Lab 1

Objectives

Familiarize with Ptolemy II
Learn about signal generators
Learn about signal plotters
Learn about transfer functions in Laplace space
Work in the continuous time domain (CT)
Learn about transfer functions in Z space
Work in the synchronous data flow domain (SDF)

Rules

Answer the questions in this same document (in Word format). Paste screen snapshots when needed.

Send the final document by email to david.castells@uab.cat

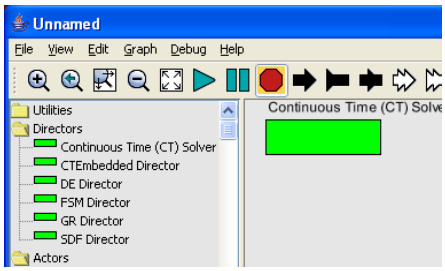
Step 1

Ptolemy II allows us to model systems with several computation models. We will start working with continuous time systems.

1.1 Create a new file (File/New/Graph Editor)

The computation models are choosen by selecting the appropiate director.

1.2 Choose the CT Director.



The modules that we will use in the models are called actors. There are three kind of actors: source actors, sink actors, and transformation actors.

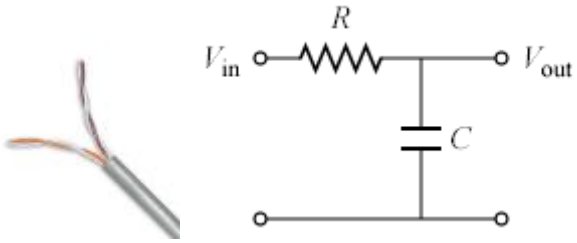
1.3 Generate a sinusoidal signal (Actors / DomainSpecific / ContinuousTime / Sources / ContinuousSinewave) of 440Hz and visualize it with (Actors / Sinks / TimedPlotter).

Question: Which values have you given to the different elements of your design to correctly visualize the sinusoidal signal ? by correctly I mean to avoid seeing a bunch of conected segments instead of a nice curve.

Step 2

We are very much interested in learn the ability of Ptolemy II to model communication systems.

We can think that a communications cable, in a very extremely simplified way, is just a RC circuit as the shown below:



Of course, we quickly identify this circuit as a low-pass filter with a cutoff frequency given by the following expression

$$F_c = \frac{1}{2\pi \cdot RC}$$

The transfer function in Laplace space is

$$H(S) = \frac{1}{1 + RC \cdot S}$$

2.1 Imagine that we have a standard cable with a resistance of 0.0921 Ω/m and a capacity of 5.472 pF/m

Question: Compute the cutoff frequency of a cable of 50 meters.

Question: Compute the cutoff frequency of a cable of 100 meters.

Question: Compute the cutoff frequency if we connect a MAX232 circuit to the opposite endpoint of the cable. Consider that the MAX232 has an input capacity of 90pF.

Step 3

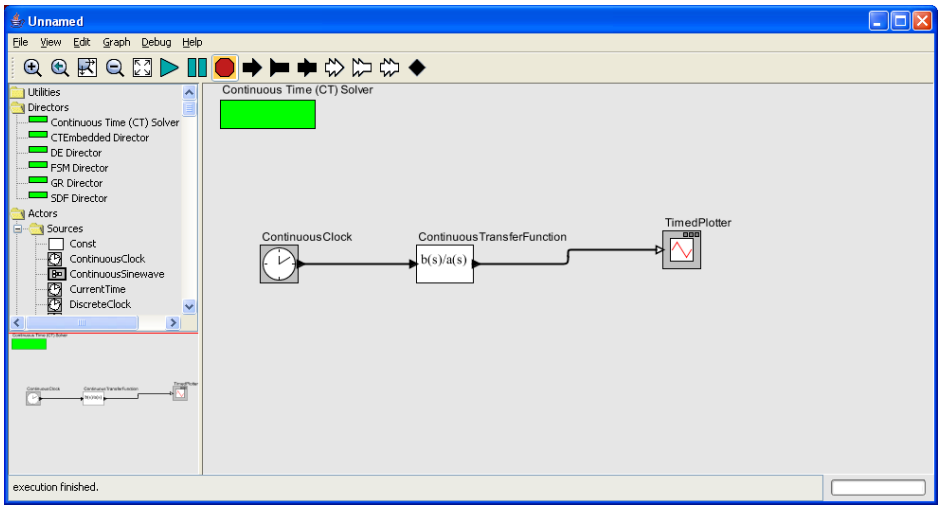
3.1 Create a new Ptolemy II with the continuous time domain.

3.2 Create a square signal (Actors / DomainSpecific / CotinuousTime / Sources / ContinuousClock) with a frequency of 50MHz

3.3 Create a transfer function in the continuous domain (Actors / DomainSpecific / CotinuousTime / Dynamics / ContinuousTransferFunction) equivalent to the cable model created in the previous point.

You have to consider that Ptolemy expects a expression of the form $\{a_n, ...a_1, a_0\}$ in the *numerator* and *denomitator* fields, which correspond to the coefficients of a polynomial $a_n \cdot S^n + ... + a_1 \cdot S^1 + a_0 \cdot S^0$

3.4 Add a plotter to view the result (Actors / Sinks / TimePlotter). Your system should look like the one depicted below



Question: What is the system response if we use the RC values of the 50m cable ?

Question: What is the system response if we use the RC values of the 100m cable ?

Step 4

The continuous models are not very used in modern electronics. Nowadays the signal processing is mostly done by digital processors that do not work with continuous signals but signals sampled at a given rate.

Ptolemy II is able to work in the discrete time domain. There are at least two computation models supporting this domain: *Synchronous Data Flow* (SDF) and its derivative *Discrete Time* (DT).

In the digital world there are no resistors and capacities. Nevertheless we want to model the behaviour of our simple cable in the digital world.

As you probably know in the discrete time domain the transfer functions are based on Z transform instead of Laplace transform. Fortunatelly there are methods to convert a function from the S space into the Z space, such as the bilinear transform.

However, we can find a way to build a digital model of a RC circuit in http://www.dsplog.com/2007/12/02/digital-implementation-of-rc-low-pass-filter/ .

$$H(Z) = \frac{kz^{-1}}{1+(1-k)z^{-1}}$$

$$k = \frac{dt}{RC}$$

4.1 Create a new Ptolemy II project with the *Synchronous Data Flow* model.

4.2 Create a square signal (Actors / Sources / SequenceSources / Sinewave + Actors / Math / Quantizer) of 50MHz having a sampling frequency of 1GHz

4.3 Create a transfer function in the discrete domain domini discret (Actors / SignalProcessing / Filtering / IIR) of the form H(Z).

Consider that expressions expected by Ptolemy II in *numerator* and *denomitator* fields are of the form $\{a_0,...,a_1, a_n\}$ that correspond to the coefficients of the polynomial $a_0 \cdot Z^0 + a_1 \cdot Z^{-1} + ... + a_n \cdot Z^{-n}$

Take an arbitrary value of k between 0 i 1

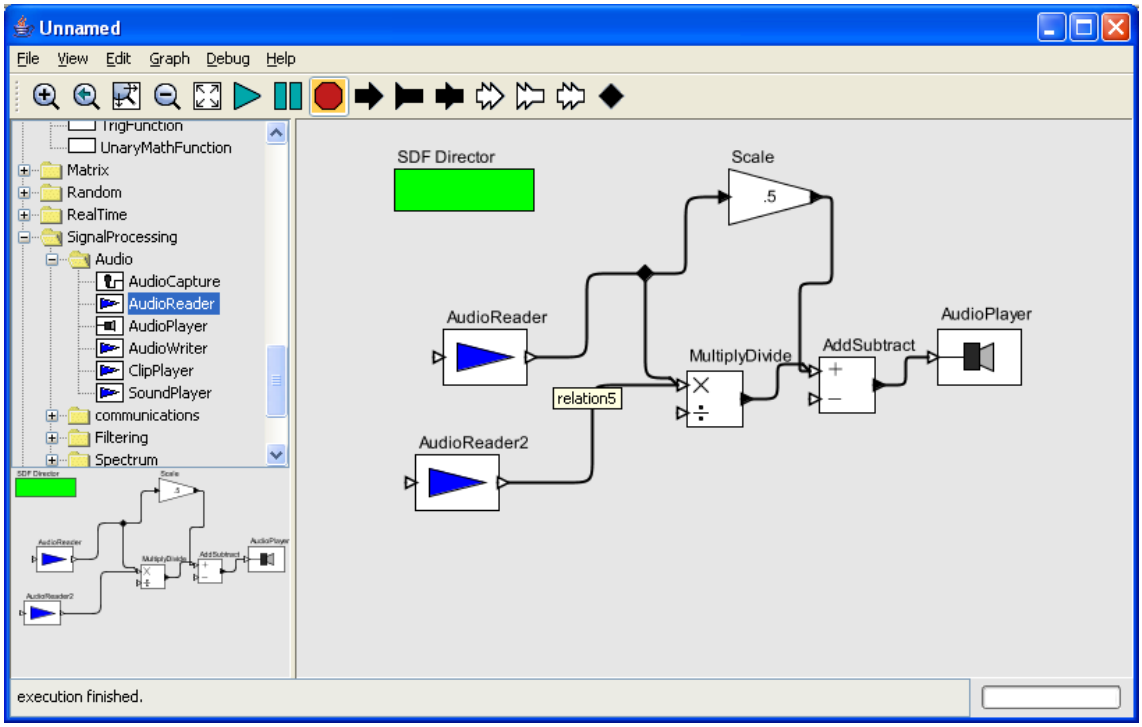
4.4 Add a signal viewer (Actors / Sinks / SequencePlotter).

Question: Which value of k gives a results equivalent to the continuous model for the 50m cable? Include a time diagram to illustrate this.

Question: Which value of k gives a results equivalent to the continuous model for the 100m cable? Include a time diagram to illustrate this.

Just for fun

If you have some spare time and want to enjoy a little bit discovering the power of Ptolemy II try to replicate the system shown below. You can configure the first *AudioReader* to take any Windows standard audio file that you will find in the folder (c:\windows\media) and configure the second *AudioReader* to use the voice message that comes by default. You will modulate the original signal resulting in a sound effect known as *Vocoder*.



Lab 2

Objectives

Familiarize with Ptolemy II
Learn configuration possibilities
Learn about math expressions
Learn about array handling
Learn about flow control methods
Work in the Synchronous Data Flow domain (SDF)
Work with sound card I/O and sound files.

Rules

Answer the questions in this same document (in Word format). Only write in response to a **Question** identified as such. Paste screen snapshots when needed.

Send the final document by email to david.castells@uab.cat

Introduction

As we saw in the first Lab, Ptolemy II allows to model systems in several computation domains.

Today we will try to create a communication system using the sound card of our PC and a modulation technique called Multiple Frequency Shift Keying (MFSK).

It is probable that our computers are not powerful enough to be able to perform the processing of the signals in real time, so we will use sound files as an intermediate step. So we will create a .WAV file instead of generating a sound in the loudspeakers. In the reception side we will read the saved file instead of using the microphone input.

Our modulation system will be based on encoding each symbol that we want to transmit as a different frequency.

Step 1

1.1 Create a new design using Ptolemy II Vergil.

1.2 Include the SDF director

1.3 Create a sinusoidal wave generator (Actors / Sources / SequenceSources / Sinewave) to create a 440Hz frequency with a sampling frequency of 8KHz. Rename the module using the context menu (Customize / Rename).

1.4 Connect the signal to a scope (Actors / Sinks / SequenceSinks / SequenceScope). The difference with a plotter, which we used in the first lab, is that the scope only shows a temporal window in a similar way as an oscilloscope. Put the same values in *width* and *persistence*.

Question: Which window width (in number of samples) do you have to set to be able to see 5 signal cycles without having a "dancing" signal ?

Step 2

Let's see how to modulate in FM

2.1 Take the previous design and add a second sinusoidal generator of 0.2 Hz. We will call it MODSOURCE.

We want to end up having a periodic signal MODSIGNAL generating values between 420 and 460 as shown by the following expression

$$MODSIGNAL(t) = 440 + 20 \cdot MODSOURCE(t)$$

2.2 Use the actors (Actors / Math / AddSubstract) and (Actors / Math / Scale) to generate the MODSIGNAL signal and connect it to the *frequency* input in the CARRIER module.

Question: Include the diagram of your resulting system.

2.3 Rreplace the Scope by an audio player d'audio (Actors / SignalProcessing / AudioPlayer). Make sure your loudspeaker is enabled and enjoy the produced sound for a while. Don't do it for too long, your colleagues will appreciate it.

Step 3

What we have done is known as FM modulation, now we will see how can we generate frequencies from discrete symbols.

But before, we should learn some things about string handling.

3.1 Crete a new SDF system using Vergil

3.2 Crete a string sequence (Actors / Sources / SequenceSources / Sequence) containing the strings shown below, and that repeats forever.

```
{  'A','S','E','C','R','E','T',  'z'}
```

3.3 Add a display (Actors / Sinks / GenericSinks / Display) to show the generated sequence.

Each character that we have used is really an string. As we want to generate a frequency from each uppercase letter, we must convert each string into a numeric value that

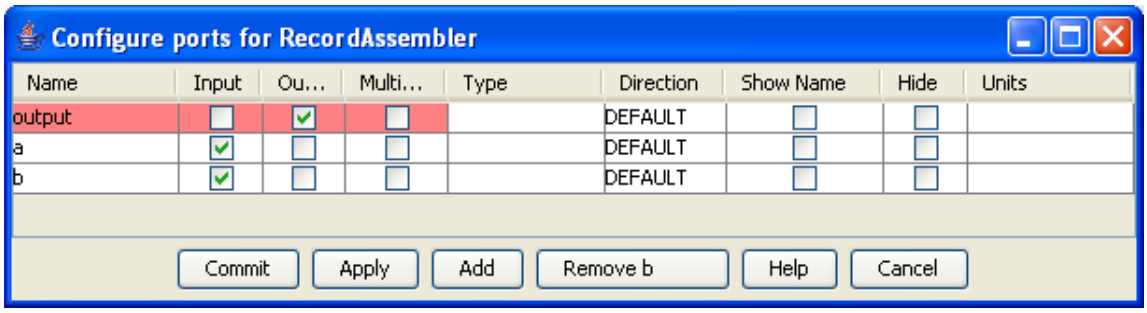
corresponds to the position of the letter in our alphabet. A into1, B into 2, etc. Later we will use this numeric value to create the frequency.

First we have to convert the string into a byte array. As we know that our string have one character, we will only take the first element.

3.4 Add a bifurcation point between the sequence generator and the display.

3.5 In a new path from the bifurcation point add a string to byte array converter (Actors / Conversions / StringToUnsignedByteArray) and then a selector of the first element of the array (Actors / Arrays / ArrayElement). You will need a constant value (Actors / Sources / GenericSources / Const), and notice that the first index of an array is 0.

3.6 In order to simultaneously view the sequencer and converter events we need an assembler (Actors / FlowControl / Aggregators / RecordAssembler). When you instantiate this element you will be surprised to see that it has no inputs, you have to add them by using the context menu Customize / Ports.



Add two inputs and connect them to the sequencer and ArrayElement outputs.

3.7 Connect the RecordAssembler output to the display.

Question: Include a diagram of your system

Question: What is the numeric value associated to each letter ?

3.8 Put a converter to double value double (Actors / Conversions / LongToDouble) before the RecordAssembler. The UnsignedByte is seldom used by Ptolemy, on the other hand Double is much more habitual.

3.9 We want to end up having a different frequency for each letter of the uppercase alphabet.

FREQKEY(lleter code) = (letter code-64)·100 + 100

Create an expression (Actors / Math /Expression) to implement the above formula. Consider that this actor has no input as well. You have to create the input and reference it in the formula.

3.10 Interpose this expression after the LongToDouble actor.

Question: Include the diagram of your system

Step 4

Our system never stops, but we would like to finish execution when the message end is reached.

4.1 Make the sequence avoid to cycle forever by disabling the *repeat* option.

Question: Why do you think that the program crashes ? Think that the rythm of production and consumption of tokens should be balanced in each side of a link.

4.2 You can make the sequence stop by comparing the sequence produced string with the value "z", which we have deliberately chosen to detect the end of the message. When this symbols is detected the system will stop. To implement this you will need the actors (Actors / Logic / Equals) and (Actors / FlowControl / ExecutionControl / Stop).

Question: Include a diagram of your system

Step 5

We are very close to complete our transmission system, we only need to inject the frequency value generated in step 3 to a sine wave generator. Unfortunately we do not have control of the production rate of the sequence element, and we must give the synthesis part some time (some cycles) to be able to create a sound.

5.1 Add a value repeater (Actors / FlowControl / SequenceControl / Repeat) to maintain the same value for an enough number of cycles (for instance 100K). Connect it to the output that produces the frequency value.

5.2 Add a sine wave generator (Actors / Sources / SequenceSources / Sinewave) with a sampling frequency of 22050 Hz.

5.2 Connect the sine wave to the loudspeaker (Actors / SignalProcessing / Audio / AudioPlayer) and simultaneously to a .WAV file writer (Actors / SingalProcessing / Audio / AudioWriter). You should name the audio file as secret.wav. Take care to use the same sampling frequency for this module, i.e. 22050 Hz.

5.3 Save your Ptolemy model as SecretSender.xml (File / Save As)

Question: Include a diagram of your system.

Step 6

Now we only need a receiver. You probably know that usually a receiver is more complex than a transmitter.

We need to detect the frequency from the incoming signal, and then recover its associated symbol, but how can we recover the frequency from a sampled incoming signal ? I mean, how can we build an FM demodulator ?

If we analyze a sine wave function f(t) and its derivative f'(t)...

$$f(t) = A \cdot \sin(2\pi ft)$$
$$f'(t) = A \cdot 2\pi f \cdot \cos(2\pi ft)$$

... we see that the derivative is proportional to the frequency, so we can make use of that. Consider that we are working in the discrete domain, so the actual derivative funcion that we will measure will be

$$f'(t) = \frac{\Delta f(t)}{\Delta t}$$

If we equal the discrete function with the continuous function of the derivative and isolate *f* we find

$$\frac{\Delta f(t)}{\Delta t} = A \cdot 2\pi f \cdot \cos(2\pi ft)$$
$$f = \frac{\Delta f(t)}{A \cdot 2\pi \cdot \cos(2\pi ft)\Delta t}$$

We can easily find a the maximum value of $\Delta f(t)$, it's just a matter of taking a window of values from the differences (big enough to capture a period from the frequency to detect) and find the maximum value. As we known that the maximum of any cosine function is 1, and that the Δt is the sampling period, we find...

$$f = \frac{Max(\Delta f(t))}{A \cdot 2\pi \cdot \cdot \Delta t} = \frac{Max(\Delta f(t))}{A \cdot 2\pi \cdot} f_s$$

However the receiver does not know the amplitude value of the original signal (A in out equation). No problem, because we can simply find it by computing the maximum of the *f(t)* function following the same previous reasoning, as we already known that the maximum value of any sinus function is 1.

$$A = Max(f(t))$$

So, let's go...

6.1 Create a new SDF model in Vergil.

6.2 Create an audio reader (Actors / SignalProcessing / Audio / AudioReader) to read the audio file that you have created in the previous step (secret.wav).

6.3 Verify that the audio file is working, connecting an audio player (Actors / SignalProcessing / Audio / AudioPlayer) and making it sound. You can remove it afterwards.

6.4 Convert the audio signal to an array of 500 positions and find the maximum value by using the actors (Actors / Array / SequenceToArray) and (Actors / Array / ArrayMaximum). As we said before this value will be the Amplitude of the original signal.

$$A = Max(f(t))$$

6.5 Add a bifurcation point and feed the input signal into a differential actor (Actors / Math / Differential), that will compute the differences between consecutive samples of the input signal, i.e. $\Delta f(t)$.

6.6 Convert the output to an array of 500 positions and find the maximum value by using the actors (Actors / Array / SequenceToArray) and (Actors / Array / ArrayMaximum)

6.7 Make a graph (SequencePlotter) of maximum value of differences over time

Question: Include the graph of maximum value of differences over time

6.8 Create a mathematical expression (Actors / Math / Expression) to generate the frequency value from the maximum values we have computed before, Amplitude and the derivative function. Remember that you have to use the following formula.

$$f = \frac{Max(\Delta f(t))}{A \cdot 2\pi \cdot} f_s$$

Question: Include the graph of detected frequencies

Punt 7

Once we have a frequency value we have to decode the symbol

$$\text{index} = \frac{FREQKEY - 100}{100}$$

7.1 Create a mathematical expression (Actors / Math / Expression) to obtain the symbol index. The first valid symbol would be 1, but we need it to be 0 because of the modules that we will connect afterwards. So we will subtract 1 from the previous expression...

$$\text{index} = \frac{FREQKEY - 100}{100} - 1$$

7.2 Add a rounding actor (Actors / Math / Round) to convert the value into an integer

7.3 Add a table of indexed symbols (Actors / Math / LookupTable) containing the strings of each letter of the uppercase alphabet, and connect it to the output of the previous module.

{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'}

7.4 Connect a display (Actors / Sinks / GenericSinks / Display) to the output of the previous module and execute the system.

Question: Include the diagram of your system. Is it working properly?

Step 8

Unfortunately we have not considered synchronization in our clumsy communication system. Although the symbol rate is fixed in our emitter, the receiver has not clock recovery method. So we cannot determine the instants when a symbol starts or ends.

To overcome this problem we will do a very nasty thing, if we detect an incoming symbol, which is different than the previously detected, we will assume that a new valid symbol has arrived. If not, we do not inform of any incoming symbol.

8.1 Remove the duplicate symbols in the list generated in the previous step.

You will need to use the following actors:

(Actors / DomainSpecific / DiscreteEvent / Previous) to keep the previous value

(Actors / String / StringCompare) to detect if the detected symbol is the same as the previously detected.

(Actors / DomainSpecific / DiscreteEvent / EventFilter) to reject the false events coming out from the previous comparison.

(Actors / DomainSpecific / DiscreteEvent / Inhibit) to avoid the propagation of the symbols that have been detected as repeated.

So you have all the building blocks to build a system that removes the repeated symbols.

Question: Include a diagram of your system.

Question: Include the output of your Display.

8.2 Save your Ptolemy model as SecretReceiver.xml

8.2 Try to substitute the AudioReader by an AudioCapture and see if it works...

Lab 3

Objectives

Familiarize with Ptolemy II
Learn configuration possibilities
Work in the Discrete Events (DE) and Synchronous Reactive (SR) domains
Work with sound card I/O and sound files.

Rules

Answer the questions in this same document (in Word format). Only write in response to a **Question** identified as such. Paste screen snapshots when needed.

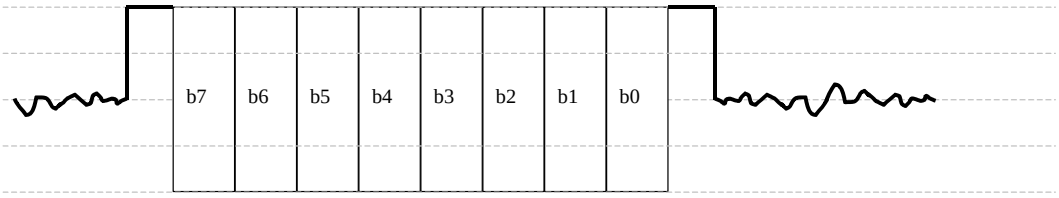
Send the final document by email to david.castells@uab.cat

Introduction

In the last Lab we saw how to use MFSK modulation to transmit signals over the sound card. We could interpret that we were using a modulation of continuous signals system (Frequency Modulation) to encode discrete symbols.

Today we will try to create a digital communication, meaning that we will use only two voltage levels to encode the data to transmit. We will us an approach very similar to the well known RS-232 (Recommended Standard 232). In RS-232 we actually have 33 valid voltage levels: the logic 0 is a voltage between (-12, -5), the logic 1 is a voltage between (+5, +12), and other values are interpreted as signal absence. In our sound card we can not drive a especific voltage level (and also depends of the configured volume level that we set), so we will use values relative to the full scale. If we interpret the audio (A) as a sequence of floating point values between (-1, +1) then we can could similarly assign the range (A > +0.5) to the logic 1, (A < -0.5) to the logic 0, and signal absence to the rest of the range.

Our version of the RS-232 stream will use an start bit, 8 data bits and a stop bit.



It is probable that our computers are not powerful enough to be able to perform the processing of the signals in real time, so we will use sound files as an intermediate step. So we will create a .WAV file instead of generating a sound in the loudspeakers. In the reception side we will read the saved file instead of using the microphone input.

Step 1

1.1 Besides having a large number of predefined actors, Ptolemy II can be extended with new actors developed by you. You only need a Java compiler, some Java programming skills and to carefully read the Ptolemy documentation about how to create new actors.
Since learning Java programming is not an objective of this course (although I encorage you to do it) you can use the actors that I have compiled myself that you can find in the NonBlockingShell.jar (you can also find the source code in the anex).
To be able to use the new actors you will have to start the Vergil from a new command line.

Create a vergil.cmd file in the Ptolemy directory (by default C:\Ptolemy\ptII7.0.1) with the following content:

```
SET PTII=C:\Ptolemy\ptII7.0.1
java                                                                    -cp
%PTII%\ptolemy\ptsupport.jar;%PTII%\lib\diva.jar;%PTII%\ptolemy\vergil\vergil.
jar;%PTII%\ptolemy\domains\domains.jar;%PTII%\ptolemy\domains\ct\ct.jar;%PTII%
\ptolemy\domains\de\de.jar;%PTII%\ptolemy\domains\fsm\fsm.jar;%PTII%\ptolemy\d
omains\psdf\psdf.jar;%PTII%\ptolemy\domains\ddf\ddf.jar;%PTII%\NonBlockingShel
l.jar ptolemy.vergil.VergilApplication
```

1.2 Run the created script and create a new design using the DE model director, which is the one that we will be using from hereafter.

1.3 Create two parameters (Utilities/Parameters/Parameter) called *Baudrate* and *SamplingFreq*. Initialize their values to 1200 and 22050 respectivelly.

1.4 Instantiate one of the actors that I developed, which is included in the NonBlockingShell.jar. As it does not appear in the actor's library that you have in the left part of the application, you will have to instantiate it by using the (Graph / Instantiate Entity) menu and and writing down the name of the Java class "org.cephis.ptolemy.NonBlockingShell".

1.5 Notice that this actor, unlike InteractiveShell, has an input called trigger to induce its execution. This is necessary because this actor does not blocks when activation event occurs as did the InteractiveShell actor.

1.6 Add a Clock (Actors / Sources /Timed Sources / Clock) with a 1s period connected to the trigger input of NonBlockingShell.

1.7 Connect to the NonBlockingShell output the necessary modules to serialize the characters typed in the console. Display them with a display actor. Use the actors we already know (...StringToUnsignedByteArray), (...ArrayToSequence) and (...Display).

Question: Paste the resulting system's diagram.

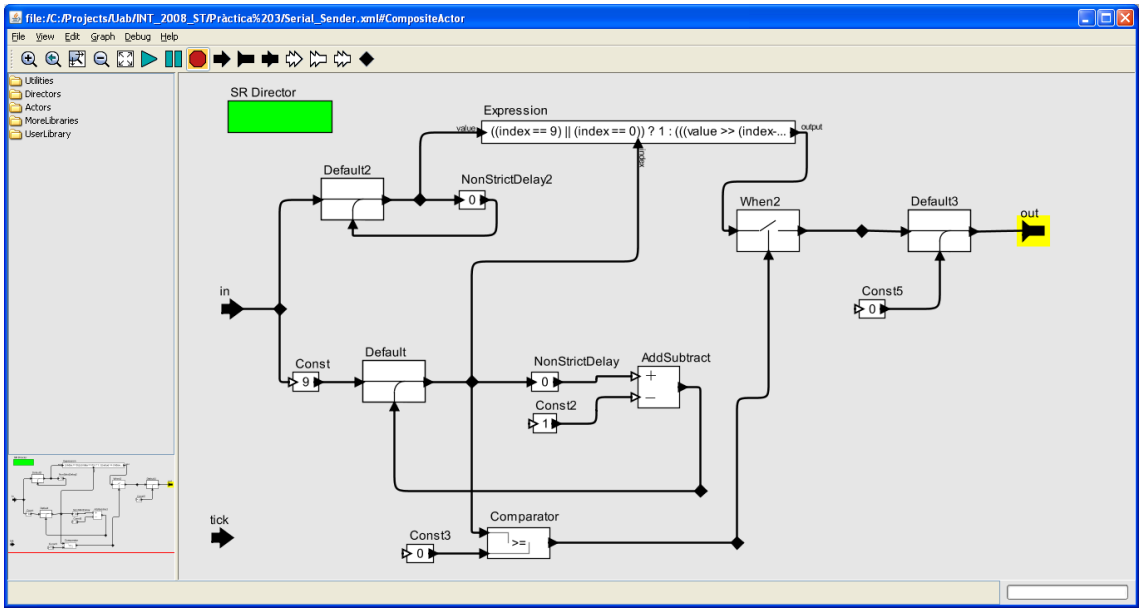
Step 2

Now we have to generate a temporal sequence (similar to RS232) for each character that we want to trasmit. To accomplish this we will use one of the most interesting features of Ptolemy, the capacity to define composite actors i.e. actors containing other actors that can even mix different domains.

2.1 Create a composite actor with the menu (Graph / Create Hierarchy)

2.2 Enter the definition of the new actor by selecting the actor and using the contextual menu (OpenActor).

2.3 Replicate the following diagram as the description of this actor



Notice that the domain of this actor is SR (Synchronous Reactive) and that we use some of the actors addressed to this domain.

The mathematic expression that we will use is:

```
((index == 9) || (index == 0)) ? 1 : (((value >> (index-1)) & 1)>0)? 1:-1
```

Question: Can you summerize how this actor works ?

2.3 Connect the output of the character sequencer to this new actor "in" input, and a clock of frequency *Baudrate* (you can use the name instead of the value in the actor's edit field) to the "tick" input.

2.4 Connect the output of this actor to a (...SequencePlotter)

Question: Execute the system, type "A" and press the Return key in the Shell. Paste the diagram showing how the serial sequence is produced.

Question: Execute the system, type "AB" and press the Return key in the Shell. Include the diagram showing how the serial sequence is produced.

Question: The system is not working well. Why ?

Step 3

We are close to finish out transmission system...

3.1 Interpose an actor (Actors / DomainSpecific / DiscreteEvent / Server) between the character sequencer and the composite actor. Assign it a working frequency of *Baudrate* / 15.

Question: What do you think we can gain doing this ?

3.2 Connect the output of the composite actor to a (Actors / DomainSpecific / DiscreteEvent / Register) where the trigger input is connected to a clock of frequency *SamplingFreq*.

Question: Why do you think we need this module ?

3.3 Finally connect the Register output to an AudioWriter to save the output sequence into a file named serial.wav.

3.4 Save your design in a file called fitxer Serial_Sender.xml

Annex 1: Codi de l'actor NonBlockingShell

```
/*
 * A twig to the original interactive shell code perpetrated
 * by David Castells i Rufas (david.castells@uab.cat)
 */


package org.cephis.ptolemy;

/* An interactive shell that reads and writes strings.

 @Copyright (c) 1998-2008 The Regents of the University of California.
 All rights reserved.

 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the
 above copyright notice and the following two paragraphs appear in all
 copies of this software.

 IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF
 CALIFORNIA HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.

 PT_COPYRIGHT_VERSION 2
 COPYRIGHTENDKEY
 */


import java.awt.Container;
import java.io.IOException;
import java.io.Writer;
import java.util.LinkedList;
import java.util.List;

import javax.swing.SwingUtilities;

import ptolemy.actor.TypedAtomicActor;
import ptolemy.actor.TypedIOPort;
import ptolemy.actor.gui.Configuration;
import ptolemy.actor.gui.Effigy;
import ptolemy.actor.gui.ExpressionShellEffigy;
import ptolemy.actor.gui.ExpressionShellFrame;
import ptolemy.actor.gui.ExpressionShellTableau;
import ptolemy.actor.gui.Placeable;
import ptolemy.actor.gui.TableauFrame;
import ptolemy.actor.gui.WindowPropertiesAttribute;
import ptolemy.actor.parameters.PortParameter;
import ptolemy.data.StringToken;
import ptolemy.data.type.BaseType;
import ptolemy.gui.ShellInterpreter;
import ptolemy.gui.ShellTextArea;
import ptolemy.kernel.CompositeEntity;
import ptolemy.kernel.util.IllegalActionException;
import ptolemy.kernel.util.NameDuplicationException;
import ptolemy.kernel.util.Nameable;
import ptolemy.kernel.util.Workspace;

//////////////////////////////////////////////////////////////////////////
//// InteractiveShell

/**
 <p>This actor creates a command shell on the screen, sending commands
 that are typed by the user to its output port, and reporting strings
 received at its input by displaying them.  Each time it fires, it
 reads the input, displays it, then displays a command prompt
 (which by default is "&gt;&gt;"), and waits for a command to be
```

```
 typed.  The command is terminated by an enter or return character,
 which then results in the command being produced on the output.
 In a typical use of this actor, it will be preceded by a SampleDelay
 actor which will provide an initial welcome message or instructions.
 The output will then be routed to some subsystem for processing,
 and the result will be fed back to the input.
 </p><p>
 Note that because of complexities in Swing, if you resize the display
 window, then, unlike the plotters, the new size will not be persistent.
 That is, if you save the model and then re-open it, the new size is
 forgotten.  The position, however, is persistent.</p>

 @author  Edward A. Lee
 @version $Id: InteractiveShell.java,v 1.30.4.2 2008/03/25 22:31:27 cxh Exp $
 @since Ptolemy II 1.0
 @Pt.ProposedRating Yellow (eal)
 @Pt.AcceptedRating Red (cxh)
 */
public class NonBlockingShell extends TypedAtomicActor implements Placeable,
        ShellInterpreter {
    /** Construct an actor with the given container and name.
     *  @param container The container.
     *  @param name The name of this actor.
     *  @exception IllegalActionException If the actor cannot be contained
     *   by the proposed container.
     *  @exception NameDuplicationException If the container already has an
     *   actor with this name.
     */
    public NonBlockingShell(CompositeEntity container, String name)
            throws IllegalActionException, NameDuplicationException {
        super(container, name);

        trigger = new TypedIOPort(this, "trigger", true, false);
        trigger.setTypeEquals(BaseType.UNKNOWN);

        input = new TypedIOPort(this, "input", true, false);
        input.setTypeEquals(BaseType.STRING);

        output = new TypedIOPort(this, "output", false, true);
        output.setTypeEquals(BaseType.STRING);

        prompt = new PortParameter(this, "prompt", new StringToken(">>"));

        // Make command be a StringParameter (no surrounding double quotes).
        prompt.setTypeEquals(BaseType.STRING);
        prompt.setStringMode(true);

        _windowProperties = new WindowPropertiesAttribute(this,
                "_windowProperties");
        // Note that we have to force this to be persistent because
        // there is no real mechanism for the value of the properties
        // to be updated when the window is moved or resized. By
        // making it persistent, when the model is saved, the
        // attribute will determine the current size and position
        // of the window and save it.
        _windowProperties.setPersistent(true);

        _attachText("_iconDescription", "<svg>\n"
                + "<rect x=\"-20\" y=\"-20\" " + "width=\"40\" height=\"40\" "
                + "style=\"fill:lightGrey\"/>\n" + "<rect x=\"-14\" y=\"-14\" "
                + "width=\"28\" height=\"28\" " + "style=\"fill:white\"/>\n"
                + "<polyline points=\"-10,-10, -5,-5, -10,0\" "
                + "style=\"stroke:black\"/>\n"
                + "<polyline points=\"-7,-10, -2,-5, -7,0\" "
                + "style=\"stroke:black\"/>\n" + "</svg>\n");
    }

    ///////////////////////////////////////////////////////////////////
    ////                     ports and parameters                  ////

    public TypedIOPort trigger;

    /** The input port. */
    public TypedIOPort input;

    /** The output port. */
    public TypedIOPort output;
```

```
/** The prompt.  The initial default is the string >>.  Double
 * quotes are not necessary.  If you would like to have no prompt
 * (aka, the empty string), create a Parameter that has the value
 * "" (for example <code>foo</code>) and then set the value of the
 * prompt parameter to <code>$foo</code>.
 */
public PortParameter prompt;

/** The shell window object. */
public ShellTextArea shell;

////////////////////////////////////////////////////////////////////
////                         public methods                     ////

/** Clone the actor into the specified workspace.
 *  @param workspace The workspace for the new object.
 *  @return A new actor.
 *  @exception CloneNotSupportedException If a derived class has an
 *   attribute that cannot be cloned.
 */
public Object clone(Workspace workspace) throws CloneNotSupportedException {
    NonBlockingShell newObject = (NonBlockingShell) super.clone(workspace);
    newObject.shell = null;
    newObject._container = null;
    newObject._frame = null;
    return newObject;
}

/** Evaluate the specified command.
 *  @param command The command.
 *  @return The return value of the command, or null if there is none.
 *  @exception Exception If something goes wrong processing the command.
 */
public String evaluateCommand(String command) throws Exception {
    // NOTE: This method is typically called in the swing event thread.
    // Be careful to avoid locking up the UI.
    setOutput(command);

    // Return null to indicate that the command evaluation is not
    // complete.  This results in disabling editing on the text
    // widget until returnResult() is called on it, which happens
    // the next time fire() is called.
    return null;
}

/** Read and display the input, then
 *  wait for user input and produce the user data on the output.
 *  If the user input is "quit" or "exit", then set a flag that causes
 *  postfire() to return false.
 *  @exception IllegalActionException If producing the output
 *   causes an exception.
 */
public void fire() throws IllegalActionException {
    super.fire();
    prompt.update();

    shell.mainPrompt = ((StringToken) prompt.getToken()).stringValue();

    if (trigger.hasToken(0))
    {
        // consume the token
        trigger.get(0);
    }

    if ((input.numberOfSources() > 0) && input.hasToken(0))
    {
        String value = ((StringToken) input.get(0)).stringValue();

        // FIXME: shell will never be null because it is dereferenced
        // above.  We leave this code for documentation purposes.
        // If window has been dismissed, there is nothing more to do.
        //if (shell == null) {
        //    return;
        //}

        if (_firstTime)
```

```
            {
                _firstTime = false;
                shell.initialize(value);
            } else {
                shell.returnResult(value);
            }
        }

        shell.setEditable(true);

        // FIXME: What to type if there is no input connected.
        if (_outputValues.size() > 0)
        {
            String userCommand = (String) _outputValues.remove(0);

//            if (userCommand.trim().equalsIgnoreCase("quit")
//                    || userCommand.trim().equalsIgnoreCase("exit")) {
//                _returnFalseInPostfire = true;
//            }



            output.broadcast(new StringToken(userCommand));

        }


    }

    /** Get the output string to be sent. This does not
     *  return until a value is entered on the shell by the user.
     *  @return The output string to be sent.
     *  @see #setOutput(String)
     */
//    public synchronized String getOutput() {
//        while ((_outputValues.size() < 1) && !_stopRequested) {
//            try {
//                // NOTE: Do not call wait on this object directly!
//                // If another thread tries to get write access to the
//                // workspace, it will deadlock!  This method releases
//                // all read accesses on the workspace before doing the
//                // wait.
//                workspace().wait(this);
//            } catch (InterruptedException ex) {
//            }
//        }
//
//        if (_stopRequested)
//        {
//            return ("");
//        }
//        else {
//            return ((String) _outputValues.remove(0));
//        }
//    }

    /** If the shell has not already been created, create it.
     *  Then wait for user input and produce it on the output.
     *  @exception IllegalActionException If the parent class throws it.
     */
    public void initialize() throws IllegalActionException {
        super.initialize();

        if (shell == null)
        {
            // No container has been specified for the shell.
            // Place the shell in its own frame.
            // Need an effigy and a tableau so that menu ops work properly.
            Effigy containerEffigy = Configuration.findEffigy(toplevel());

            if (containerEffigy == null) {
                throw new IllegalActionException(this,
                        "Cannot find effigy for top level: "
                                + toplevel().getFullName());
            }

            try {
```

```
            ExpressionShellEffigy shellEffigy = new ExpressionShellEffigy(
                    containerEffigy, containerEffigy.uniqueName("shell"));

            // The default identifier is "Unnamed", which is no good for
            // two reasons: Wrong title bar label, and it causes a save-as
            // to destroy the original window.
            shellEffigy.identifier.setExpression(getFullName());

            ShellTableau tableau = new ShellTableau(shellEffigy, "tableau");
            _frame = tableau.frame;
            shell = tableau.shell;
            shell.setInterpreter(this);

            // Prevent editing until the first firing.
            shell.setEditable(false);
        } catch (Exception ex) {
            throw new IllegalActionException(this, null, ex,
                    "Error creating effigy and tableau");
        }

        _windowProperties.setProperties(_frame);
        _frame.pack();
    }
    else
    {
        shell.clearJTextArea();
    }

    if (_frame != null) {
        // show() used to override manual placement by calling pack.
        // No more.
        _frame.show();
        _frame.toFront();
    }

    _firstTime = true;
    _returnFalseInPostfire = false;
}

/** Return true if the specified command is complete (ready
 *  to be interpreted).
 *  @param command The command.
 *  @return True.
 */
public boolean isCommandComplete(String command) {
    return true;
}

/** Specify the container into which this shell should be placed.
 *  This method needs to be called before the first call to initialize().
 *  Otherwise, the shell will be placed in its own frame.
 *  The background of the plot is set equal to that of the container
 *  (unless it is null).
 *  @param container The container into which to place the shell, or
 *   null to specify that a new shell should be created.
 */
public void place(Container container) {
    _container = container;

    if (_container == null) {
        // Dissociate with any container.
        // NOTE: _remove() doesn't work here.  Why?
        if (_frame != null) {
            _frame.dispose();
        }

        _frame = null;
        shell = null;
        return;
    }

    shell = new ShellTextArea();
    shell.setInterpreter(this);
    shell.clearJTextArea();
    shell.setEditable(false);

    _container.add(shell);
```

```
    // java.awt.Component.setBackground(color) says that
    // if the color "parameter is null then this component
    // will inherit the  background color of its parent."
    shell.setBackground(null);
}

/** Override the base class to return false if the user has typed
 *  "quit" or "exit".
 *  @return False if the user has typed "quit" or "exit".
 *  @exception IllegalActionException If the superclass throws it.
 */
public boolean postfire() throws IllegalActionException {
    if (_returnFalseInPostfire) {
        return false;
    }

    return super.postfire();
}

/** Override the base class to remove the shell from its graphical
 *  container if the argument is null.
 *  @param container The proposed container.
 *  @exception IllegalActionException If the base class throws it.
 *  @exception NameDuplicationException If the base class throws it.
 */
public void setContainer(CompositeEntity container)
        throws IllegalActionException, NameDuplicationException {
    Nameable previousContainer = getContainer();
    super.setContainer(container);

    if ((container != previousContainer) && (previousContainer != null)) {
        _remove();
    }
}

/** Specify an output string to be sent. This method
 *  appends the specified string to a queue. Strings
 *  are retrieved from the queue by getOutput().
 *  @param value An output string to be sent.
 *  @see #getOutput()
 */
public synchronized void setOutput(String value) {
    _outputValues.add(value);
    notifyAll();
}

/** Override the base class to call notifyAll() to get out of
 *  any waiting.
 */
public void stop() {
    super.stop();

    synchronized (this) {
        notifyAll();
    }
}

/** Override the base class to make the shell uneditable.
 *  @exception IllegalActionException If the parent class throws it.
 */
public void wrapup() throws IllegalActionException {
    super.wrapup();

    if (_returnFalseInPostfire && (_frame != null)) {
        _frame.dispose();
        _frame = null;
        shell = null;
    } else if (shell != null) {
        shell.setEditable(false);
    }
}

////////////////////////////////////////////////////////////////
////                         protected methods                  ////

/** Write a MoML description of the contents of this object. This
```

```
     *  overrides the base class to make sure that the current frame
     *  properties, if there is a frame, are recorded.
     *  @param output The output stream to write to.
     *  @param depth The depth in the hierarchy, to determine indenting.
     *  @exception IOException If an I/O error occurs.
     */
    protected void _exportMoMLContents(Writer output, int depth)
            throws IOException {
        // Make sure that the current position of the frame, if any,
        // is up to date.
        if (_frame != null) {
            _windowProperties.recordProperties(_frame);
        }

        super._exportMoMLContents(output, depth);
    }

    ////////////////////////////////////////////////////////////////////
    ////                         private members                    ////

    /** Container into which this plot should be placed. */
    private Container _container;

    /** Indicator of the first time through. */
    private boolean _firstTime = true;

    /** Frame into which plot is placed, if any. */
    private TableauFrame _frame;

    /** The list of strings to send to the output. */
    private List _outputValues = new LinkedList();

    /** Flag indicating that "exit" or "quit" has been entered. */
    private boolean _returnFalseInPostfire = false;

    // A specification for the window properties of the frame.
    private WindowPropertiesAttribute _windowProperties;

    ////////////////////////////////////////////////////////////////////
    ////                         private methods                    ////

    /** Remove the shell from the current container, if there is one.
     */
    private void _remove() {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                if (shell != null) {
                    if (_container != null) {
                        _container.remove(shell);
                        _container.invalidate();
                        _container.repaint();
                    } else if (_frame != null) {
                        _frame.dispose();
                    }
                }
            }
        });
    }

    ////////////////////////////////////////////////////////////////////
    ////                         inner classes                      ////

    /** Version of ExpressionShellTableau that records the size of
     *  the display when it is closed.
     */
    public class ShellTableau extends ExpressionShellTableau {
        /** Construct a new tableau for the model represented by the
         *  given effigy.
         *  @param container The container.
         *  @param name The name.
         *  @exception IllegalActionException If the container does not accept
         *   this entity (this should not occur).
         *  @exception NameDuplicationException If the name coincides with an
         *   attribute already in the container.
         */
        public ShellTableau(ExpressionShellEffigy container, String name)
                throws IllegalActionException, NameDuplicationException {
```

```
            super(container, name);
            frame = new ShellFrame(this);
            setFrame(frame);
            frame.setTableau(this);
        }
    }

    /** The frame that is created by an instance of ShellTableau.
     */
    public class ShellFrame extends ExpressionShellFrame {
        /** Construct a frame to display the ExpressionShell window.
         *  Override the base class to handle window closing.
         *  After constructing this, it is necessary
         *  to call setVisible(true) to make the frame appear.
         *  This is typically accomplished by calling show() on
         *  enclosing tableau.
         *  @param tableau The tableau responsible for this frame.
         *  @exception IllegalActionException If the model rejects the
         *   configuration attribute.
         *  @exception NameDuplicationException If a name collision occurs.
         */
        public ShellFrame(ExpressionShellTableau tableau)
                throws IllegalActionException, NameDuplicationException {
            super(tableau);
        }

        /** Overrides the base class to record
         *  the size and location of the frame.
         *  @return False if the user cancels on a save query.
         */
        protected boolean _close() {
            if (_frame != null) {
                _windowProperties.setProperties(_frame);
            }

            // Return value can be ignored since there is no issue of saving.
            super._close();
            place(null);
            return true;
        }
    }
}
```

Lab 4

Objectives

Familiarize with Ptolemy II
Learn configuration possibilities
Learn how to handle mathematic expressions
Learn how to handle arrays
Learn about control flow methods

Work in the Discrete Events (DE) domain
Work with sound card I/O and sound files.

Rules

Answer the questions in this same document (in Word format). Only write in response to a **Question** identified as such. Paste screen snapshots when needed.

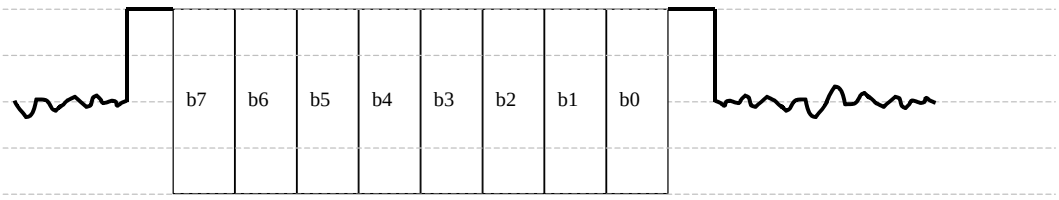
Send the final document by email to david.castells@uab.cat

Introduction

In the last Lab we saw how to use MFSK modulation to transmit signals over the sound card. We could interpret that we were using a modulation of continuous signals system (Frequency Modulation) to encode discrete symbols.

Today we will try to create a digital communication, meaning that we will use only two voltage levels to encode the data to transmit. We will us an approach very similar to the well known RS-232 (Recommended Standard 232). In RS-232 we actually have 3 valid voltage levels: the logic 0 is a voltage between (-12, -5), the logic 1 is a voltage between (+5, +12), and other values are interpreted as signal absence. In our sound card we can not drive a specific voltage level (and also depends of the configured volume level that we set), so we will use values relative to the full scale. If we interpret the audio (A) as a sequence of floating point values between (-1, +1) then we can could similarly assign the range (A > +0.5) to the logic 1, (A < -0.5) to the logic 0, and signal absence to the rest of the range.

Our version of the RS-232 stream will use an start bit, 8 data bits and a stop bit.



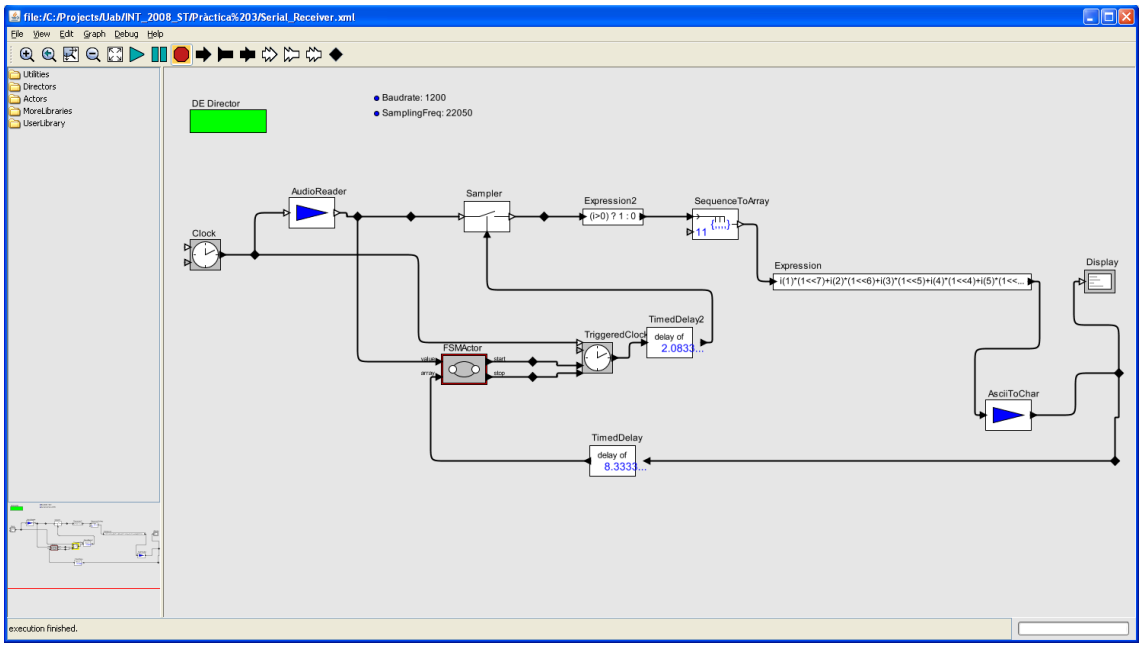
It is probable that our computers are not powerful enough to be able to perform the processing of the signals in real time, so we will use sound files as an intermediate step.

So we will create a .WAV file instead of generating a sound in the loudspeakers. In the reception side we will read the saved file instead of using the microphone input.

Step 1

Let's start building our receiver

1.1 Replicate the following system...

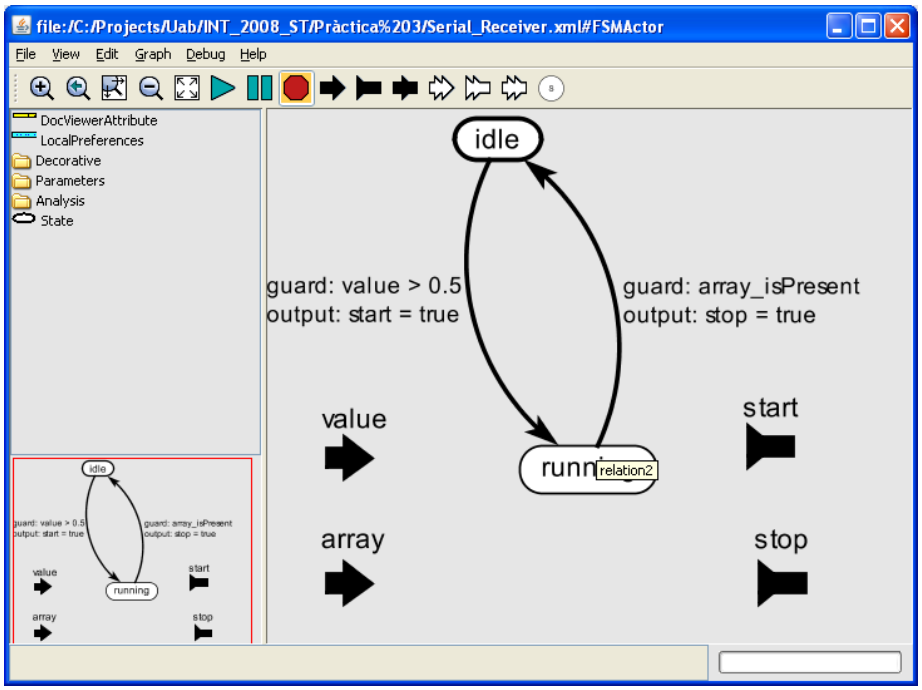


Take into account the following considerations

1) The AudioReader has to read data from the file that we previously saved
2) The Clock frequency is *SamplingFreq*
3) The TriggeredClock frequency is *Baudrate*
4) The complex mathematic expression is

```
i(1)*(1<<7)+i(2)*(1<<6)+i(3)*(1<<5)+i(4)*(1<<4)+i(5)*(1<<3)+i(6)*(1
<<2)+i(7)*(1<<1)+i(8)
```

5) The TimedDelay value is 0.1/Baudrate
6) The TimedDelay2 value is 0.25/Baudrate
7) AsciiToChar is an actor that I developed and is stored in the org.cephis.ptolemy.AsciiToChar Java class.
8) You have to create the FSMActor by using (MoreLibraries / Automata / FSMActor) and it has to contain...



Question: Does it work ?

Question: Can you summarize the operational principle ?

Question: Try to substitute the use of intermediate files by a direct sound-card to sound-card connection via a provided cable. Does it work? Why ?

Question: Do you think that it could work if we used a different Baudrate? Try if it works in different values of Baudrate and write down which ones are working.

Annex 1: AsciiToChar actor source code

```
/**
 * This code was perpetrated by David Castells i Rufas (david.castells@uab.cat)
 */

package org.cephis.ptolemy;

import ptolemy.actor.lib.Transformer;
import ptolemy.data.IntToken;
import ptolemy.data.StringToken;
import ptolemy.data.type.BaseType;
import ptolemy.kernel.CompositeEntity;
import ptolemy.kernel.util.IllegalActionException;
import ptolemy.kernel.util.NameDuplicationException;

//////////////////////////////////////////////////////////////////////////
//// AsciiToChar

/**
 * Output a string containing the character corresponing to the ascii code of the input
 */
public class AsciiToChar extends Transformer {

    public AsciiToChar(CompositeEntity container, String name)
            throws NameDuplicationException, IllegalActionException {
        super(container, name);




        input.setTypeEquals(BaseType.INT);
        output.setTypeEquals(BaseType.STRING);
    }


    public void fire() throws IllegalActionException {
        super.fire();


        if (input.hasToken(0))
        {
            IntToken inputToken = (IntToken) input.get(0);
            int value = inputToken.intValue();

            byte[] array = new byte[]{(byte)value};

            output.send(0, new StringToken(new String(array)));
        }
    }
}
```

Pràctica 5

Objectives

Familiarize with Wireshark
Understand IP routing
Capture network packets

Rules

Answer the questions in this same document (in Word format). Only write in response to a **Question** identified as such. Paste screen snapshots when needed.

Send the final document by email to david.castells@uab.cat

Introduction

Today we leave Ptolemy. We will work with the network that are connected to our computers.
In order to do that we will use some tools from the Windows console and the Wireshark network sniffer.

Step 1

We will use the *ping* and *tracert* commands to see the response time of different internet servers.

1.1 Try to do a *ping* to the www.telefonica.es server

1.2 Try to do a *ping* to the www.nasa.gov server

1.3 Try to do a *ping* to the http://www.tourist.fo server

Question Which one takes more time to respond ?

Question Use the *tracert* command to determine which part of the route is common to the tested servers. Paste it below.

Question Use the http://visualroute.visualware.com/ tool to determine the geographic location of the servers you have contacted.

Step 2

Out computer network protocol stack needs to maintain a binding table between the physical layer addresses (MAC) and the network layer addresses (IP) of the computers that we have at reach. We can view this table with the *arp* command.

2.1 Determine your IP address by running the *ipconfig* command

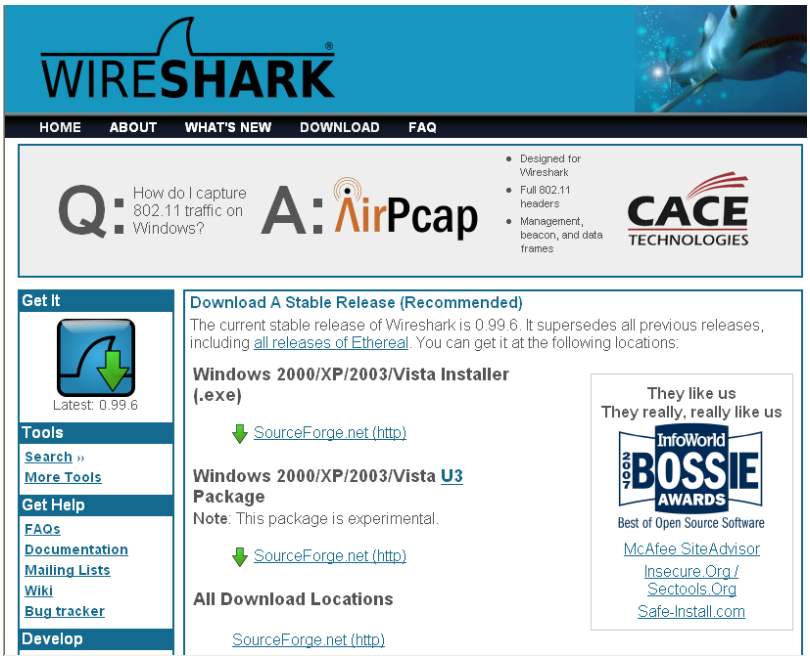
Question Execute the command "arp –a" and write the result

2.2 Ask a class fellow that has a computer which is not in the previous list to run a *ping* command to your IP address.

Question Execute again the "arp –a" command and write down the result

Question Explain what has changed and why.

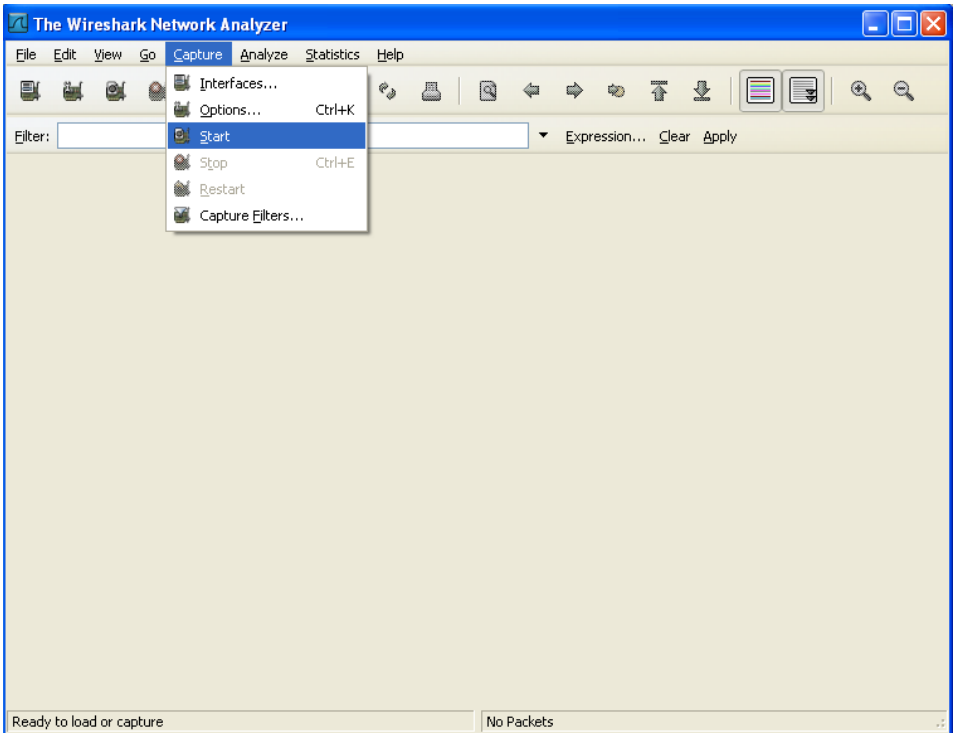
Step 3

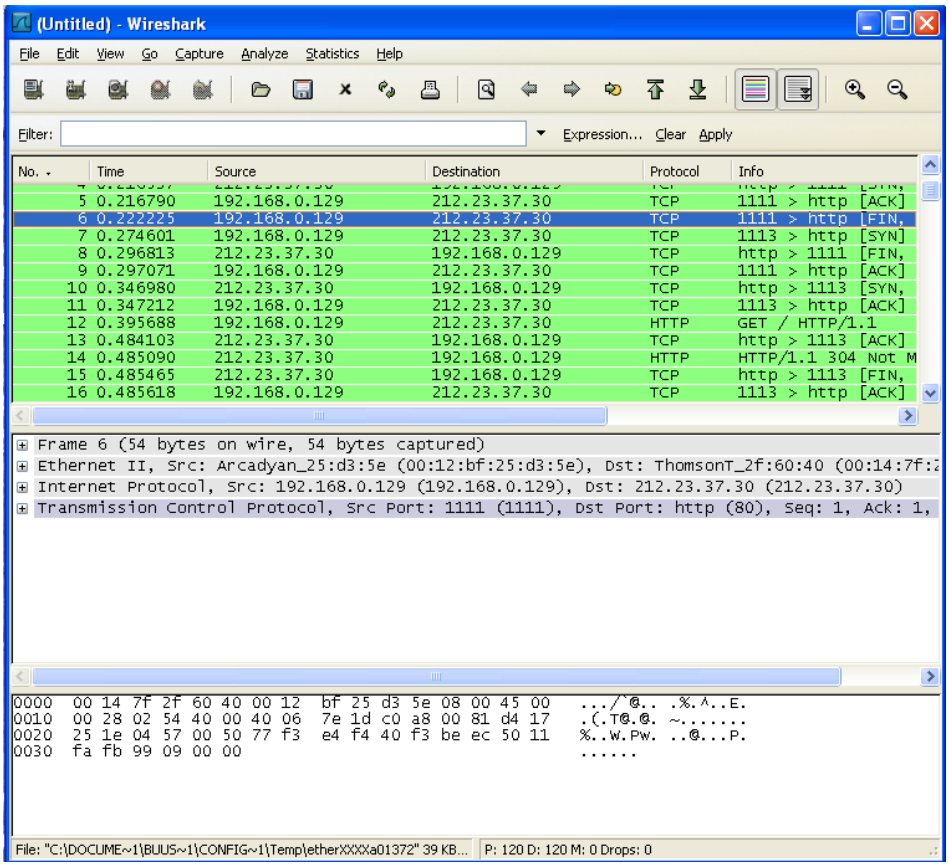


From now on we will analyze some protocols at various networking layers (data link, network, transport and application). We will analyze the network packets by using the *Wireshark* application. This is known as *sniffing* the network.

To sniff the network with Wireshark we have to use the *Capture* > *Interfaces* menu or directly *Capture* > *Start*. To stop the capture use the *Capture* > *Stop* menu.

To analyze the information of a packet simply clock it in the packet list. In the middle window you can see the structure of the frame, of the MAC (Ethernet) layer, IP, TCP/UDP and above protocols as we need it.





3.1 Capture a **ARP** (*Ethernet)* packet. Use a screen capture to show the contents of the packet header. Answer the following questions by writing the observed hexadecimal value, the numeric value if convenient, and a short description.

Question Which type of packet is it ? a request or a response?

Question Write down the source and destination addresses of the packet. Which kind of addresses are they ?

Question Who and why has this packet been sent ?

3.2 Capture a MAC (*Ethernet*) packet. Use a screen capture to show the contents of the packet header.

Question Write down the source and destination MAC addresses in hexadecimal format.

Question Who are this addresses from?

Question Which MTU is used?

3.3 Capture an **IP** packet. Use a screen capture to show the contents of the packet IP header. Answer the following questions by writing the observed hexadecimal value, the numeric value if convenient, and a short description.

Question Which version of the IP protocol is used ?

Question Which type of service ?

Question What is the header length ?

Question Is this part of a fragmented packet? Why?

Question Write down the source and destination IP addresses? Which kind of addresses are they ?

Question Who are the addresses from?

Question Explain which sequence of actions have been taken in the different routers that this packet has traversed (if any) to correctly route the packet from the source computer to the destination computer.