Advanced Programming

Final Term Paper

Start Date: 22/08/2015
Submission deadline: 11/09/2015 (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 consult or setup an online forum, to request help of consultants in producing the paper**
- 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 ≥ 10 pt** 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 exercises if necessary.
4. cite references to literature or Web pages from where information was taken.

Introduction

In this project, you will develop an event based framework, including an Event Specification Language (ESL). Here is an example of a Dining Philosopher Problem expressed in ESL:

```
<system>
  <agent id="1" name="Philo1">
    <state name="thinking">
       <event type="timeout">
          <next state="hungry"/>
       </event>
    </state>
    <state name="hungry">
       <trigger target="table" type="eat" message="this.id”>
       <event type="eat">
          <trigger type="arm" target="timer"
                   message="random(), this.id">
          <next state="eating"/>
```

```
        </event>
      </state>
      …
    </agent>
  </for>
  <agent id="table">
     <code>..declare code  for methods allowed(), delay(), etc.</code>
     <state name="ready">
       <event type="eat" message="%sender%">
           <if test="allowed(%sender%)">
              <then>
                 <trigger type="eat" target="%sender%">
              </then>
              <else>
                 <code>delay(%sender%)</code>
              </else>
           </if>
        </event>
       <event type="done" message="%sender%">
           <code>done(%sender%)>/code>
           <while test="can_eat()">
                 <trigger type="eat" target="next()">
           </while>
        </event>
     </state>
   </agent>
</system>
```

The behavior of an agent is described by a state machine: within each state, event may cause some action and possibly an event transition, marked by the <next> element. Events specify a type and a message: within a message, the %-notation, binds the corresponding variable to the message parameter. The trigger element produces an event of the given type that is sent to the given target with the given parameters.

A `code` element represents executable code that should be inserted in the generated code. In case the code contains special characters, these should be denoted as HTML entities. The code may contain %-notations: an expression `%x%` will be replaced by the value of variable `x` at the time of expansion by the code generator.

A system agent called `timer` can be used to trigger a timeout event after a given duration to the given agent.

The ESL is used to generate code for the system, by means of a code generator.

Exercise 1

Design a set of classes to represent the syntactic constructs of ESL (e.g. System, Agent, State, etc.) that are expressed in XML notation.

Exercise 2

Implement a recursive descent parser for ESL without using external libraries or parser generators. Split the parser into a lexical analyzer, recognizing as tokens just tags and text, and a syntax analyzer, as presented in the slides of the course.

Exercise 3

Design the Event Framework, consisting in a set of abstract classes to be used to build an event system and that provide the runtime support for event scheduling, triggering, state transitions and concurrent execution. Each agent should run in its own thread. Explain the role of each class, for example, by a state transition diagram.

Exercise 4

Write a code generator that takes as input a system description consisting in the classes defined in Exercise 1, and generates actual executable code for running the system. Ensure to use polymorphism in the generator as uch as possible, in order to make its code modular and reusable.

Exercise 5

Extend the ESL introducing a `for` construct, allowing defining for example multiple agents like thi:

```
<for var="n" to="5">
  <agent id=%n% name="Philo%n%"> … </agent>
</for>
```

Extend the parser and the code generator, I order to handle the new construct.
Show a full specification for the Dining Philosopher Problem expressed in this notation and the code produced for it by the code generator (for simplicity show the generated code for a single agent).

Exercise 6

Describe the technique of tracing garbage collection. Explain which information the garbage collector needs to know about the runtime data structures and the program data structures in order to perform its task. Explain how the collector can obtain such information. Explain why it is difficult to design a collector that could work on code produced by compilers for different programming languages. Discuss any relation or similarity between the information required by a garbage collection and that provided by reflection facilities.