Università degli Studi di Pisa

Laurea Magistrale in Informatica

Laurea Magistrale in Informatica e Networking

Advanced Programming

Final Term Paper

Start Date: 23/12/2009
Submission deadline: 5/2/2010 (send a single PDF file to attardi@di.unipi.it)

Rules:

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

You are allowed to discuss with others the general lines of the problems, provided that eventually each student 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 test 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), 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

You will build a Web application generator, which adopts the Model-View-Controller pattern for separating presentation and logic. The User Interface is expressed with dynamic HTML and the application logic is described by means of State Charts, expressed using the markup language SCXML (http://www.w3.org/TR/scxml/).
The following example describes the logic of a numeric calculator.

```
<scxml version="1.0" initialstate="start" name="calc">
   <datamodel>
      <data id="expr" expr="0" />
      <data id="res" expr="0" />
   </datamodel>
   <state id="start">
      <transition event="OPER" target="opEntered" />
      <transition event="DIGIT" target="operand" />
   </state>
   <state id="operand">
      <transition event="OPER" target="opEntered" />
      <transition event="DIGIT" />
   </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. 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 UI for the calculator can be expressed in HTML like this:

```
<html>
<script src="calc.js" />
<script>
   function display(txt) {
      document.getElementById("result").innerHTML = txt;
   }
</sctipt>
<body>
  <table>
    <tr>
      <td colspan="4"><div id="result"></div></td>
    </tr>
    <tr>
      <td><a href="#" onclick="calc.raise('DIGIT.7')">7</a></td>
      <td><a href="#" onclick="calc.raise('DIGIT.8')">8</a></td>
      <td><a href="#" onclick="calc.raise('DIGIT.9')">9</a></td>
      <td><a href="#" onclick="calc.raise('OPER.DIV')">/</a></td>
    </tr>
    …
```

Esercise 1

Define a suitable class framework to represent SCXML expressions (SCXML framework). To reduce code verbosity, you are allowed to use public attributes in classes.

Esercise 2

Write an LL(1) grammar for the SCXML fragment of the Introduction and develop a *recursive descent* parser that produces an internal representation for SCXML using the SCXML framework. You are allowed to use an XML reader, provided that it does not read the whole structure into memory.

Esercise 3

Develop a set of Javascript classes, including `StateChart`, `State`, `Event` to implement the generic behavior of a state machine. The `StateChart` exposes a method `raise(eventName)` to raise an event with the given name.

Esercise 4

Implement a code generator that, given an SCXML object, generates Javascript code implementing its behavior. The translator should use polymorphic methods of the SCXML framework. Provide the generated code for the fragment in the Introduction.

Esercise 5

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

```
<scxml version="1.0" initialstate="start" name="calc">
   <datamodel>
      <data id="expr" expr="0" />
      <data id="res" expr="0" />
   </datamodel>
   <state id="start">
```

```
            <transition event="OPER" target="opEntered" />
            <transition event="DIGIT" target="operand" />
         <state id="operand">
            <transition event="OPER" target="opEntered" />
            <transition event="DIGIT">
               <assign dataid="expr"
               expr="_data.expr+_event.name.charAt(_event.name.length-1)" />
               <raise event="DISPLAY.UPDATE" />
            </transition>
            <onentry>
               <assign dataid="expr"
                           expr="_event.name.charAt(_event.name.length-1)" />
               <raise event="DISPLAY.UPDATE" />
            </onentry>
         </state>
         <state id="opEntered">
            <transition event="OPER.MINUS" target="complement" />
            <transition event="DIGIT" target="operand" />
         <onentry>
            <raise event="CALC" />
            <if cond="_event.name == 'OPER.PLUS'">
               <assign dataid="expr" expr="_data.expr+'+'" />
            <elseif cond="_event.name == 'OPER.MINUS'" />
            …
            </if>
         </onentry>
         </state>
         <transition event="CALC"> … </transition>
         <transition event="DISPLAY.UPDATE">
            <script>display(_data.res);</script>
         </transition>
         …
      </scxml>
```

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

Variable `_event` is bound to the current event. Variable `_data` is bound to the machine data model structure. Variable `calc` (the value of attribute name in the `scxml` element) is bound to the state machine itself.

Extend the SCXML framework, the grammar, the parser and the code generator to handle the extended language.

Exercise 6

Complete the SCXML for the calculator, provide the generated code and test the whole application in a browser.

Exercise 7

Discuss the technique of generational garbage collection (use your own words, do not paste text from the Web). What information should a compiler for a language provide in the runtime, in order for a tracing collector to work?

Exercise 8 (for students of the Laurea Specialistica)

Write a function that, given an SCXML object produced by Exercise 2, returns an equivalent SCXML with a minimal number of states.