

# Petri-net Based Animation with TIN-CPN

Michael Westergaard

`mw@daimi.au.dk`

Department of Computer Science  
University of Aarhus

# Motivation

- Demo

# Hello World (1/3)

- 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



# Hello World (2/3)

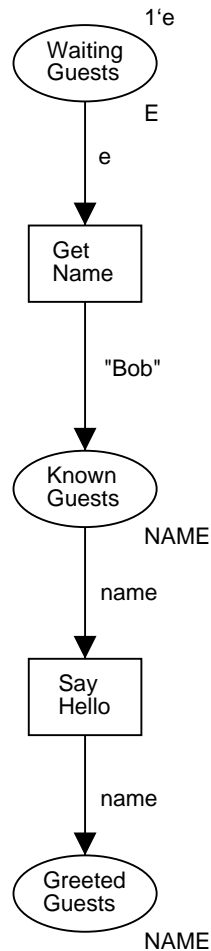
The purpose of this example is

- to introduce connections
- to get see the `ShowModal` and `GetString` animation objects
- to see how the animation functions can be used

# Animation Objects

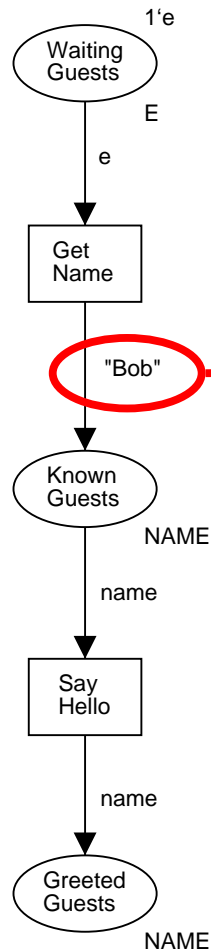
- An animation object provides some animation functionality
- During this talk, we will see animation objects for displaying messages and getting input from a user
- Yesterday, we saw animation objects for drawing message-sequence charts, drawing state-spaces and generating reports

# Hello World (3/3)



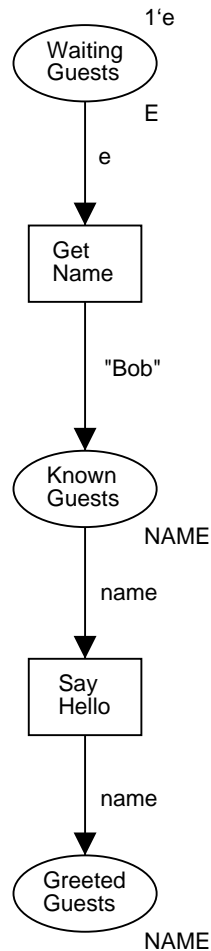
- We notice that we have hard-coded the name of the guest
- We would rather allow the user to act as the guest
- ...for this we will use some simple standard functions

# 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

# 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



# Setting up a Connection (1/2)

- 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

```
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

# Setting up a Connection (2/2)

- The interface of ShowModal is:

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

- That is, we can call

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

to show the message “Hello World” to the user

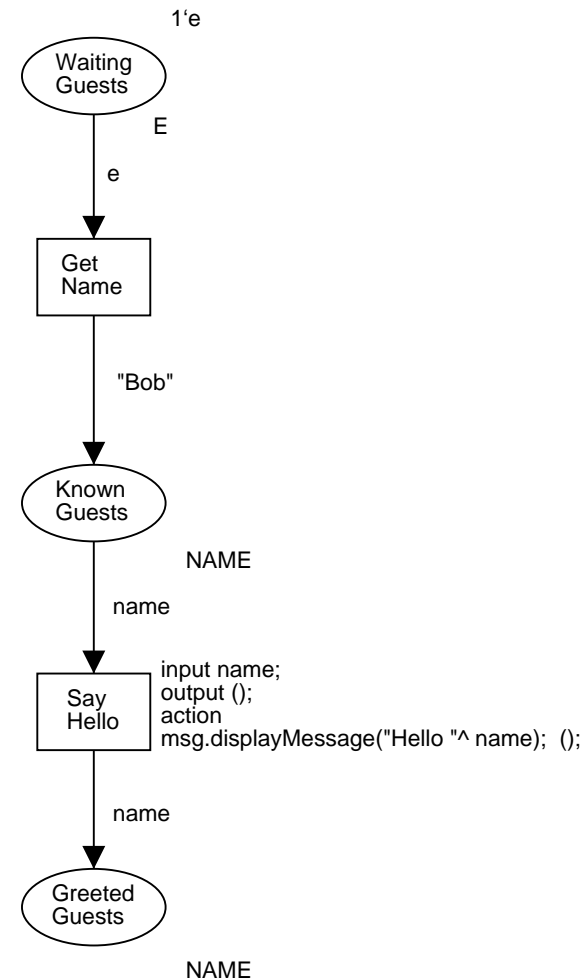
# Using Animation Functions (1/2)

- CPN Tools allows code-fragments to be executed whenever a transition occurs
- Code-fragments have the syntax:

```
input (...)
output (...)
action
...
```

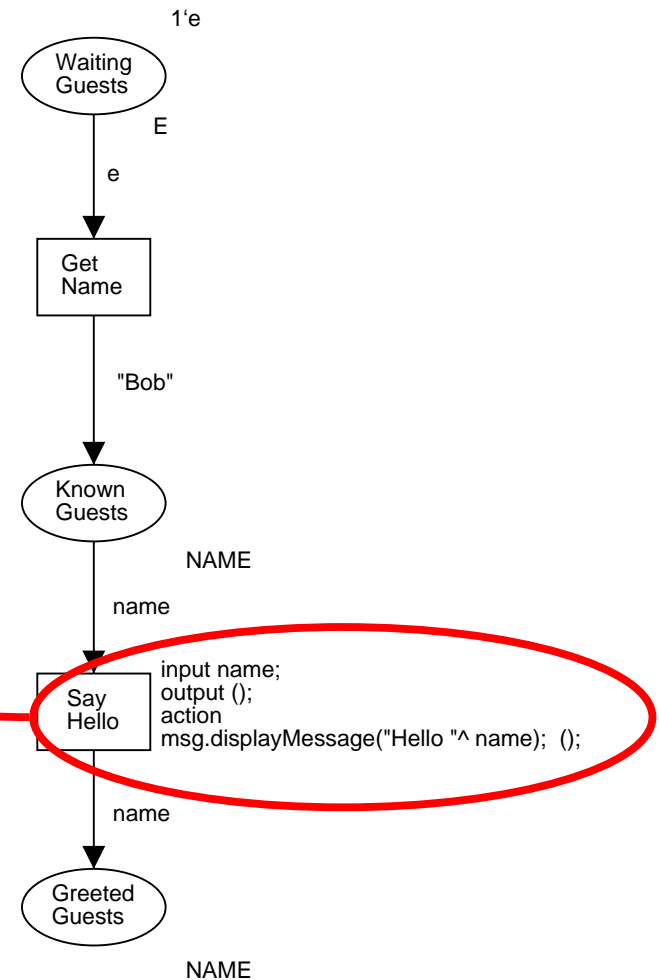
# Using Animation Functions (2/2)

- We can use code-fragments to tie the animation to our model:



# Using Animation Functions (2/2)

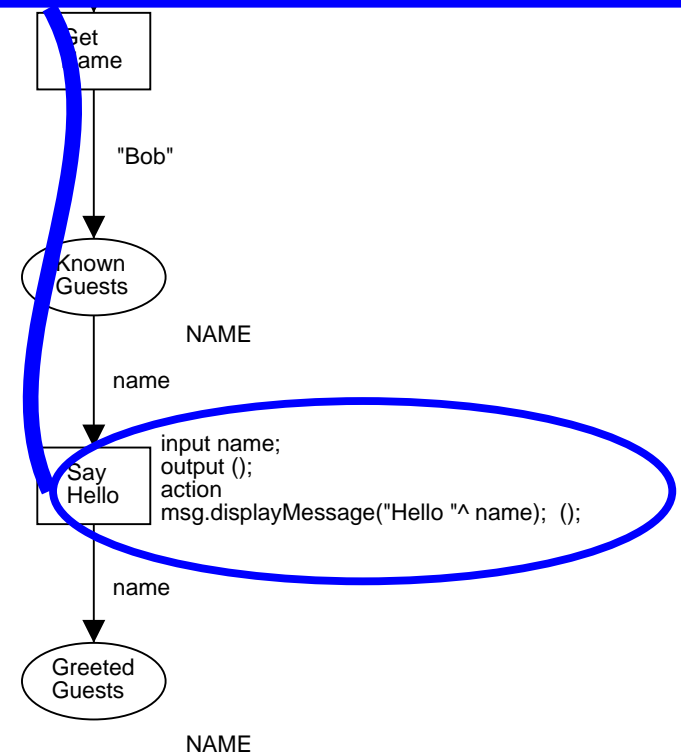
- We can use code-fragments to tie the animation to our model:



# Using Animation Functions (2/2)

```
input name;  
output ();  
action  
msg.displayMessage("Hello " ^ name); ();
```

- We can use code-fragments to tie the animation to our model:



# Asking for a Guest's Name (1/2)

- We create a connection

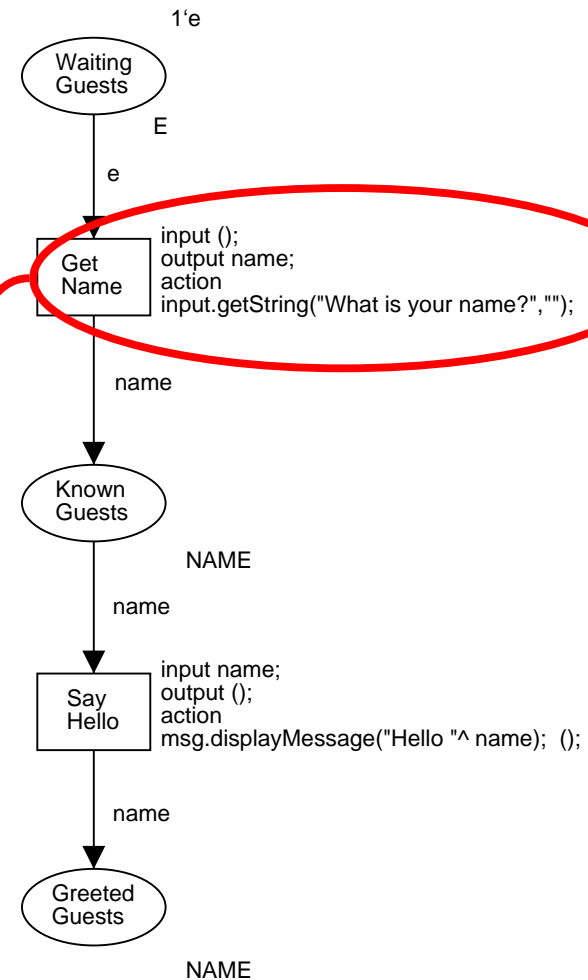
```
structure input =  
  GetStringInstance(  
    val name = "Question" );
```

with the interface:

```
sig  
  val GetString:  
    string * string -> string  
end
```

# Asking for a Guest's Name (2/2)

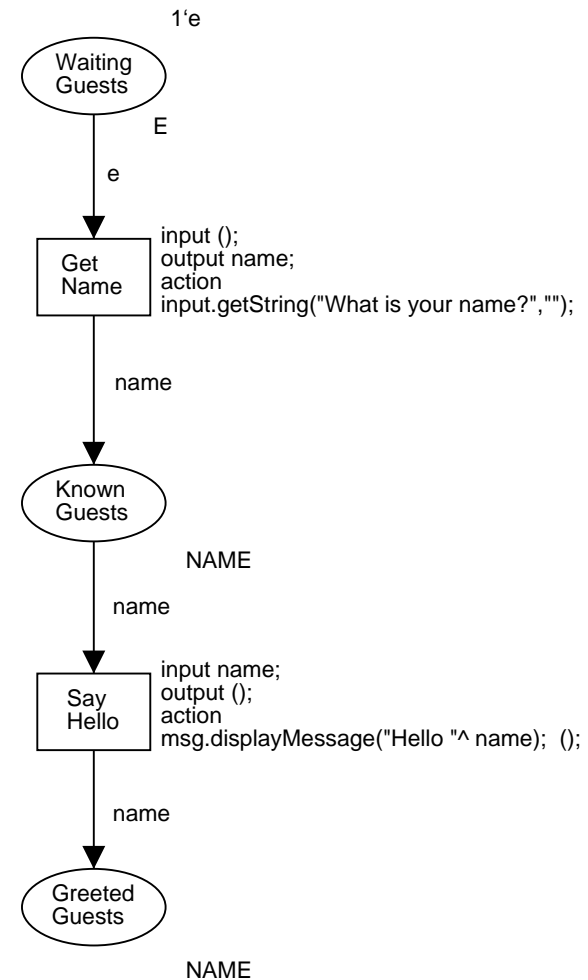
- ...and use it in our model



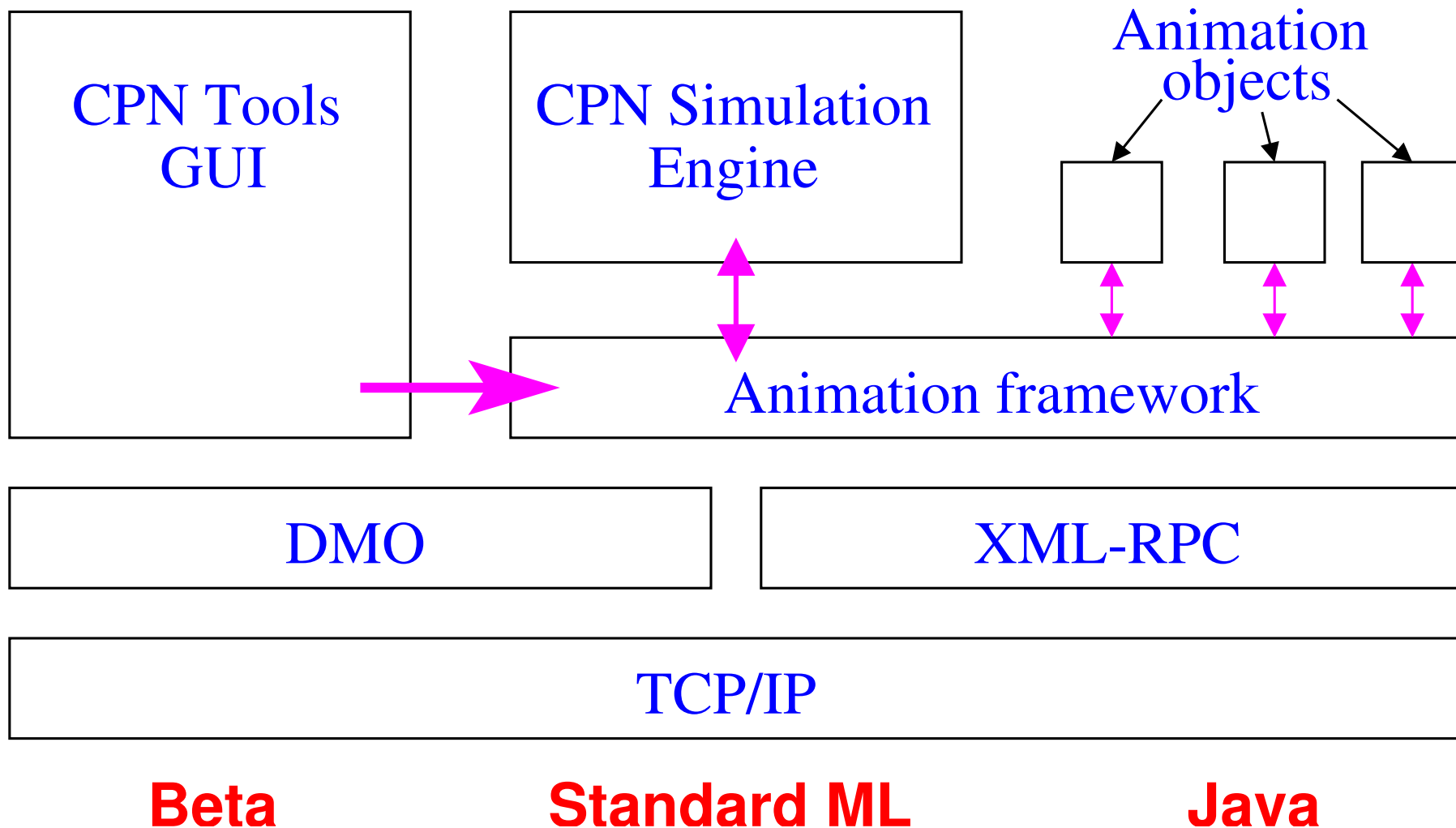


# Asking for a Guest's Name (2/2)

- ...and use it in our model



# Overall Architecture



# Why Java?

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

# TIN-CPN ↔ MIMIC/CPN

TIN-CPN	MIMIC / CPN
encourages use of domain-specific, high-level animation objects <sup>a</sup>	very general and low-level
animations can be designed using a text editor	animations can be designed using a GUI
asynchronous features designed	synchronous only
extended by ML libraries or by creating new animation objects in Java	extended by ML libraries

<sup>a</sup>But a number of quite general animation objects exist

# Summary (1/3)

During this presentation, we have seen how to create connections to an animation object:

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

# Summary (2/3)

...2 animation object interfaces:

- ShowModal:

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

- GetString:

```
sig
  val getString:
    string * string -> string
end
```

# Summary (3/3)

...how to use connections in our nets using code-fragments:

```
input name;
```

```
output ( );
```

```
action
```

```
msg.displayMessage( "Hello " ^ name ); (
```

# Future Work

- Make 1<sup>st</sup> release
- Implement asynchronous features