Petri-net Based Animation with CPN Tools and BRITNeY

Michael Westergaard
mw@daimi.au.dk

Department of Computer Science
University of Aarhus
High-level graphics makes it easier to demonstrate and communicate models
Overview

- Architecture
- Hello World
- Dining Philosophers
- Stop Signal
- A closer look at ShowModal
- Future work & conclusion
Overview

- Architecture ⇐
- Hello World
- Dining Philosophers
- Stop Signal
- A closer look at ShowModal
- Future work & conclusion
Overall Architecture

CPN Tools GUI

CPN Simulation Engine

Animation framework

Animation objects

DMO

XML-RPC

TCP/IP

Beta

Standard ML

Java
Why Java?

- Well-known by many computer scientists
- Well-suited for creating graphics
- A huge number of libraries already exist → it is easy to create even very complex animation objects
Overview

- Architecture
- Hello World ⇐
- Dining Philosophers
- Stop Signal
- A closer look at ShowModal
- Future work & conclusion
The purpose of this example is

- to introduce connections
- to see the ShowModal and GetString animation objects
- to see how the animation functions can be used
Hello World (2/3)

- We want to model part of a hotel
- We focus on the clerk at the counter
- When a guest enters the clerk asks for his name
- The clerk then greets the guest
- We notice that we have hard-coded the name of the guest
- We would rather allow the user to act as the guest
- ...for this we will use some simple standard functions
Hello World (3/3)

- We notice that we have hard-coded the name of the guest
- We would rather allow the user to act as the guest
- …for this we will use some simple standard functions
- We notice that we have hard-coded the name of the guest
- We would rather allow the user to act as the guest
- ...for this we will use some simple standard functions
In order to use the animation package, we must first set up a connection to an animation object.

In this example, we will add the declaration:

```scala
structure msg = ShowModalInstance(
    val name = "Message");
```

This can be thought of as creating a proxy object, `msg`, with an interface, `ShowModal`, in e.g. Java.
The interface of ShowModal is:

```plaintext
sig
    val displayMessage: string -> int
end
```

That is, we can call

```plaintext
msg.displayMessage("Hello World")
```

to show the message “Hello World” to the user.
CPN Tools allows code-fragments to be executed whenever a transition occurs.

Code-fragments have the syntax:

```
input (...)
output (...)
action
...
```
We can use code-fragments to tie the animation to our model:
We can use code-fragments to tie the animation to our model:
We can use code-fragments to tie the animation to our model:

```
input name;
output ();
action
msg.displayMessage("Hello " ^ name); ();
```
We create a connection

```ocaml
structure input =
  GetStringInstance(
    val name = "Question");

with the interface:

sig
  val getString: string * string -> string
end
```
…and use it in our model
...and use it in our model
So Far, We Have Seen… (1/3)

...how to create connections to an animation object:

```java
structure msg = ShowModalInstance(
    val name = "Message");
```
...2 animation object interfaces:

- **ShowModal**:

  ```plaintext
  sig
  val displayMessage: string -> int
  end
  ```

- **GetString**:

  ```plaintext
  sig
  val getString:
    string * string -> string
  end
  ```
...how to use connections in our nets using code-fragments:
input name;
output ();
action
msg.displayMessage("Hello " ^ name); ();
Overview

- Architecture
- Hello World
- Dining Philosophers ↔
- Stop Signal
- A closer look at ShowModal
- Future work & conclusion
The purpose of this example is

- to see the GraphSheet animation object
- to see how we can write small libraries using the animation functions
Dining Philosophers (2/2)

Think

Take Chopsticks

Eat

Put Down Chopsticks

Unused Chopsticks

Chopsticks(p)

PH.all()

CS.all()
We want to generate and draw the state-space.

For this, we will use `GraphSheet`:

```ocaml
sig
  val createVertex: string -> int
  val createEdge: string * string * string -> int

  val doLayout: unit -> int
  val export: unit -> int
end
```
The CPN Tools state-space tool provides a number of functions:

val EvalAllNodes: (Node -> 'a) -> 'a list
val EvalAllArcs: (Arc -> 'a) -> 'a list
val SourceNode: Arc -> Node
val DestNode: Arc -> Node
val st_Node: Node -> string
State-space Functions (2/2)

- SourceNode(a) = A
- DestNode(a) = B
- st_Node(A) = "A"
- EvalAllNodes(fn x => x) = [A, B, C]
- EvalAllArcs(fn x => x) = [a, b, c]
We assume a connection to a GraphSheet animation object named `graph`.

- Draw one node, A:
  ```
  graph.createVertex(st_Node(A))
  ```

- Draw an arbitrary node:
  ```
  fun drawNode n =
  graph.createVertex(st_Node(n))
  ```

- Draw all nodes:
  ```
  EvalAllNodes(drawNode)
  ```
Drawing all Nodes

We assume a connection to a GraphSheet animation object named graph

- **Draw one node, A:**
  ```
  graph.createVertex(st_Node(A))
  ```

- **Draw an arbitrary node:**
  ```
  fun drawNode n =
    graph.createVertex(st_Node(n))
  ```
We assume a connection to a GraphSheet animation object named graph.

- **Draw one node, A:**
  ```scala
graph.createVertex(st_Node(A))
```

- **Draw an arbitrary node:**
  ```scala```
  fun drawNode n =
    graph.createVertex(st_Node(n))
  ```scala```

- **Draw all nodes:**
  ```scala```
  EvalAllNodes(drawNode)
  ```scala```
Draw one arc, $a$, from $A$ to $B$:

```
graph.createEdge(
    st_Node(A), st_Node(B), ""
)```
## Drawing all Arcs

- **Draw one arc, a, from A to B:**
  ```java
graph.createEdge(
    st_Node(A), st_Node(B), ""
  )
```

- **Draw one arc, a:**
  ```java
graph.createEdge(
    st_Node(SourceNode(a)), st_Node(DestNode(a)), ""
  )
```
Drawing all Arcs

- Draw one arc, a:
  ```hljs
  graph.createEdge(
    st_Node(SourceNode(a)),
    st_Node(DestNode(a)), ""
  )
  ```

- Draw an arbitrary arc:
  ```hljs
  fun drawArc a =
    graph.createEdge(
      st_Node(SourceNode(a)),
      st_Node(DestNode(a)), ""
    )
  ```
Drawing all Arcs

- **Draw one arc, a:**
  ```
  graph.createEdge(
      st_Node(SourceNode(a)),
      st_Node(DestNode(a)), ""
  )
  ```

- **Draw an arbitrary arc:**
  ```
  fun drawArc a =
      graph.createEdge(
          st_Node(SourceNode(a)),
          st_Node(DestNode(a)), ""
      )
  ```

- **Draw all arcs:**
  ```
  EvalAllArcs(drawArc)
  ```
fun drawNode n =
  graph.createVertex(st_Node(n))

EvalAllNodes(drawNode)

fun drawArc a =
  graph.createEdge(
    st_Node(SourceNode(a)),
    st_Node(DestNode(a)), ""
  )

EvalAllArcs(drawArc)
The code does not depend on the net at all
We may want to draw state-spaces for other nets as well
→ creating a small library seems like a good idea
We will then be able to draw a state-space by issuing:
use("visualise.sml")
A slightly more elaborate library has been implemented

- Support for drawing only parts of a state-space
- Support for better names of nodes and arcs

Using this library, drawing a state-space is as simple as:

```haskell
use("visualise.sml");
NiceLabels();
DrawEntireGraph();
```
So Far, We Have Seen… (1/2)

...the GraphSheet animation object interface:

```ocaml
sig
  val createVertex: string -> int
  val createEdge: string * string * string -> int
  val doLayout: unit -> int
  val export: unit -> int
end
```
So Far, We Have Seen... (2/2)

...how to
- use GraphSheet with the state-space functions
- create a small library for drawing state-spaces, which can be used as:

```sml
use("visualise.sml");
NiceLabels();
DrawEntireGraph();
```
Overview

- Architecture
- Hello World
- Dining Philosophers
- Stop Signal ⇐
- A closer look at ShowModal
- Future work & conclusion
Stop Signal (1/3)

The purpose of this example is to introduce the SceneBeans animation object.

- Creating animation-description files
- Loading animation-description files
- Invoking commands in the animation
- Setting parameters in the animation
- Listening for events from the animation
A danish stop signal cycles between the colours:
- green
- yellow
- red
- yellow + red

We want to model this
We would like to visualise this...
SceneBeans (1/3)

- From the SceneBeans homepage\(^a\): “SceneBeans is a Java framework for building and controlling animated graphics... It is used in the LTSA tool to animate formal models of concurrent systems”

- SceneBeans was designed and implemented by Nat Pryce of Imperial College, London

\(^a\)http://www-dse.doc.ic.ac.uk/Software/SceneBeans/
SceneBeans (2/3)

- A SceneBeans animation is described using XML.
- Once an animation is started, it communicates with the surroundings using:
  - commands
  - events
- SceneBeans is written in Java... why not try to use it?
The interface of SceneBeans is:

```ocaml
class SceneBeans

  val setAnimation : string -> int
  val getNextEvent : unit -> string
  val peekNextEvent : unit -> string
  val waitForEvent : string -> int
  val hasMoreEvents : unit -> bool
  val setValue : string * string * string -> int
  val invokeCommand : string -> int

done
```

end
Definition of an XML File

- We create an XML file describing the animation of our model.
- Assuming we have a connection to a SceneBeans animation object named `lights`, we can load our XML file using:
  ```java
  lights.setAnimation("stop.xml");
  ```
Clearing all High-lights

- In the XML file, a clear command is defined.
- The clear command can be invoked by:
  \[ \text{lights.invokeCommand("clear")}; \]
High-lighting a Circle

- As the red, yellow, and green circles have an ID-attribute, we can change the value of their parameters.
- We can change the value of the color-parameter of the red circle by:

```javascript
lights.setValue(
    "red",
    "color",
    "ff0000");
```
Whenever a user clicks on the green circle, an event is sent

We can listen for events using:

```javascript
lights.getNextEvent();
```
So Far, We Have Seen...

...how to use the SceneBeans animation object; in particular how to:

- create animation-description files
- load animation-description files
- invoke commands in the animation
- set parameters in the animation
- listen for events from the animation
More on SceneBeans

- The Telebit example from the start of the talk is created using the SceneBeans animation object.
- Visit SceneBeans’ homepage[^1] for more information on writing animation-description files.

[^1]: http://www-dse.doc.ic.ac.uk/Software/SceneBeans/
Overview

- Architecture
- Hello World
- Dining Philosophers
- Stop Signal
- A closer look at ShowModal
- Future work & conclusion
A Closer Look at ShowModal

- It is easy to create an animation object:
  1. Create a Java class implementing a certain interface
  2. Generate appropriate ML code using rmicompiler
- Using JBuilder, a very rough outline of a dialog is created
- A couple accessor methods are added
- Using the rmicompiler, an ML interface is created
- In 15 minutes, a nice dialog can be used from a net
This way it is easy to get input and present output.

We use Petri-nets to “program” the control flow.

Combining this, we obtain executable prototypes in a very easy manner.
<table>
<thead>
<tr>
<th>Mimic/CPN</th>
<th>BRITNeY animation</th>
</tr>
</thead>
<tbody>
<tr>
<td>very general and low-level</td>
<td>encourages use of domain-specific,</td>
</tr>
<tr>
<td></td>
<td>high-level animation objects$^a$</td>
</tr>
<tr>
<td>animations can be designed using a GUI</td>
<td>(currently) no such feature</td>
</tr>
<tr>
<td>synchronous only</td>
<td>asynchronous features designed</td>
</tr>
<tr>
<td>extended by ML libraries</td>
<td>extended by ML libraries or by creating new animation objects in Java</td>
</tr>
</tbody>
</table>

$^a$But a number of quite general animation objects exist
Overview

- Architecture
- Hello World
- Dining Philosophers
- Stop Signal
- A closer look at ShowModal
- Future work & conclusion ⇐
Future Work

- Clean up and make 1st release
- Implementation of more animation objects
- New architecture
More Animation Objects

- Message sequence charts
- Charts (for use with e.g. the performance facilities)
- Report generator (to create nice simulation/state-space reports)
- Framework for Petri-net based rapid prototyping
- PNVis (Kindler & Páles: 3D-Visualization of Petri Net Models)
- . . .
Current Architecture

CPN Tools GUI
CPN Simulation Engine
Animation framework
Animation objects
DMO
XML-RPC
TCP/IP
Beta
Standard ML
Java

5th CPN Workshop – p. 53/56
New Architecture

CPN Tools GUI

CPN Simulation Engine

Animation framework

XML-RPC

TCP/IP

Beta Standard ML Java
Benefits of the New Architecture

- The simulation can be controlled better from the animation, by e.g. adding tokens to a place \(\rightarrow\) the animation can run without the CPN Tools GUI
- Simulation can be started and stopped from the animation package
- Certain animations may be shown directly in the CPN Tools GUI
- Certain animations may be defined in CPN Tools and exported to the animation tool
During this talk, we have seen:

- How to create connections to animation objects
- A number of different animation objects:
  - ShowModal
  - GetString
  - GraphSheet
  - SceneBeans
- How to use animations with nets:
  - in code-fragments
  - to create extension-libraries