts that are different. The graphical representation, on the other hand, may contain valuable hints for the mentioned problems but may also differ without change in the syntax. The merging of changes is usually a manual task, even if only different parts of the nets have been modified. In contrast, when text-based source code is used, merging of non-conflicting concurrent changes is possible. To our knowledge no tools exist so far that manages the merging to some extend or even support the developer in this task. Even if a string representation of the net code exists, usually this code is not handleable by common tools such as *diff* [2] (or windiff).

In this paper we propose a simple but efficient method that can simplify the task of the discovery of differences under certain conditions. To this means we exploit the graphical representation of the nets and transfer the problem to finding differences in the visual image of the Petri nets. We also present an implementation of the method as plugin for Renew [4,5]. In Section 2 we describe the method, its implementation and integration within Renew. Section 3 presents an example to illustrate the method and tool.

2 Discovery of Net Differences

The development of models within development groups frequently leads to conflicting models. Even if the system model is decomposable in many parts, still the problem persists – as with all source code – that within one design artifact (Petri net) several changes can occur concurrently and have to be merged. In this situation two tasks have to be performed. First, the differences have to be identified. Second, the changes have to be included. For Petri nets these tasks usually have to be performed manually.[1] We propose that tool support for the discovery of net differences can accelerate the development of net system models significantly.

2.1 Scenarios

We can distinguish at least two different scenarios in which the tool can be utilized: the *similarity check* and the *difference discovery*. In the similarity check a developer does not know, whether two Petri nets or two versions of the Petri net own the same code (are syntactically/semantically equal but may differ in visually). For text-based code exist code beautifiers that manage to unify the style of code as a preparation for the differences tools. Often net elements or text inscriptions have been moved in the image by another developer and this has been committed to the repository resulting in a conflict. If the nets (or the net versions) contain only small differences (e.g. only one node has been moved) the ImageNetDiff image will show instantly that the nets are syntactically equal. The checking of the equality of the nets is thus reduced to the checking of the graphically differing parts.

In the difference discovery the visual areas of the net that own differences can be easily spotted by the developer. Again if simple changes have been made in the Petri net, such as the removal or the addition of net elements, the ImageNetDiff image will directly and clearly show the differences. If this is not the case and substantial changes have been made, at least the ImageNetDiff image points out the net areas which are of concern to the developer and which parts have not changed.

2.2 Technique

The tool makes use of the internal export function of Renew and the ImageMagick[3] tool kit. For the production of the differences image in the format of Portable Network Graphics (PNG) or alternatively Encapsulated Postscript (EPS) first the nets are being exported to the file system as image. Then the

[1] An alternative strategy is the avoidance of concurrent changes.

exported images are passed on as arguments to the imaging tool to compute the differences image, which will also be stored in the file system. The resulting image will feature light grayish drawing elements for the parts of the original images that are equal and two different shades of red for the additional and removed graphical parts Finally, for the convenience of the user the image is displayed by Renew once the computation of the differences image has finished. Sources of nets that are to be compared can be either nets that are opened within the editor of Renew or nets from the file system.

2.3 Constrains and Limitations

Several limitations to the presented method exist that result from the underlying tools. For a flawless comparison the compared images must have the same size. The comparison can not be customized, yet. For instance, the color scheme is fix. The results for nets in which all graphical elements have been moved are not satisfying, yet, because the images are compared coordinate pixel against coordinate pixel. There is no integration with the Petri net representation, yet. Thus, the discovery of changes is supported but the knowledge has to be transferred to the Petri net.

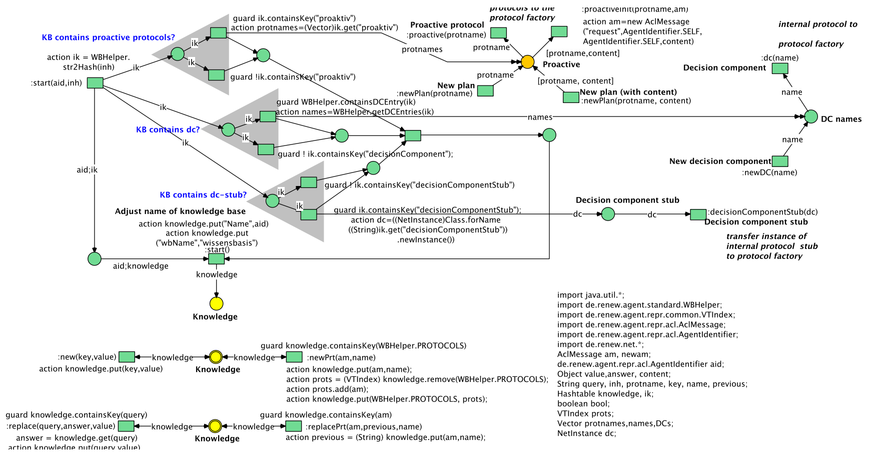
3 Example

As an example net for the presentation of the method we present the knowledge base net of the Mulan standard agents. The two nets differ – pragmatically – in the fact that they support two different property files formats: simple properties (*kb*) and XML notation (*kbe*, kb enhanced). The net that supports the enhanced representation is built upon the simple version, thus they are comparable. To find the similarities and differences of the implementation we present fragments of both nets in Figures 1 and 2. The fragments show the initialization of the net with the initial knowledge parts of the agents interface to the knowledge base and the interface that handles the initialization of decision components (active knowledge). Figure 3 then shows a screenshot of the resulting difference image (similar fragment).[2]

The developer's awareness is instantly attracted by the bright red net elements and inscriptions. One can see simple additions – manually marked in the image by dotted outlined squares – and also changes to the code / inscriptions – manually marked by dotted outlined ellipses –that have been made. The image shows clearly that all of the old net structure has been preserved. Only additional net elements and inscriptions have been added and some inscriptions have been altered.

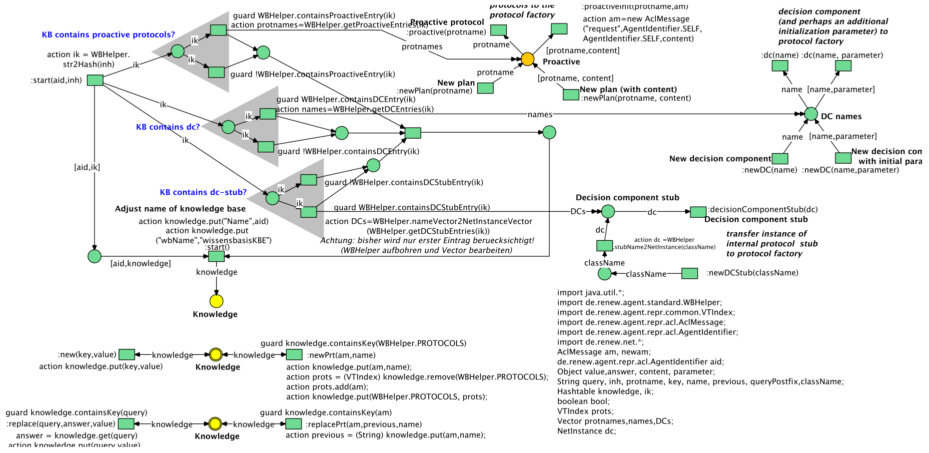
In a scenario of a shared development, if a developer is confronted by a concurrent change of the net, which results in a conflicting version of the net code, the tool can help the developer to decide whether the code has been manipulated,

[2] The dashed squares and ellipses are added manually.


KB contains proactive protocols?
action ik = WBHelper.
str2Hash(inh)
:start(aid,inh)
ik
ik
ik
guard ik.containsKey("proaktiv")
action protnames=(Vector)ik.get("proaktiv")
guard !ik.containsKey("proaktiv")
protnames
Proactive protocol
:proactive(protname)
protname
protocols to the
protocol factory
:proactiveInit(protname,am)
action am=new AclMessage
("request",AgentIdentifier.SELF,
AgentIdentifier.SELF,content)
[protname,content]
Proactive
New plan
:newPlan(protname)
protname
[protname, content]
New plan (with content)
:newPlan(protname, content)
internal protocol to
protocol factory
:dc(name)
Decision component
name
DC names
name
New decision component
:newDC(name)
KB contains dc?
guard WBHelper.containsDCEntry(ik)
action names=WBHelper.getDCEntries(ik)
names
guard ! ik.containsKey("decisionComponent");
aid;ik
KB contains dc-stub?
guard ! ik.containsKey("decisionComponentStub")
guard ik.containsKey("decisionComponentStub");
action dc=((NetInstance)Class.forName
((String)ik.get("decisionComponentStub"))
.newInstance())
Decision component stub
dc
dc
:decisionComponentStub(dc)
Decision component stub
transfer instance of
internal protocol stub
to protocol factory
Adjust name of knowledge base
action knowledge.put("Name",aid)
action knowledge.put
("wbName","wissensbasis")
:start()
aid;knowledge
knowledge
Knowledge
:new(key,value)
action knowledge.put(key,value)
knowledge
Knowledge
knowledge
guard knowledge.containsKey(WBHelper.PROTOCOLS)
:newPrt(am,name)
action knowledge.put(am,name);
action prots = (VTIndex) knowledge.remove(WBHelper.PROTOCOLS);
action prots.add(am);
action knowledge.put(WBHelper.PROTOCOLS, prots);
guard knowledge.containsKey(query)
:replace(query,answer,value)
answer = knowledge.get(query)
action knowledge.put(query,value)
knowledge
Knowledge
knowledge
guard knowledge.containsKey(am)
:replacePrt(am,previous,name)
action previous = (String) knowledge.put(am,name);

```
import java.util.*;
import de.renew.agent.standard.WBHelper;
import de.renew.agent.repr.common.VTIndex;
import de.renew.agent.repr.acl.AclMessage;
import de.renew.agent.repr.acl.AgentIdentifier;
import de.renew.net.*;
AclMessage am, newam;
de.renew.agent.repr.acl.AgentIdentifier aid;
Object value,answer, content;
String query, inh, protname, key, name, previous;
Hashtable knowledge, ik;
boolean bool;
VTIndex prots;
Vector protnames,names,DCs;
NetInstance dc;
```

Figure 1. Knowledge base net template of a Mulan agent.


KB contains proactive protocols?
action ik = WBHelper.
str2Hash(inh)
:start(aid,inh)
ik
ik
ik
guard WBHelper.containsProactiveEntry(ik)
action protnames=WBHelper.getProactiveEntries(ik)
guard !WBHelper.containsProactiveEntry(ik)
protnames
Proactive protocol
:proactive(protname)
protname
protocols to the
protocol factory
:proactiveInit(protname,am)
action am=new AclMessage
("request",AgentIdentifier.SELF,
AgentIdentifier.SELF,content)
[protname,content]
Proactive
New plan
:newPlan(protname)
protname
[protname, content]
New plan (with content)
:newPlan(protname, content)
decision component
(and perhaps an additional
initialization parameter) to
protocol factory
:dc(name) :dc(name, parameter)
name [name,parameter]
DC names
name [name,parameter]
New decision component
New decision con
with initial para
:newDC(name) :newDC(name,parameter)
KB contains dc?
guard WBHelper.containsDCEntry(ik)
action names=WBHelper.getDCEntries(ik)
names
guard !WBHelper.containsDCEntry(ik)
[aid,ik]
KB contains dc-stub?
guard !WBHelper.containsDCStubEntry(ik)
guard WBHelper.containsDCStubEntry(ik)
action DCs=WBHelper.nameVector2NetInstanceVector
(WBHelper.getDCStubEntries(ik))
Achtung: bisher wird nur erster Eintrag beruecksichtigt!
(WBHelper aufbohren und Vector bearbeiten)
Decision component stub
DCs
dc
:decisionComponentStub(dc)
Decision component stub
dc
action dc =WBHelper
.stubName2NetInstance(className)
className
className
:newDCStub(className)
transfer instance of
internal protocol stub
to protocol factory
Adjust name of knowledge base
action knowledge.put("Name",aid)
action knowledge.put
("wbName","wissensbasisKBE")
:start()
[aid,knowledge]
knowledge
Knowledge
:new(key,value)
action knowledge.put(key,value)
knowledge
Knowledge
knowledge
guard knowledge.containsKey(WBHelper.PROTOCOLS)
:newPrt(am,name)
action knowledge.put(am,name);
action prots = (VTIndex) knowledge.remove(WBHelper.PROTOCOLS);
action prots.add(am);
action knowledge.put(WBHelper.PROTOCOLS, prots);
guard knowledge.containsKey(query)
:replace(query,answer,value)
answer = knowledge.get(query)
action knowledge.put(query,value)
knowledge
Knowledge
knowledge
guard knowledge.containsKey(am)
:replacePrt(am,previous,name)
action previous = (String) knowledge.put(am,name);

```
import java.util.*;
import de.renew.agent.standard.WBHelper;
import de.renew.agent.repr.common.VTIndex;
import de.renew.agent.repr.acl.AclMessage;
import de.renew.agent.repr.acl.AgentIdentifier;
import de.renew.net.*;
AclMessage am, newam;
de.renew.agent.repr.acl.AgentIdentifier aid;
Object value,answer, content, parameter;
String query, inh, protname, key, name, previous, queryPostfix,className;
Hashtable knowledge, ik;
boolean bool;
VTIndex prots;
Vector protnames,names,DCs;
NetInstance dc;
```

Figure 2. Enhanced knowledge base net template of a Mulan agent.

the syntax has not been changed and/or if the changes have been made in the same areas of the net. Thus, the manual act of merging the code or model can be significantly accelerated.

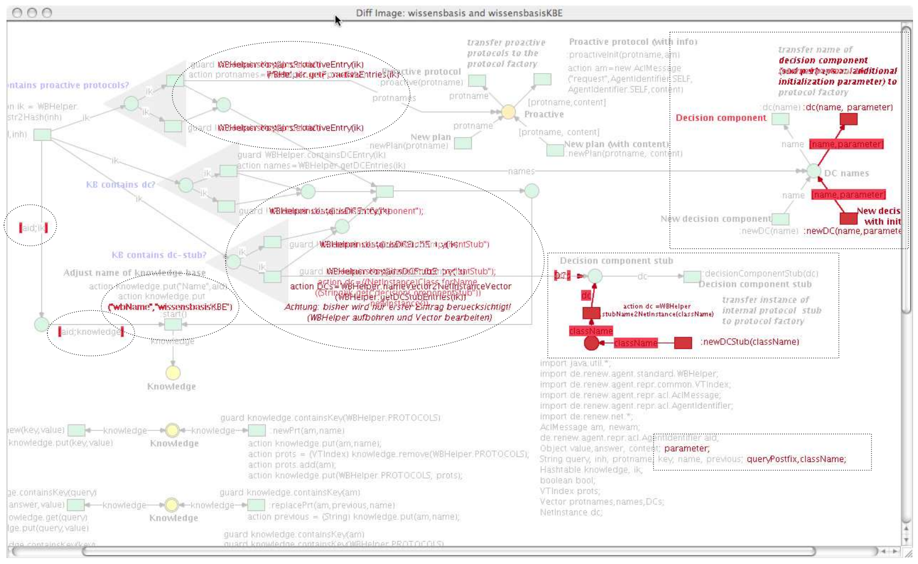


Figure 3. Screenshot showing differences of the two Petri nets.

4 Conclusion, Discussion and Outlook

Although the approach is rather simple, the results are effective and surprisingly efficient. Developers of Petri net models have the means to check for differences in their graphical code by the means of visual support. Clearly a code beautifier for Petri nets would improve the results of the ImageNetDiff plugin considerably. Here net components [1] could help to impose a conventionalized structure upon the nets.

The presented approach makes use of the graphical representation of the Petri nets, the export to an image format and the power of the graphical framework ImageMagick. There are, however, several other possibilities to tackle the presented problem. A similar possibility would be if the Petri net views in the editor could use layered canvas and alpha channels. Thus, one could easily find differences between versions, which are loaded in overlapping layers. One could compute equality of Petri nets on the ground of the formal representation including node and arc ids. Alternatively, one could program a diff tool on the ground of an exchange format such as PNML.

The presented method and the tool leaves room for many improvements. By choosing different color schemes for the diff images the readability could be improved significantly. However, since the used tools main purpose of comparing images is not concerned with graph representations, it does not support this feature and a reimplementation or switch to another tool could – with some effort – produce better results. The interpretation of the graphically highlighted elements could lead to a integration of useful information within the Petri net editor to further support the merging of concurrent changes.

Principally, with the presented method the results from image processing have to be re-transferred to the application domain. Alternatively, similar differences can be computed and presented to the developer on the direct analysis of Petri net structures. Here, additional information could support the process of matching elements in Petri net versions. For instance, id-tagged net elements (in Renew transitions and places have ids) could be matched. However, this would not solve the problem of constructs that have different ids but are syntactically equal. A method based on a Petri net representation is also less general than the presented method, which can be applied to other graphs such as UML diagrams.

In the future we want to investigate possibilities to automatically merge concurrent non-conflicting changes within a source code management system. Here, the questions of mergeable representations of graph structures and reimplementations of diff tools that may handle Petri net code are in the focus of research.

References

1. Lawrence Cabac, Michael Duvigneau, and Heiko Rölke. Net components revisited. In Daniel Moldt, editor, *Proceedings of the Fourth International Workshop on Modelling of Objects, Components, and Agents. MOCA'06*, number FBI-HH-B-272/06, pages 87–102, Vogt-Kölln Str. 30, D-22527 Hamburg, Germany, June 2006.
2. Gnu diff utilities. online, 2008. `http://www.gnu.org`.
3. Imagemagick homepage. online, 2008. `http://www.imagemagick.org/`.
4. Olaf Kummer, Frank Wienberg, and Michael Duvigneau. Renew – the Reference Net Workshop. Available at: `http://www.renew.de/`, July 2008. Release 2.1.1.
5. Olaf Kummer, Frank Wienberg, Michael Duvigneau, Jörn Schumacher, Michael Köhler, Daniel Moldt, Heiko Rölke, and Rüdiger Valk. An extensible editor and simulation engine for Petri nets: Renew. In Jordi Cortadella and Wolfgang Reisig, editors, *Applications and Theory of Petri Nets 2004. 25th International Conference, ICATPN 2004, Bologna, Italy, June 2004. Proceedings*, volume 3099 of *Lecture Notes in Computer Science*, pages 484–493. Springer, June 2004.