APNNed goes Internet

P. Kemper and C. Tepper *

Dept. of Computer Science IV, University of Dortmund
D-44221 Dortmund, Germany
e-mail: {kemper, tepper}@ls4.cs.uni-dortmund.de

Abstract

APNNed is an editor for hierarchical, colored, and stochastic Petri nets. It is an integral part of the APNN toolbox and serves as a graphical user interface for other members of this toolset. Since it is implemented in Java, it is rather platform independent. In this paper we describe recent enhancements to support modeling and analysis of Petri nets with APNNed via internet. We give a brief sketch of technical issues like HTTP-tunneling that need to be solved to have an APNNed applet running at the client side and an APNNed servlet at the server side, that provides load, save, and analysis functionality for nets.

1 Introduction

A major attraction of web applications is its ease of use; a user need not install the specific application but can rely on a general purpose web browser possibly enriched by a set of plug-ins. Furthermore, web applications can be made accessible to an arbitrary or to a restricted set of users with limited effort. A Petri net tool can profit from these advantages in many ways, for example, for teaching modeling with Petri nets in classes, students can work with Petri net tools at their home PC, or for large research projects, in which different groups of researchers interact and need to use and provide tools or applications for each other. However, few Petri net tools make use of this opportunity yet.

In this paper, we briefly describe a recent attempt to enhance the java-implemented Petri net editor APNNed [7] of the APNN toolbox with functionality to perform as a web application. We consider a scenario, in which APNNed applet is run by a web browser. The applet allows to load a set of available models from an HTTP server or to create new Petri net models. Any model can be edited and modified as in the stand-alone APNNed. Fig. 1 shows a screenshot of APNNed as an applet of an HTML page viewed by a browser. Note that, once started, applets are not restricted to the canvas space taken by the web browser but can expand over the whole screen. The applet provides some functionality like the token game by itself and more sophisticated analysis by help of a servlet that resides at the HTTP server and that uses the APNN toolbox. The servlet provides a load and save operation for persistent storage of nets. Obviously, the possibility to save models at the server's site makes sense only in a restricted group of users. In the following, we will not focus on security but on some technical issues to set up the necessary communication among applet and servlet to transfer information on nets, requests for analysis and analysis results. During its implementation, we had to recognize that

*This research is supported by DFG, collaborative research center 559 'Modeling of Large Logistic Networks'.

this case is not standard and among the many Java books that discuss servlets we found only in [10] a substantial treatment of HTTP-tunneling. Since many Petri net tool developers may consider to turn their tools into web applications, we believe that our experiences are useful and such technical issues are of broader interest.
Before we go into these details, let us briefly recall the functionality and architecture of the APNN toolbox. The APNN Toolbox [2] is a collection of analysis tools, all of which are combined by APNNed [7], which serves as a graphical user interface to edit Petri net models and also controls application of analysis tools for a given net, such that results of analysis are subsequently visualized. All tools interact via a textual interface called Abstract Petri net notation (APNN) [3]. The APNN toolbox contains programs for the functional and quantitative analysis. Functional analysis includes the calculation of invariants, based on the algorithm of Martinez and Silva [13], a randomized token game to detect deadlocks, a analysis techniques based on the reachability graph like deciding liveness and modelchecking. Model checkers exists for a computational tree logic (CTL) [12] and a linear temporal logic (LTL). Quantitative analysis is supported by a discrete event simulator and numerical analysis methods based on an associated continuous time Markov chain (CTMC). The APNN toolbox has a clear focus on state based analysis, in which extremely large but finite state transitions systems are represented by hierarchical or modular Kronecker representations [4, 5, 6, 11]. State-based analysis tools interact via a second, a state level interface that describes the reachability set as a multi-way decision diagram and the reachability graph as a set of matrices that are composed by Kronecker operations.

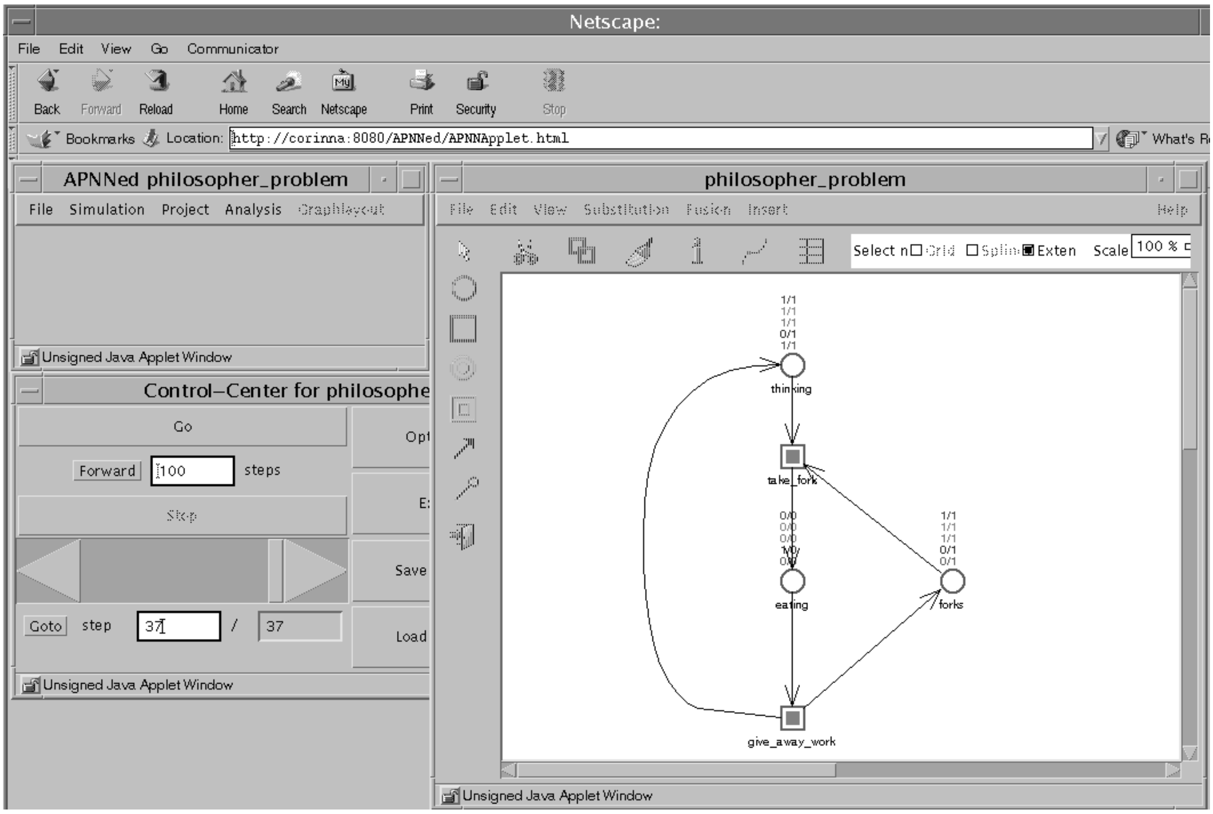

Figure 1: Screenshot of the internet version of APNNed

The paper is structured as follows. Section 2 describes design considerations and identifies four basic operations to achieve an internet version of APNNed. Section 3 describes how to set up an applet-servlet communication (HTTP-tunneling) for an applet, while Section 4 gives the counterpart for the servlet. We summarize in Section 5.

2 Design issues

Web applications are often implemented by Common Gateway Interface (CGI) programs such that a HTTP request of a client results in the instantiation of a process at the server's site to generate a response which often is a web page in HTML. Starting a new process for each HTTP-request causes a significant overhead if the response computation itself only takes a minor effort. A servlet is a similar concept in Java but it runs on threads rather than processes with obvious advantages for performance and the ability to exchange data among threads, such that a servlet can store data over a sequence of requests from a client. Java servlet technology [10] provides web developers with a simple mechanism for extending the functionality of a web server. Servlets are server- and platform-independent and have access to the entire Java APIs [8], [9]. A servlet profits from Java's general properties like portability, performance, re-usability and crash protection (to the extend an existing virtual machine and language implementation can keep pace with expectations).
APNNed is implemented in Java, so that it is a natural choice to use servlets. As mentioned above, our scenario focuses on four simple operations, namely to load, save and analyze nets at the server's site and an additional operation to give information on which nets are available at the server. We do not consider additional topics imposed by storing nets by the server, which nevertheless need to be considered in practice, these include security, restricted access, privacy and sharing of nets, but also traversal of directory structures, disc usage and cpu usage by external clients for analysis etc.
We focus on the applet-servlet communication. The four operations require transfer of an operation identifier plus operation specific parameters as given below:

Operation	Parameter1	Parameter2	Parameter3
load	operation=load	netname=Example	
save	operation=save	net=<APNN description>	netname=Example
analyze	operation = analyze	netname=Example	method=name+parameters
directory	operation=dir		

The information for each operation need to be encoded at the sender and decoded by the receiver. Parameters are rather straightforward, i.e. netname gives the filename of the net, net gives a textual description of the net in APNN format, method gives an identifier for the selected analysis method plus some additional method-specific parameters.
Since applet and servlet are both Java applications, one can transfer serialized Java objects, such that a receiver can pick the complete information for an object just by a single call to *readObject* and one need not programme the parsing and assignment procedure to instantiate an appropriate object. Alternatively, applet and servlet can communicate via buffered input stream to transfer binary or ASCII data. In both cases, an Java *URLConnection* is needed to establish an input stream. The internet version of APNNed uses an object input stream to send and receive objects of class *String*. Clearly, we would use an object stream to exchange net objects but since some internal APNNed classes to represent a net are not serializable yet, we

need to transfer an APNN description as a string object and the receiver needs to parse this string. The encoding of an operation into a string uses an URLencoder object, e.g., for the load operation it is: `String data = "operation=" + URLEncoder.encode("load") + "netname=" + URLEncoder.encode(<name of Petri net>);`
Encoding for other operations is done accordingly.
In the following, we present the necessary steps to call the operations above by an applet and to transfer corresponding answer by an servlet.

3 Starting communication by a client

The applet sends first data to the servlet before the servlet sends back any kind of response. To start the communication, an applet mainly needs to open a connection with an URLconnection object and to generate a request with an appropriate output stream . The steps given below give the details and clearly identify the Java classes that are involved, for more information see [10]. The steps need to be integrated in a try/catch block to account for exceptions.

1.) Create a *URL* object which references to the home of the applet
```
URL currentPage = getCodeBase();
String protocol = currentPage.getProtocol();
String host = currentPage.getHost();
int port = currentPage.getPort();
String urlSuffix = "/servlet/SimpleServlet";
URL dataURL = new URL(protocol, host, port, urlSuffix);
```
2.) Create a URLConnection object
```
URLconnection connection = dataURL.openConnection();
```
3.) Prohibit the web browser to cache the URL data
```
connection.setUseCaches(false);
```
4.) Disclaim the system to allow read and write of data
```
connection.setDoOutput(true);
```
5.) Create a *ByteArrayOutputStream* to buffer the data that should be sent to the servlet
```
ByteArrayOutputStream byteStream = new ByteArrayOutputStream(512);
```
6.) Connect the *ByteArrayOutputStream* with the output stream *PrintWriter*
```
PrintWriter out = new PrintWriter(byteStream, true);
```
7.) Put the data for the server in the buffer
```
String data = "param1=" + URLEncoder.encode(value1)
           + "&param2=" + URLEncoder.encode(value2);
out.print(data);
out.flush();
```
8.) Set the *Content-Length*-Header
```
connection.setRequestProperty("Content-Length",
           String.valueOf(byteStream.size()));
```
9.) Set the *Content-Type*-Header
```
connection.setRequestProperty("Content-Type",
           "application/x-www-form-urlencoded");
```
10.) Send the data to the servlet
```
byteStream.writeTo(connection.getOutputStream());
```

The servlet sends answers through a response object that connects itself to the connection object that was instantiated for the request. Hence the applet needs to set up an input stream on this connections in order to receive the answer of the servlet. The following steps are necessary for this task.

11.) Create an ObjectInputStream
```
ObjectInputStream in = new ObjectInputStream(connection.getInputStream());
```
12.) Read data (*String*) with the method *readObject*
```
String in = (String)in.readObject();
```
13.) Close input stream
```
in.close();
```

In order to implement the applet, we made only minor extensions to APNNed to implement the communication procedures above and to avoid any kind of access to the local file system. It remains to describe the servlet side of the communication.

4 Responding to client request by a servlet

The servlet needs to take care of incoming requests by generating a new thread for each request. In this thread, the servlet must decode the message and interpret its content to trigger appropriate actions for a response. A servlet extends the class HttpServlet, its most relevant methods are *init*, *service*, *doGet*, and *doPost*. The last three functions have two parameters, a request object of class *HttpServletRequest* and a response object of class *HttpServletResponse*. The request object allows to access incoming HTTP-headers or data by opening an input stream. The response object allows to send HTTP-status codes and data by opening an output stream. Due to the similarities between receiving and sending, we describe only how to send data to an applet, the servlet opens an output stream using the response object. The servlet must only realize 4 steps to send serialized data to an applet.

1.) Set the content type for sending a seralized data object
```
String contentType = "application/x-java-serialized-object";
response.setContentType(contentType);
```
2.) Create an ObjectOutputStream
```
ObjectOutputStream out = new ObjectOutputStream(response.getOutputStream());
```
3.) Write data with *writeObject* into the output stream
```
out.writeObject(new String(<APNN description>));
```
4.) The stream must be vacated. This ensures that the client get the complete data
```
out.flush();
```

Clearly, in addition to communication, the servlet implementation causes more work and code to establish the required functionality. Especially the control of the operations needs more considerations, for instance, the selection of analysis methods, its start, termination detection and return of results since the different analysis tools of the APNN toolbox are independent executables that are implemented in C and perform as independent processes. A sophisticated servlet implementation needs to take care of security, privacy, sharing among different users and applets; it need to take care of the usage of disc space, memory and cpu to avoid misuse and to possibly report on failures.

5 Conclusion

We briefly present the internet version of APNNed which supports loading, storing, and analysis of Petri nets via an HTTP server, so that APNNed can run as an applet in a general purpose web browser. Instead of describing the rather unchanged usage and appearance of APNNed, we describe how the necessary applet-servlet communication (HTTP-tunneling) can be realized in Java. Clearly, the current implementation covers a lot more details, for instance, transferring hierarchical nets implies to send a set of nets and information on its connectivity instead of a single net description. The current status of APNNed and the APNN toolbox is available at [1]. Ongoing work considers asynchronous interaction with time consuming analysis tools and further extensions to allow for a secure usage in an open environment.

References

[1] http://ls4-www.cs.uni-dortmund.de/qm/werkzeuge.html.

[2] F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of DEDS. *Quantitative Evaluation of Comp. and Comm. Sys.*, pages 356–359, Springer LNCS 1469, 1998.

[3] F. Bause, P. Kemper, and P. Kritzinger. Abstract Petri net notation. *Petri Net Newsletter*, 49:9–27, 1995.

[4] P. Buchholz. Hierarchical structuring of superposed GSPNs. In *Proc. 7th Int. Workshop Petri Nets and Performance Models (PNPM'97), St-Malo (France), June 1997*, pages 81–90. IEEE CS Press, 1997.

[5] P. Buchholz and P Kemper. Efficient computation and representation of large reachability sets for composed automata. Forschungsbericht 705, Fachbereich Informatik, Universität Dortmund (Germany), 1999.

[6] P Buchholz and P. Kemper. Modular state level analysis of distributed systems - techniques and tool suppport. In W.R. Cleaveland, editor, *Proc. 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*. Springer, LNCS 1579, 1999.

[7] P. Buchholz, P. Kemper, and APNNed group. APNNed - a net editor and debugger within the APNN Toolbox. *In J. Desel, P. Kemper, E. Kindler, and A. Oberweis, editors, 5. Workshop Algorithmen und Werkzeuge für Petri Netze*, Universität Dortmund, Fachbereich Informatik, Forschungsbericht Nr. 694:19–24, 1998.

[8] David Flanagan. *Java in a nutshell: A Desktop Quick Reference.* O'Reilly, 3 edition, Nov 1999.

[9] David Flanagan, Jim Farley, William Crawford, and Kristopher Magnusson. *Java Enterprise in a nutshell.* O'Reilly, May 1999.

[10] Marty Hall. *Core Servlets and Java Server Pages.* Prentice Hall, June 2000.

[11] P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Trans. on Software Engineering*, 22(9), Sep 1996.

[12] P. Kemper and R. Lübeck. Model checking based on kronecker algebra. Forschungsbericht 669, Fachbereich Informatik, Universität Dortmund (Germany), 1998.

[13] J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized Petri net. *In C. Girault and W. Reisig, editors, Application and Theory of Petri Nets*, Informatik Fachberichte 52, 1982.