Università degli Studi di Pisa

Laurea Magistrale in Informatica

Laurea Magistrale in Informatica e Networking

Advanced Programming

Final Term Paper

Start Date: 19/08/2011
Submission deadline: 19/08/2011 (send a single PDF file to attardi@di.unipi.it)

Rules:

The paper must be produced personally by the student, signed implicitly via his mail address.

You are allowed to discuss with others the general lines of the problems, provided that each student eventually formulates his own solution. Each student is expected to understand and to be able to explain his solution.

You are allowed to consult documentation from any source, provided that references are mentioned.

It is not considered acceptable:

- to develop code or pseudo-code with others
- to use code written by others
- to let others use someone's code
- to show or to examine the work of other students.

Violation of these rules will result in the cancellation of the exam and a report to the Presidente del Consiglio di Corso di Studio.

For the programming exercises you can choose a programming language among C++, C# and Java.

The paper must:

1. be in a single PDF file, formatted readably (font size ≥ 9pt with suitable margins, single column), of **no more than 10 numbered pages**, including code: for each extra page one point will be subtracted from the score.
2. include **the student name**
3. **provide the solution and the code for each exercise separately**, referring to the code of other exercise if necessary.
4. cite references to literature or Web pages from where information was taken.

Introduction

We will develop a generator of applications, described by means of State Charts, expressed in the markup language SCXML (http://www.w3.org/TR/scxml/).

The following example describes the StateChart for the game of TicTacToe.

```
<scxml version="1.0" initialstate="start" datamodel="ecmascript">
  <datamodel>
     <data id="board" expr="[[0, 0, 0], [0, 0, 0], [0, 0, 0]]" />
  </datamodel>
    <state id="start">
       <transition event="choose" target="playerO"/>
    </state>
    <state id="playerO">
       <transition event="choose" cond="isWin(_data.board)"
                   target="win"/>
       <transition event="choose" target="playerX"/>
    </state>
    <state id="playerX">
```

```
      <transition event="choose" cond="isWin(board)" target="win"/>
      <transition event="choose" target="playerO"/>
    </state>
    <state id="win">
      <transition event="play" target="start"/>
      <transition event="quit" target="end"/>
    </state>
    <state id="lose">
      <transition event="play" target="start"/>
      <transition event="quit" target="end"/>
    </state>
    ...
  </scxml>
```

The `datamodel` element defines the Model on which the application operates. The `state` elements correspond to the states of a finite state machine and the transitions describe the logic of the application. A `transition` is triggered by events that match the `event` attribute, i.e. whose name starts with the attribute value. In the example, events for choosing cell <i, j> would correspond to event with name `chooseij`. If a `cond` attribute is present, the transition is performed only if the evaluation of the condition returns true.

If a `target` attribute is specified in a transition, the machine enters that state; otherwise the actions associated to the transition are performed while remaining in the same state.

The code expressions occurring in the attribute values are written in the programming language specified in attribute `datamodel` of element `scxml`.

Exercise 1

Provide a suitable set of classes to represent an Object Model for SCXML State Charts.

Exercise 2

Implement a parser (using if needed a streaming parsing library for XML, but not SAX nor document readers) to analyze SCXML documents and to produce their representation in the above SCXML Model.

Exercise 3

Implement a suitable set of classes to provide the behavior for State Charts, including general or abstract classes such as `StateMachine`, `State`, `Event`.

Exercise 4

Implement a code generator that transforms an SCXML model into code for a class that specializes `StateMachine`, and provides a method `triggerEvent(String event)` to perform a transition in the machine. This method should be specific for the given machine and should distinguish according to the current state which transition should be triggered.

Instance variables `_data`, `_event` are bound to the datamodel and the current event, respectively.

Esercise 5

Extend the language to allow specifying actions to be performed corresponding to transitions, for example:

```
  <state id="playerO">
    <transition event="choose" cond="isWin(_data.board)"
                target="win"/>
    <transition event="choose" target="playerX"/>
    <onentry>
      <assign location="_data.board[
                        _event.name.charAt(_event.name.length-2)-'0',
                        _event.name.charAt(_event.name.length-1)-'0']"
              expr="1" />
      <raise event="SHOW.BOARD" />
    </onentry>
  </state>
  <transition event="SHOW.BOARD">
    <script>display(_data.board);</script>
  </transition>
```

The `onentry` element specifies actions to be performed when a state is entered. Actions include `raise`, `assign`, `script` and may depend on conditions.

Extend correspondingly also the code generator.

Exercise 6

Complete and implement the application of the Introduction, and build an HTML page that uses the generated code to play the game and to display the board after each move.

Provide the code produced by the generator in Exercise 5.

Exercise 7

Describe a possible technique for implementing exceptions, in particular the mechanism that allows unwinding the stack, including performing any necessary cleanup before leaving a frame and returning to the frame where the exception is handled.

Exercise 8 (for students of the Laurea Specialistica)

Extend the game in order to handle situations of stale.