SSE2 Project Report

BoCoLa

Course:	SSE2 Spring 2010
Teacher:	Bent Brunn Kristensen
Faculty:	Faculty of Engineering at SDU
Due date:	Spring 2010

Group 1

Martin Moghadam
Kalle Grafström
Audrius Palisaitis
Robertas Jacauskas

Table of contents

1 - Introduction 3

2 – Language Definition 3

3 – Language Translation 4

4 – Runtime System Modification 6

5 – Example Experimentation 8

6 – Conclusion 9

7 – Appendix 10

1 - Introduction

In the course SSE2 we design, implement and experiment with a collaboration abstraction mechanism, BoCoLa, as a supplement to Java.

2 – Language Definition

The abstractions of collaboration the project team defined are **what** event operations are performed. The information is generalized, with that the team skip or reduce details of **how** to actually perform these operations. The collaboration is defined as such a skeleton at abstraction level, shown below:

```
collaboration collaboration-identifier
between bot-list
[
operation list
]
```

Figure 2.1 collaboration definition at an abstraction level

The detailed definition of **how** to define for example bot-list and operation-list is skipped, but from an abstraction level it is known that it should be a part of the collaboration definition. That is why the collaboration should be described as an object (**what** is performed) rather than a class (**how** it is performed).

Tokens are used, tokenized keywords are identifiers, methods (operators), it is reserved symbols, and it could consist of single character or several of them. The parser uses a set of tokens; tokens are used for language translation (created AST, etc.)

```
collaboration c1
between 0, 1, 2
[
work by 1;
move by 2;
sync(move by 1, work by 2);
move by 0;
]
```

Figure2.2 Example of BoCola code

The BoCola language was extended, the keyword **sync** was added in order to perform synchronization. The semantics of operation **sync:**

sync(method by n, method by n);

It should consist of two methods (method could be “work” or “move”), and these methods could be executed by bot(s) n.

The Contextual constrains are; methods could be the same or different as well as bot number n, bot number n should be described in bot list.

3 – Language Translation

The project group created some tokens according to the BoCoLa grammar with an addition of an extra token that wasn't in the grammar at the beginning but were added as an addition to the language at the second iteration of the compiler construction. The sync token was the one added that were to be used to synchronize operations between two bots. The simple lexical analysis was conducted using mainly the Token and Scanner classes. The context free analysis was conducted by the Parser and AST classes. The simple translation was conducted by the Encoder and Visitor class with some additions to the AST classes. In figure 3.2 it can be seen how an abstract syntax tree is constructed from the Experiment 1 BoCoLa source code in fig. 3.1.

```
collaboration c1
between 0, 1, 2
[
sync(move by 1, work by 2);
move by 0;
]
collaboration c2
between 3,4,5
[
move by 5;
work by 3;
move by 4;
]
```

Figure 3.1 BoCoLa Eperiment 1 source code

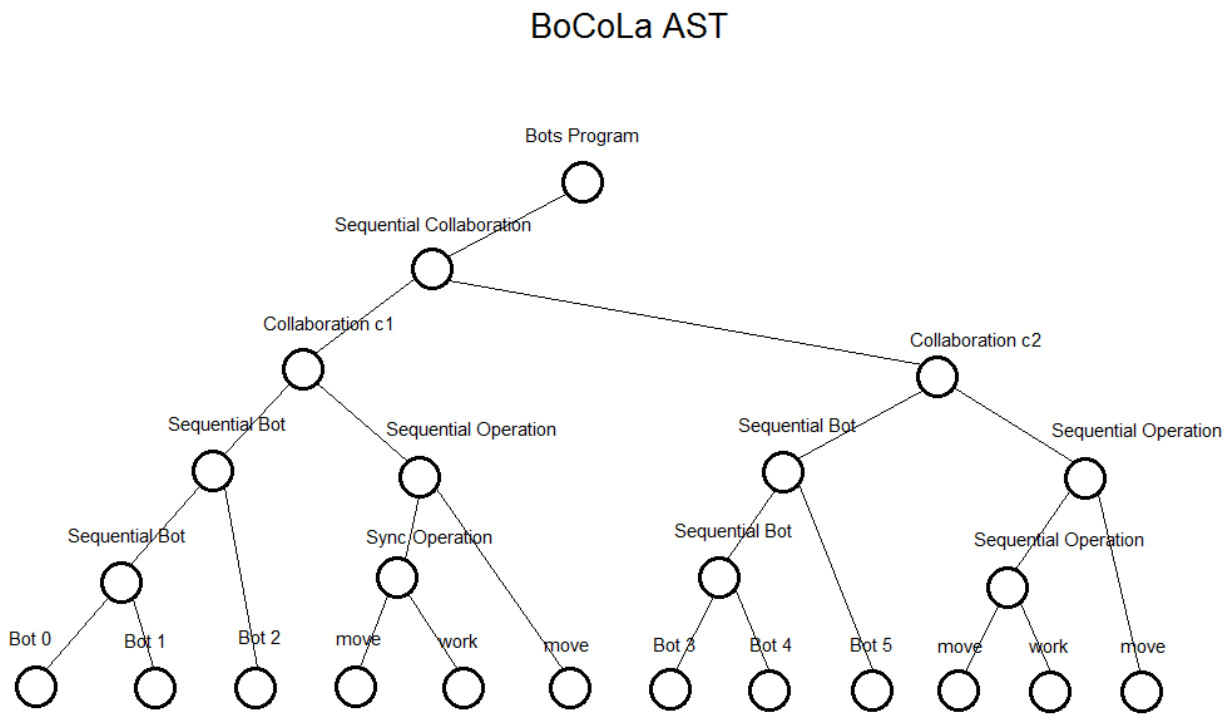

Figure 3.2 BoCoLa experiment 1 AST

The new Sync method that was added to the Parser class is shown in fig.3.3.

```
Synchronization parseSynchronization() throws SyntaxError{
        Synchronization sAST;// = parseCollaboration();
        accept(Token.SYNC);
        accept(Token.LPAREN);
        Operations o1 = parseOperation();
        accept(Token.COMMA);
        //Bot b1 = parseBot();
        Operations o2 = parseOperation();
        //Bot b2 = parseBot();
        accept(Token.RPAREN);
        sAST = new Synchronization(o1, null, o2, null);
        return sAST;
    }
```

Figure 3.3 code snippet of the Synchronization method in the Parser class

The translation of the BoCoLa code to java was made in the Encoder class and is printed in at text file and then copied into the bot program. An example of how the translation were done can be seen in figure 3.4. The str is printed out to the text file when it visits the particular method in the AST.

```
public Object visitCollaboration(Collaboration c, Object o) {
    c.ID.visit(this, o);
    c.BL.visit(this, o);
    c.OL.visit(this, o);

    //temp = str;
    str += "Collaborations"+"["+"No"+"]"+"="+"new Collaboration();"+"\r\n";
    str+="{\r\n";
    str +=temp;
    str+="}\r\n";

    temp="";
    return null;
}
```

Figure 3.4 code snippet of the visitCollaboration method in the Encoder class

The translated output of the Experiment 1 shown in figure 3.1 is shown in figure 3.5.

```
Collaborations[No]=new Collaboration();
{
Collaborations[No-1].include(new BotMethod.botMove(),Bot.bots[1],new BotMethod.botWork(),Bot.bots[2]);
Collaborations[No-1].include(new BotMethod.botMove(),Bot.bots[0]);
}
Collaborations[No]=new Collaboration();
{
Collaborations[No-1].include(new BotMethod.botMove(),Bot.bots[5]);
Collaborations[No-1].include(new BotMethod.botWork(),Bot.bots[3]);
Collaborations[No-1].include(new BotMethod.botMove(),Bot.bots[4]);
}
```

Figure 3.5 Translated java output from the Encoder class

The translated output was then pasted into the Collaboration method in the Collaboration class of the BotsSystem program that was modified to incorporate the new synchronization functionality.

4 – Runtime System Modification

Runtime system was changed to support a new BoCoLa feature – synchronization. Then we execute our BotSystem program, it includes bots into the lists.

By default BotSystem has the function **include,** however if we want to include bots which have to be synchronized. That is why we created a new function **include,** which has twice as many parameters than default, as shown in figure 4.1.

```
public void include(Method m1, Participant p1,Method m2, Participant p2) {
        dirList.add(size(), new dir(m1, p1, Participant.act.START_SYNC));
        dirList.add(size(), new dir(m2, p2, Participant.act.END_SYNC));
}
```

Figure 4.1 Synchronization between p1 and p2

Before adding to the **dirList**, we mark participant that it is synchronized (START_SYNC, END_SYNC). Then when the runtime system executes checking synchronization is easy, and then locks other participants.
Modifications to the BotsSystem is shown in the figure 4.2.

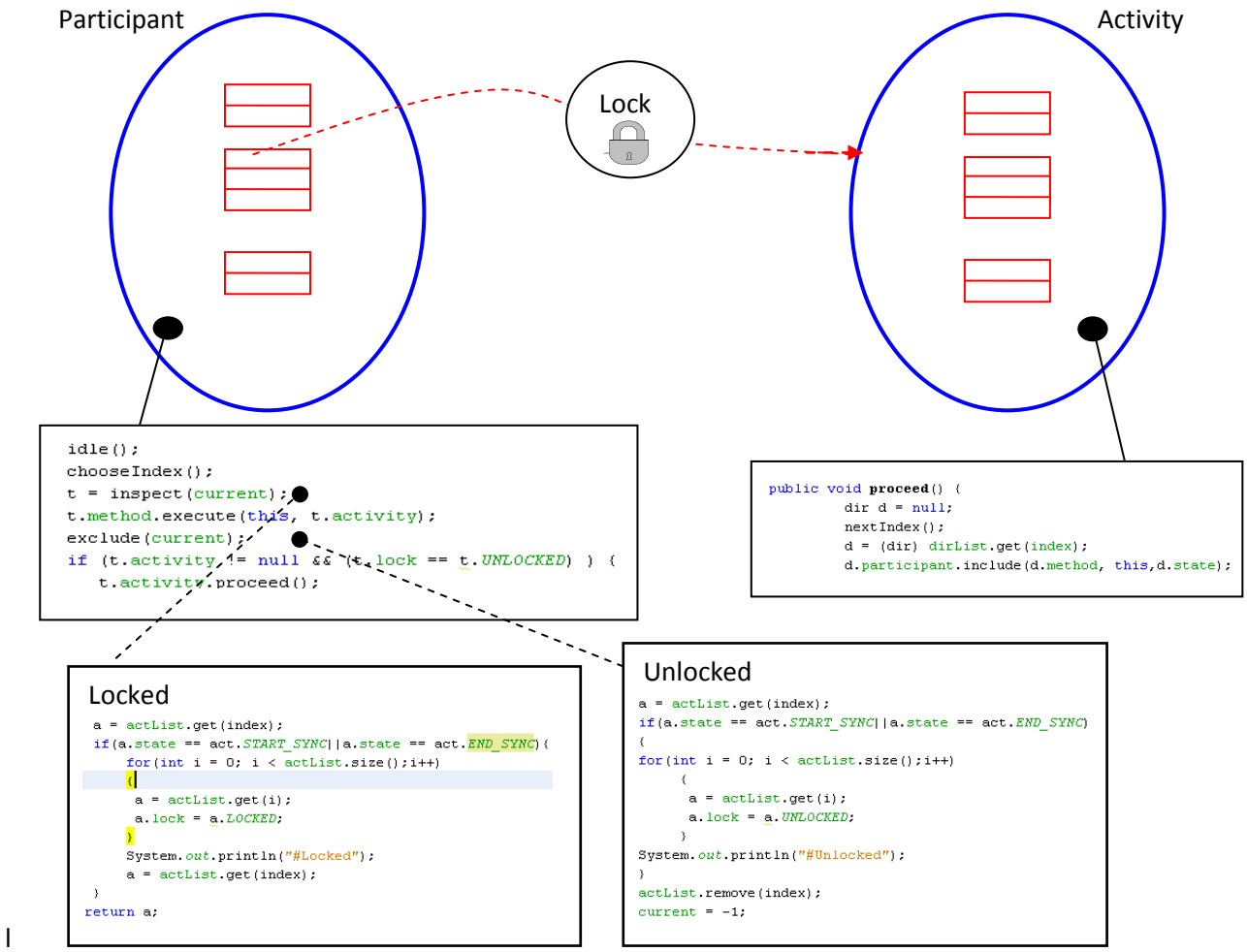

Figure 4.2 Run Time System (modified)

Figure 4.3 illustrates a list of participants. Synchronized participant has a pointer to another participant. When the synchronized participant is chosen and the other participants of the list are locked, and are again unlocked when the synchronised execution is finished. Synchronization is finished when there are no more participants in the execution list, this is indicated by a null pointer.

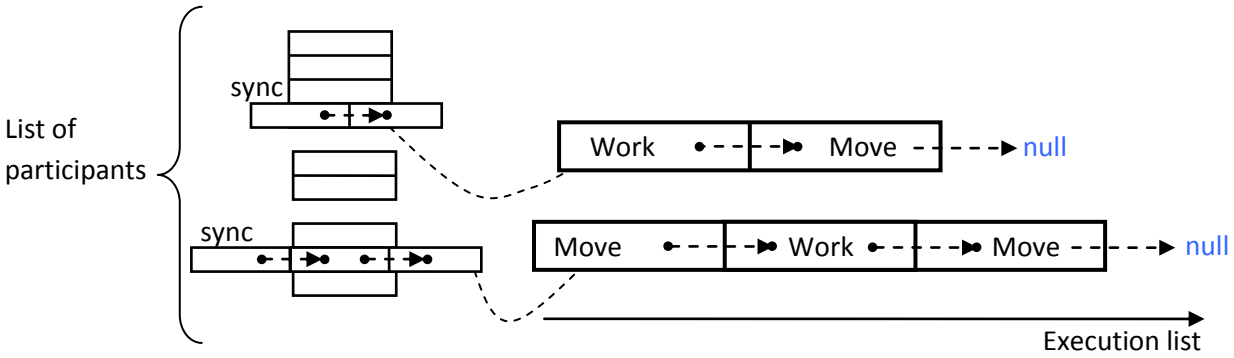

Figure 4.3 Synchronization between bots

5 – Example Experimentation

In the BoCoLa program we write program syntax in the text file, as shown in table 5.1. There are two collaborations c1 and c2. Collaboration c1 synchronized, it was done using the special keyword ***sync***.

Language	Syntax
BoCoLa	collaboration c1 between 0, 1, 2 [sync(move by 1, work by 2); move by 0;] collaboration c2 between 3,4,5 [move by 5; work by 3; move by 4;]

Java	Collaborations[No] = new Collaboration();
	{
	Collaborations[No-1].include(new BotMethod.botMove(),Bot.bots[1],new BotMethod.botWork(),Bot.bots[2]);
	Collaborations[No-1].include(new BotMethod.botMove(),Bot.bots[0]);
	}
	Collaborations[No]=new Collaboration();
	{
	Collaborations[No-1].include(new BotMethod.botMove(),Bot.bots[5]);
	Collaborations[No-1].include(new BotMethod.botWork(),Bot.bots[3]);
	Collaborations[No-1].include(new BotMethod.botMove(),Bot.bots[4]);
	}

Table 5.1 Translation from BoCoLa to Java

Then BoCola program text was saved in the text file and then executed the program. The program produced java code which can be used in a BotsSystem program. Java code from the BoCola program is shown in the table 5.1.The BoCoLa language is more structured and easier to read. To run the BotsSystem and test it, we need to copy java code into the system, compile and execute. The collaborations are shown in figure 5.1. Green collaboration between bot 1 and bot 2 are synchronized. Synchronized collaboration takes more time to execute, because it is necessary to wait and then execution can continue.

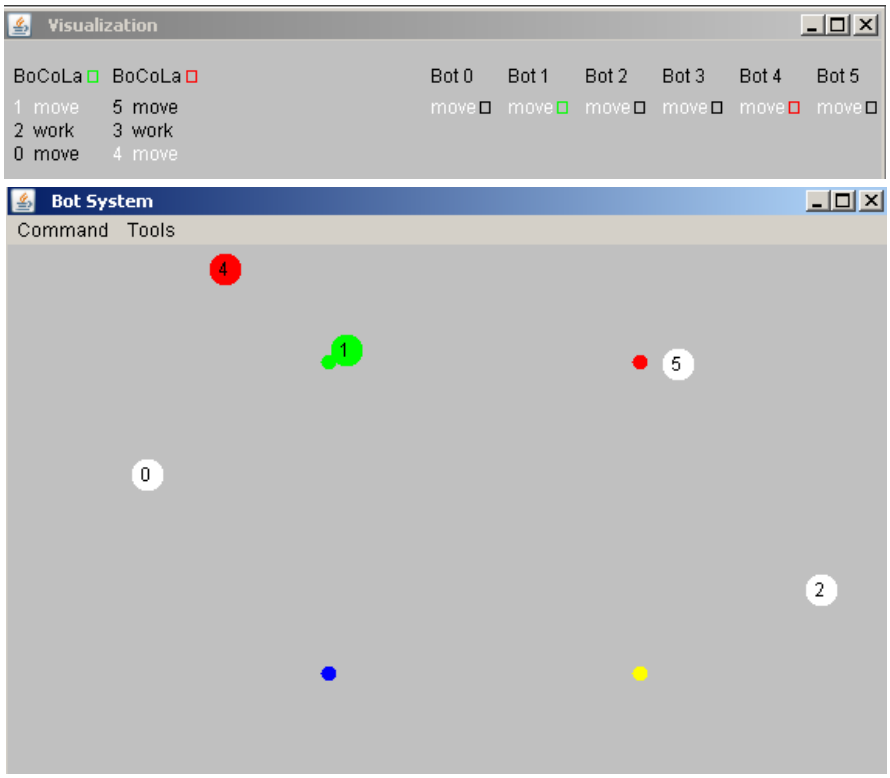

Figure 5.1 Bot System

By using the keyword **sync** it takes longer to execute, because of the wait. During synchronized bot execution no other bots execute. This guarantee right order of bots execution (move or/and work).

6 – Conclusion

The project team has developed a BoCoLa to java compiler with an extra addition of synchronization functionality. The team also modified the Runtime system to incorporate this functionality. An extension to make more bots collaborate was also discussed.

From the view point of the programming languages BoCoLa was created to satisfy particular area of applications, also it gives more abstraction of the language. If using directly java language we need take care about many things, like objects, instances, threads and so on. BoCoLa provide more abstract things to achieve the same result compared to java, and do not need to take care of task that are of no interest to us, we can concentrate to other things – more specific task solving.

7 – Appendix

The source code is included in the zip file and on the CD.

Setup of the program:

1. How to Compile from BoCoLa to Java?
    a) By default text file where we need to write BoCoLa code should be kept in disk "D:\" and name of text file should be "tam.2txt", if you want you can change the root by modifying "Main.java" file, which kept in: "…\SSE2Project\src\sse2project\Main.java".
    b) Then you can execute program which is: "…\SSE2Project\dist\SSE2Project.jar"
    c) If execution successful you get output file with java code which automatically saved to folder on root: ""…\SSE2Project\dist\Text.txt".
2. How to run BotsSystem program?
    a) Before start you need to modify:
        - Open java file which is on root: "…\BotsSystem\src\BotsSystem\Collaboration.java"
        - Find function "static public void initialize(){ … }"
        - Insert java code from file"Text.txt" into beginning of function "static public void initialize()".
        - Recompile project and execute

Printed results of AST tree:

visitBotsProgram
visitSequentialCollaboration
visitCollaboration
visitIdentifier<c1>
visitSequentialBot
visitSequentialBot
visitBot
visitIntegerLiteral<0>
visitBot
visitIntegerLiteral<1>
visitBot
visitIntegerLiteral<2>
visitSequentialOperation
visitOperations
visitSync
visitOperations
visitIdentifier2<move>
visitIntegerLiteral<1>
visitOperations
visitIdentifier2<work>
visitIntegerLiteral<2>
visitOperations
visitIdentifier2<move>
visitIntegerLiteral<0>
visitCollaboration
visitIdentifier<c2>
visitSequentialBot

visitSequentialBot
visitBot
visitIntegerLiteral<3>
visitBot
visitIntegerLiteral<4>
visitBot
visitIntegerLiteral<5>
visitSequentialOperation
visitSequentialOperation
visitOperations
visitIdentifier2<move>
visitIntegerLiteral<5>
visitOperations
visitIdentifier2<work>
visitIntegerLiteral<3>
visitOperations
visitIdentifier2<move>
visitIntegerLiteral<4>