Typestate and Session Types for Java

Dimitris Kouzapas

Department of Computing Science, University of Glasgow

January 13, 2014

Objective

- University of Glasgow Objective:
  - Extend the Java compiler to support typestate for Java objects.
- Express (multiparty) session types as typestates for objects.
- Published work:
  S. Gay, V. Vasconcelos, A. Ravara, N. Gesbert, A. Caldeira, **Modular Session Types for Distributed Object-Oriented Programming**, POPL '10, 2010.
- A first implementation was called **Bica**:
  - University of Lisbon.
  - Used the Java annotation support to describe typestate for Java objects.
  - Never finished.

Java Typestate Project

- Use a compiler tool.
- Develop a language for the description of a typestate.
- Implement typestate for basic features of the Java language.
- Test different implementation semantics.
- Research/implement new features and issues that may arise.

Jast Add Compiler Tool

- Java based
- Describe the nodes for an abstract syntax tree as java classes.
- Parsing creates the AST.
- Aspect-oriented programming to traverse/process/transform the AST.

Development to the date

- Use the Java 1.4 compiler developed by the Jast Add team.
- Suppress features that are not supported by the theory so far (e.g. inheritance, multithreading).
- Develop a language to describe typestate and extend the Java syntax.
- Finite state graph to abstract the typestate type.
- Analyse and extract the typestate of local variables (i.e variables declared in methods) as a graph.
- Simulation (bisimulation) algorithm to check if a local variable usage graph conforms to the typestate definition of the local variable.

Incomplete Features

- Analyse and extract typestate for class fields.
- Check that a field usage graph conforms to the typestate definition of the field.
- Extend the typestate language to support typestate for method arguments - Enable for the return of typestate objects.
- Develop a layer on top of Java sockets (and Java IO) to support multiparty session types communication.

Hello World Example

```
1.  typestate HelloTS {
2.      main {
3.          void HelloWorld(String); either{Alice, Bob}
4.      }
5.      Alice {
6.          void HelloAlice();
7.      }
8.      Bob {
9.          void HelloBob(int);
10.     }
11. }
```

Hello World Example

```
1.  class Hello typestates HelloTS {
2.      public void HelloWorld(String s) {
3.          System.out.println("Hello World : " + s);
4.      }
5.      public void HelloAlice() {
6.          System.out.println("Hello Alice");
7.      }
8.      public void HelloBob(int i) {
9.          System.out.println("Hello Bob : " + i);
10.     }
11.     public static void main(String[] args) {
12.         Hello h = new Hello();
13.         h.HelloWorld("Hi");
14.         if(cond)
15.             h.HelloBob(5);
16.         else
17.             h.HelloAlice();
11. }
```

Automated code generation using Scribble

```
1.  protocol Greetings(role Alice, role Bob) {
2.      say(Hello) from Alice to Bob;
3.      say(Hello) from Bob to Alice;
4.  }
```

Automated code generation using Scribble

```
1.  protocol Greetings(role Alice, role Bob) {
2.      say(Hello) from Alice to Bob;
3.      say(Hello) from Bob to Alice;
4.  }
```

```
1.  typestate BobSession{
2.      main {
3.          String receiveFromAlice(); void sendToAlice(String);
4.      }
5.  }
```

```
1.  typestate AliceSession{
2.      main {
3.          void sendToBob(String); String receiveFromBob();
4.      }
5.  }
```

Automated Code Generation using Scribble

```
1.  class Bob typestates BobSession {
2.      MPSocket s = initSocket();
3.      public String receiveFromAlice(){
4.          return (String) s.receive();
5.      }
6.      public void sendToAlice(String s) {
7.          s.send(s);
8.      }
```

Automated Code Generation using Scribble

```
1.  class Bob typestates BobSession {
2.      MPSocket s = initSocket();
3.      public String receiveFromAlice(){
4.          return (String) s.receive();
5.      }
6.      public void sendToAlice(String s) {
7.          s.send(s);
8.      }
9.      public static void main(String[] args) {
10.         Bob bob = new Bob();
11.         String s = bob.receiveFromAlice();
12.         bob.sendToAlice("Hello Alice");
13.     }
14. }
```

Future Work

- Gay et. al. work develops an inductive type system for type checking whereas the implementation uses a co-inductive simulation algorithm for type checking. Need to develop the theory.
- Typestate inference.
- Typestate and Inheritance.
- Typestate and Polymorphism (also session types and polymorphism).
- Support for runtime typestate check.