The BRITNeY Suite Animation Tool

Michael Westergaard and Kristian Bisgaard Lassen

Department of Computer Science, University of Aarhus,
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark,
Email: {mw,k.b.lassen}@daimi.au.dk

Abstract. This paper describes the BRITNeY suite, a tool which enables users to create visualizations of formal models. BRITNeY suite is integrated with CPN Tools, and we give an example of how to extend a simple stop-and-wait protocol with a visualization in the form of message sequence charts. We also show examples of animations created during industrial projects to give an impression of what is possible with the BRITNeY suite.

1 Introduction

Colored Petri nets (CP-nets or CPN) [7] have proved their usefulness in modeling and understanding complex systems [2,10,12,19], e.g., for verification of existing behavior or requirements engineering of needed behavior.

However, when using CP-nets, only people familiar with the formalism are able to truly understand the model of the system. A domain expert may understand a CP-net, when introduced to CP-nets in general and when the particular CP-net is explained by the model developer, but the domain expert is seldom able to talk back, say precisely what is wrong with the model, and offer suggestions to fix the model of the system, because of lack of technical expertise with the formalism. CP-net models of systems are prone to errors if they can not be fully understood and validated by someone with domain knowledge. The contribution of the BRITNeY[1] suite animation tool is to give a visualization of the state and actions of a CP-net so the domain expert can validate the model.

In this paper we present the BRITNeY suite [23] which introduces an animation layer for CP-nets. BRITNeY suite provides a uniform way to implement, integrate, and deploy visualizations of CP-nets and has a pluggable architecture which makes it possible to write customized plug-ins to animate the model in addition to more than a dozen predefined plug-ins. The BRITNeY suite has already been used successfully to animate a network protocol [11], to animate a workflow process in a bank for the purpose of requirements engineering [8] and to visualize how patient, nurse and doctor work together with a system that dispenses sedatives, again for the purpose of requirements engineering [13].

Even though BRITNeY suite is designed with CPN Tools in mind, it is possible to integrate the tool with any executable formalism as the interface to

[1] An abbreviation for **B**asic **R**eal-time **I**nteractive **T**ool for **Ne**t-based animation.

CPN Tools is based on well-known public standards. For example the tool has been used successfully to visualize the execution of a timed automaton [1] model as well as the reachability graphs of systems created using a subset of the π-calculus [17], bigraphical reactive systems [15], finite and timed automata, and Coloured Petri nets. Also goto-graphs of Java programs have been visualized using the BRITNeY suite.

The paper is structured as follows. In Sect. 2 we give a brief overview of the architecture of the BRITNeY suite. In Sect. 3 we demonstrate how to add a message sequence chart visualization to a CP-net of a simple stop-and-wait protocol. Sect. 4 contains some example visualizations created as part of industrial projects. In Sect. 5 we mention related work and outline some of the new features planned for BRITNeY suite.

2 Architectural Overview

A well-known design pattern from the object-oriented world is the model-view-controller (MVC) design pattern [6]. In the MVC design pattern, three participants collaborate to provide the implementation of an application, namely a model, a view, and a controller, see Fig. 1. The model contains the state of the system, the view is a (graphical) representation of the current state of the model, and the controller implements the behavior of the system. The view may initiate actions in the controller.

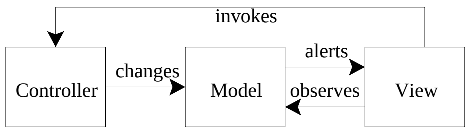


Fig. 1. Architectural overview of the model-view-controller design pattern.

The idea behind the BRITNeY suite is to use a CP-net (or any other formal executable model) to model the state and behavior of the system (the model and controller), and use BRITNeY suite for visualizing the system (view). This division is natural as places of CP-nets are used to model the state of a system and transitions the behavior.

In Fig. 2, we see how BRITNeY suite is integrated with CPN Tools [3] to provide simulation-based visualizations and animations. CPN Tools itself is split into two components, an editor and a simulator. The animation tool, in the right part of the figure, communicates with CPN Tools using a standard Remote Procedure Call protocol, called XML-RPC [25], in order to allow vendors of other tools to directly integrate their tools with BRITNeY suite. BRITNeY suite uses plug-ins to make the actual visualizations, which makes it easy to create your own animations. 15 plug-ins are currently available in the tool. Table 1 lists each plug-in with a short description. Over time, more plug-ins will be added.

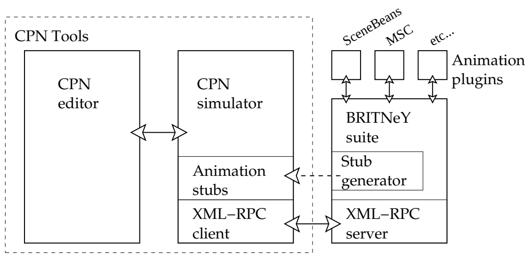


Fig. 2. A more detailed view of the integration of the animation tool with CPN Tools.

Table 1. Plug-ins for the BRITNeY suite. The first group of plug-ins is for creating various charts, the second group is for displaying directed graphs, the third group is for interacting with a user and the final group contain plug-ins that do not fit in any group.

Name	Description
AreaChart	For visualizing data values by filling the area below them
GanttChart	For drawing Gantt charts
Histogram	For drawing histograms
MSC	For drawing message sequence charts
PieChart	For drawing pie charts
PieChart3D	For drawing 3D pie charts
StepChart	Similar to a histogram
XYChart	For visualizing data values as points
Graph	For drawing 2D graphs
Graph3D	For drawing 3D graphs
GetString	For getting short text-messages from the user
ShowString	Display short text-messages to the user
DataStore	Storage for simple data-types
Report	Nice presentation of data
SceneBeans	For displaying and interacting with a SceneBeans [20] animation

BRITNeY does not contain a fixed set of plug-ins as plug-ins can be added and removed, so stubs are generated on-the-fly as needed by using the reflection mechanism in Java to inspect the signatures of the plug-ins. The stubs make sure that values are passed correctly to the appropriate Java object's method and takes care of passing the return value back to the caller. Stubs are generated automatically by the stub generator component of BRITNeY suite. The stubs are injected into CPN Tools and are available as regular functions in the inscription language of CPN Tools, namely Standard ML (SML) [16], which allows the modeler to use the animation plugins anywhere SML expressions are allowed.

The modeler will often want to update the visualization when a transition occurs. This is done by calling the stubs in code segments that are special transition inscriptions allowed by CPN Tools. A code segment is executed when the transitions it belongs to occurs. It consists of input, output, and action parts. The input and output parts make it possible to receive input from the model and to provide situmli back to the model respectively. This makes it possible to, e.g., invoke a stub with values dictated by tokens and to generate new tokens from the result of executing the stub.

3 Using BRITNeY to Generate Message Sequence Charts

In this section we will describe how to show a simulation of a CP-net as a message sequence chart (MSC), i.e. generate a chart which displays the simulation of the CP-net in terms of events being passed between processes. This is instead of, e.g., in CPN Tools where simulation is shown as enabling of transitions, and tokens being consumed and generated when transitions occur. The description is fairly high-level, and a more detailed and technical description can be found in [24], but the reader is assumed to have basic knowledge of object oriented programming and an ability to read Java and SML code.

3.1 Model

The model that we will use in this paper is a very simple stop-and-wait protocol as seen in Fig. 3. The model consists of three parts: 1) A sender who can Send Data from the Out Buffer with a packet number from Next Id. Also the sender can Receive Ack thereby updating the token on Next Id. 2) A network that can Drop packets that are sent to the receiver from place Network 1. Network 2 contains acknowledgments that the receiver is sending back to the sender. 3) The receiver can Receive Data and update the Receive Id that the next packet must have.

3.2 Adding the MSC primitives in CPN Tools

MSCs are well-known to protocol engineers, and it is therefore a good idea to be able to present the execution of a CP-net as an MSC. The first part of an MSC that is generated from the model in Fig. 3 can be seen in Fig. 4. The Sender process corresponds to the sender part of the CP-net, Network process to the network part of the CP-net and Receiver process to the receiver part of the CP-net. In the following we will describe how to extend the model in Fig. 3 with primitives to draw this MSC.

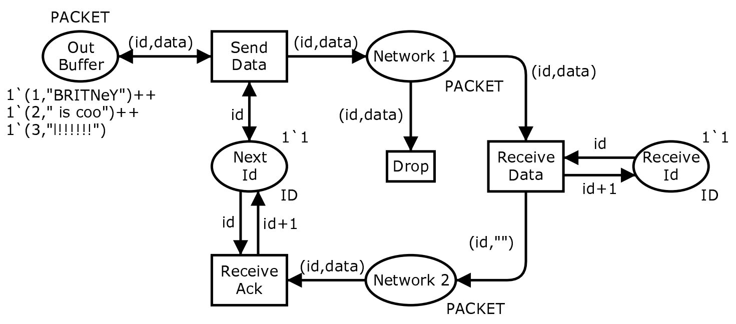


Fig. 3. CP-net of a stop-and-wait protocol.

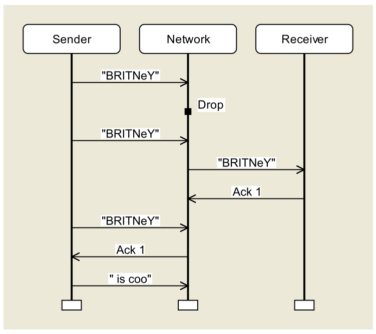


Fig. 4. First part of an MSC generated from the model in Fig. 3

In Listing 1 we show the signature of the Java plug-in for the MSC class. It contains functions for adding new process, adding events between processes, and adding events internal to a single process. This will, as explained in Sect. 2, be translated, by the stub generator, to a corresponding SML representation. In the following we show how to apply SML primitives to the CP-net to call these methods.

Listing 1 Java signature of the MSC object.

```
void addProcess(String name);
void addEvent(String from, String to, String name);
void addInternalEvent(String process, String name);
```

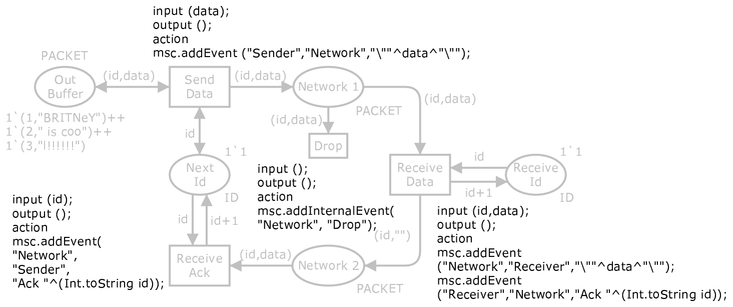
To set up the MSC view we need to add some declarations to the CP-net. In CPN Tools we add declarations as in Listing 2. Line 1 initializes an MSC object with the name "Stop-and-Wait Protocol". Lines 2–4 creates the three processes as seen in Fig. 4; i.e. Sender, Network, and Receiver.

Listing 2 Initialization of the MSC view.

```
structure msc = MSC(val name = "Stop-and-Wait Protocol");
val _ = msc.addProcess("Sender");
val _ = msc.addProcess("Net");
val _ = msc.addProcess("Receiver");
```

Next we need to extend our model from Fig. 3 to generate the events that correspond to those in Fig. 4.

In Fig. 5 we see how the methods from Listing 1 are incorporated into the CP-net. The idea is that we want to generate an event in the MSC when one of the transitions in the model occurs. We did this as follows: When Send Data in the CP-net occurs we add an event from Sender to Network in the MSC, where the label is the same as the data being sent, i.e. "data" where data is bound from the string in the packet from Out Buffer. When Drop in the CP-net occurs we add the internal event Drop on the process Network in the MSC. When Receive Data in the CP-net occurs, an event is added from Network to Receiver in the MSC, with label stating what data is received (the label is "data", where data is bound from the string in the packet from Network 1) and also, an event from Receiver to Network in the MSC, with an acknowledgment with the received packet number as label; the label is Ack i where i is the integer in the packet bound in the occurence of Receive Data. Finally, when Receive Ack occurs, an event is sent from Network to Sender in the MSC with the acknowledgment as the label; here the label is again Ack i.



```
input (data);
output ();
action
msc.addEvent ("Sender","Network","\""^data^"\"");
```

```
input ();
output ();
action
msc.addInternalEvent(
"Network", "Drop");
```

```
input (id);
output ();
action
msc.addEvent(
"Network",
"Sender",
"Ack "^(Int.toString id));
```

```
input (id,data);
output ();
action
msc.addEvent
("Network","Receiver","\""^data^"\"");
msc.addEvent
("Receiver","Network","Ack "^(Int.toString id));
```

Fig. 5. Model from Fig. 3 with MSC primitives

4 Visualization Examples

This section will give a number of examples of practical use of BRITNeY suite. We will not describe the examples in detail, but just refer to papers with detailed descriptions.

In Fig. 6, we see an animation created to visualize an interoperability protocol for mobile ad-hoc networks [11]. The protocol is used to ensure that the mobile ad-hoc nodes (the laptops) can communicate with the stationary host, even when on the move. The domain-specific GUI makes it possible for the user to observe the behavior of the system as packets, visualized by colored dots, flow along the network and to provide stimuli to the protocol by dragging and dropping the laptops to indicate the node movements. The use of an underlying formal model can be completely hidden when experimenting with the prototype. The domain-specific GUI has been used in the project both internally during protocol design and externally when presenting the designed protocol to management and protocol engineers not familiar with CPN modeling.

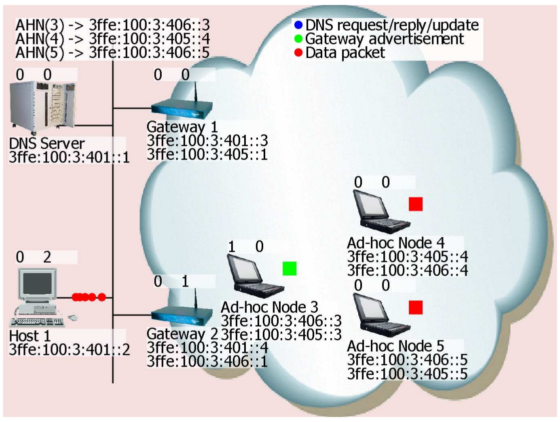


Fig. 6. A visualization of an interoperability protocol for mobile ad-hoc networks.

In Fig. 7 we see the domain specific animation based on the SceneBeans plug-in. This was used in [8] for requirements engineering of a new workflow system. The goal of the workflow was to support the handling of a blanc loan applications.

The animation is constructred as follows: There are always two bank assistants, Ann and Bill. Up to two customers can be present, in the figure only Mr. Smith is present. A bank manager, Mr. Banks, is always present. The balls represent blanc loan requests and the position of it shows who is responsible for the request. Whenever a transfer of responsibility occurs in the CP-net the ball is moved from one person to another in the animation. One ball has a `P` on it. This means that it is suspended, or parked, but can be picked up by one of the bank employees when they have the time. The square is part of the animation interface.

Once in a while the user can interact with the animation by e.g., setting up a loan for the customer when he wants to make a loan request, or setting the status of a loan request on behalf of Ann, Bill or Mr. Banks to e.g. granted or rejected. By not making the animation look like a normal prototype with windows, menues etc., the focus of the user was on the workflow and not on how the interface of the future system should be like.

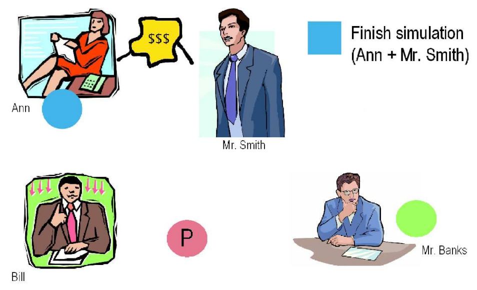


Fig. 7. SceneBeans animation used for requirements engineering.

5 Related Work and Future Improvements

BRITNeY suite supports adding animations to CPN models by annotating transitions with function calls, which are executed whenever the transition occurs. In the following, we outline how a number of other modeling tools facilitate visualization.

ExSpect [21], a tool for modeling based on CP-nets, allows the user to view the state by associating widgets with the state of the model, and to asynchronously interact with the model, also using simple widgets. In this way, it is easy to create simple user interfaces that support displaying information, but support for creating more elaborate animations is not readily available.

Mimic/CPN [18] makes it possible to animate models within Design/CPN [4], which is another tool for modeling using CP-nets. CPN models are animated by Mimic/CPN by using function calls that are executed whenever a transition of the CP-net occurs. The animations are drawn using an application that resembles traditional drawing programs. Input from the user is possible by showing a modal dialog, where the simulation of the model is stopped while the user is expected to input information. It is also possible to make click-able regions, and the model can then query if one of these has been clicked. Another approach, which is taken by the Comms/CPN [5] library for Design/CPN and CPN Tools, is to provide a TCP/IP abstraction, allowing the user to code the user interface in any language and use RPC to communicate with it.

LTSA [14], a tool for modeling using timed labeled transition systems, allows users to animate models using the SceneBeans library. In LTSA animations are tied to the models by associating each animation activity with a clock; resetting a clock corresponds to starting an animation sequence. The animation sequence or a user with his mouse can then send events which correspond to the progress of the timer.

PNVis [9] is an add-on for the Petri Net Kernel [22], a highly modular tool for editing Petri nets. PNVis associates tokens with 3D objects and certain places with locations in a 3D world. Moving tokens corresponds to moving the associated object in the 3D world. PNVis is suitable for modeling physical systems, but not so applicable for creating prototypes of software or requirements engineering.

Using some of these animation tools/libraries, animation is integrated with the modeling formalism, such as the use of timers in LTSA or the ability to view or change the marking of places in ExSpect. Some libraries are easy to extend, such as animations in LTSA, as the SceneBeans library allows users to easily extend it with new animation primitives. Also, animations created using COMMS/CPN can easily be extended, as the "animation" is just a custom (e.g. Java) application. Some libraries make it easy to design animations, such as ExSpect and MIMIC/CPN, which both provide a graphical user interface to design animations. The approach of the current version of BRITNeY suite resembles a combination of MIMIC/CPN and COMMS/CPN, as the animation is driven by function calls associated with transitions to an external application. The main feature offered by BRITNeY suite from a user point of view is thus compatibility with CPN Tools (rather than the discontinued DESIGN/CPN) and platform-independence. BRITNeY suite also makes it easy to extend the tool using simple Java classes. From a developer point of view, BRITNeY provides good foundations for allowing closer integration with the model by allowing parts of the animation to inspect and modify tokens on fusion places of the CPN model, much like how widgets are associated with places in ExSpect. This is an important part of future work.

An important new feature of BRITNeY suite is that it is possible to deploy animations in a way that allows even non-technical users to download and experiment with the animation. Another part of the future work is to make this process even easier by adding a wizard to take care of all the details.

BRITNeY suite has already proven itself useful in real projects, and has already been used in several industrial projects.

References

1. J. Bengtsson and W. Yi. Timed Automata: Semantics, Algorithms and Tools. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *LNCS*, pages 87–124. Springer-Verlag, 2004.
2. C. Bossen and J.B. Jørgensen. Context-descriptive prototypes and their application to medicine administration. In *DIS '04: Proc. of the 2004 conference on Designing interactive systems*, pages 297–306, Boston, MA, USA, 2004. ACM Press.
3. CPN Tools. Online `www.daimi.au.dk/CPNTools/`.
4. Design/CPN. Online `www.daimi.au.dk/designCPN/`.
5. G. Gallasch and L.M. Kristensen. A Communication Infrastructure for External Communication with Design/CPN. In *Proc. of Third CPN Workshop*, volume PB-554 of *DAIMI*, pages 79–93, 2001.

6. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
7. K. Jensen. *Coloured Petri Nets—Basic Concepts, Analysis Methods and Practical Use. Volume 1: Basic Concepts*. Springer-Verlag, 1992.
8. J.B. Jørgensen and K.B. Lassen. Aligning Work Processes and the Adviser Portal Bank System. In *REBNITA05*, 2005.
9. E. Kindler and C. Páles. 3D-Visualization of Petri Net Models: Concept and Realization. In *Proc. of ICATPN 2004*, volume 3099 of *LNCS*, pages 464–473. Springer-Verlag, 2003.
10. L.M. Kristensen and K. Jensen. Specification and Validation of an Edge Router Discovery Protocol for Mobile Ad-hoc Networks. In *Integration of Software Specification Techniques for Application in Engineering*, volume 3147 of *LNCS*, pages 248–269. Springer-Verlag, 2004.
11. L.M. Kristensen, M. Westergaard, and P.C. Nørgaard. Model-based Prototyping of an Interoperability Protocol for Mobile Ad-hoc Networks. Accepted for Fifth International Conference on Integrated Formal Methods, 2005.
12. L. Lorentsen, A-P Tuovinen, and J. Xu. Modelling Features and Feature Interactions of Nokia Mobile Phones Using Coloured Petri Nets. In *Proc. of ICATPN 2002*, volume 2360 of *LNCS*, pages 294–313, 2002.
13. R.J. Machado, K.B. Lassen, S. Oliveira, M. Couto, and P. Pinto. Execution of UML Models with CPN Tools for Workflow Requirements Validation. In *Proc. of Sixth CPN Workshop*, volume PB-576 of *DAIMI*, pages 231–250, 2005.
14. J. Magee and J. Kramer. *Concurrency – State Models and Java Programs*. John Wiley & Sons, 1999.
15. R. Milner. Bigraphical Reactive Systems. In K.G. Larsen and M. Nielsen, editors, *Proc. of CONCUR 2001*, volume 2154 of *LNCS*, pages 16–35. Springer-Verlag, 2001.
16. R. Milner, R. Harper, and M. Tofte. *The Definition of Standard ML*. MIT Press, 1990.
17. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes. *Information and Computation*, 100(1):1–77, 1992.
18. J.L. Rasmussen and M. Singh. *Mimic/CPN. A Graphical Simulation Utility for Design/CPN. User's Manual*. `www.daimi.au.dk/designCPN`.
19. J.L. Rasmussen and M. Singh. Designing a Security System by Means of Coloured Petri Nets. In *Proc. ICATPN 1996*, volume 1091 of *LNCS*, pages 400–419. Springer-Verlag, 1996.
20. SceneBeans. Online `www-dse.doc.ic.ac.uk/Software/SceneBeans`.
21. The ExSpect tool. `www.exspect.com`.
22. M. Weber and E. Kindler. The Petri Net Kernel. In *Petri Net Technologies for Modeling Communication Based Systems*, volume 2472 of *LNCS*, pages 109–123. Springer-Verlag, 2003.
23. M. Westergaard. BRITNeY suite website. Online `wiki.daimi.au.dk/tincpn/`.
24. M. Westergaard and K.B. Lassen. Building and Deploying Visualizations of Coloured Petri Net Models Using BRITNeY animation and CPN Tools. In *Proc. of Sixth CPN Workshop*, volume PB-576 of *DAIMI*, pages 119–136, 2005.
25. D. Winer. XML-RPC Specification. `www.xmlrpc.org/spec`.