SSE2-PLDE: BoCoLa

Bent Bruun Kristensen
bbk@mmmi.sdu.dk
Maersk Mc-Kinney Moller Institute
Spring 2010

BoCoLa

The purpose is to design, implement and experiment with a *collaboration* abstraction mechanism, *BoCoLa,* as a supplement to Java:

- A program (i.e. some collaboration's) in BoCoLa is translated to Java and added to an existing partial Java program to form a complete Java program. i.e.
  Bot System = "Collaboration"s + Bot System Skeleton + RunTime System.
- Collaborations describe which and how bots collaborate. Bots are visualized by an simulator system and are already programmed in the Bot System Skeleton.
- In Bot System Skeleton class Collaboration inherits from Activity and class Bot inherits from Participant. Activity and Participant are abstract classes in the RunTime System framework.

The project includes

- To *design* and *complete* your version of BoCoLa
- To *design* and *program* a translation from BoCoLa to Java
- To *modify* and *extend* the RunTime System
- To *illustrate* and *experiment* with examples in the Bot System

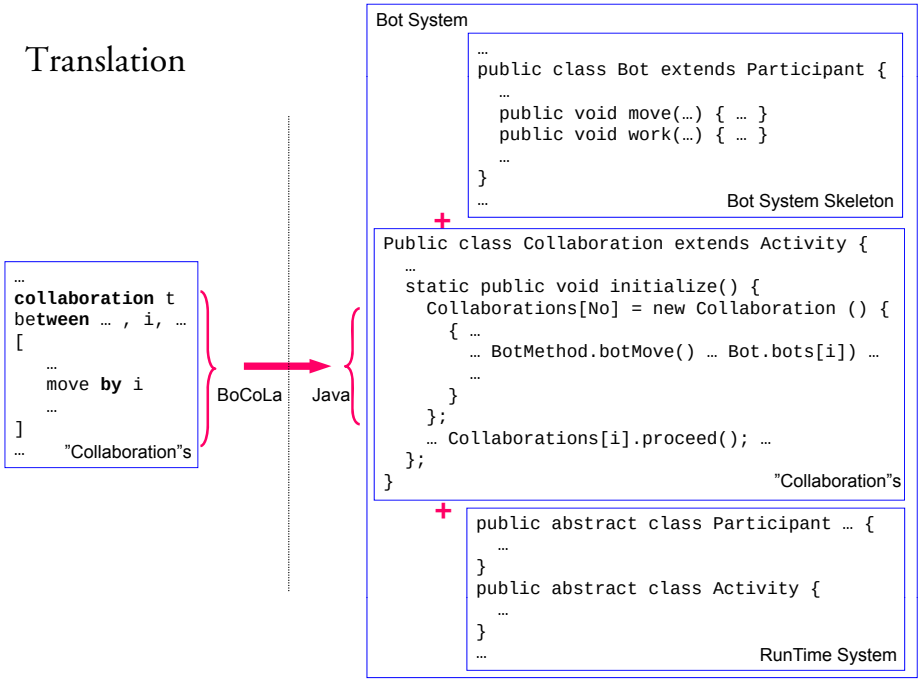
Example

```
collaboration c1
between 1, 2
[
   work by 1
   move by 2
]
```

```
collaboration c2
between 2, 3
[
   move by 3
   move by 2
   work by 3
   move by 2
]
```

```
collaboration c3
between 1, 2, 3
[
   work by 2
   work by 3
   move by 1
   work by 2
   move by 1
   move by 3
]
```

Translation



Project Description

Project Requirements
• The report includes sections according to the project parts described below.
• The report is in Danish or English, has approximately 10 pages, and includes no program listings (only a few program extracts of 10-20 lines may be included and must be explained)
• Two hard copies of the project report are delivered—a CD with program listings is included

Project Parts
1. Language Definition
2. Language Translation
3. Runtime System Modification
4. Example Experimentation

1. Language Design
• Design your version of BoCoLa: A definition of your BoCoLa in the form of a specific collaboration abstraction mechanism is required, including a description of the tokens and a description of a CFG. The contextual constraints and semantics are already given and are fixed
• A collaboration should only describe an object-like instance and not a class-like abstraction (i.e. no dynamic creation of collaboration instances)
• Design (i.e. specify syntax, contextual constraints and semantics) an extended version of BoCoLa with a *synchronization* construction of "`m1` **by** `i1`" and "`m2` **by** `i2`" meaning that `i1` executes `m1` and `i2` executes `m2` but neither of `i1` nor `i2` continues until both executions of `o1` and `o2` are completed
• Discuss alternative design proposals

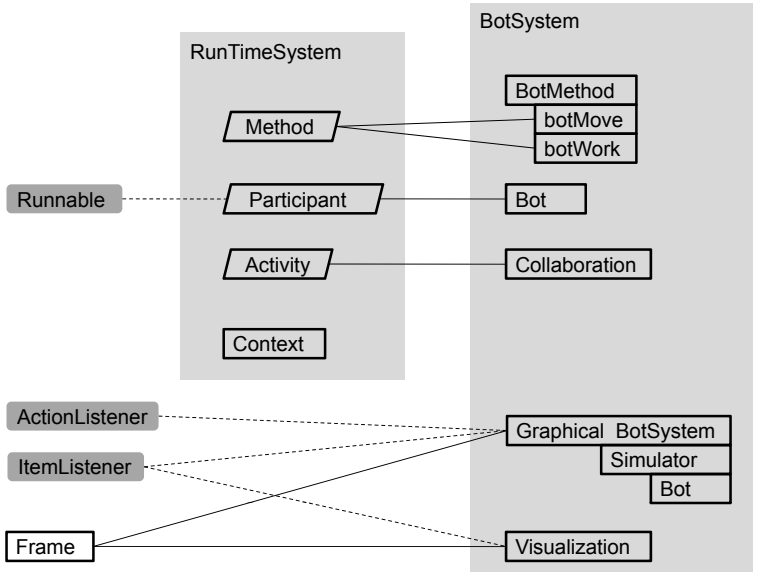
Project Description (cont.)

2. Language Translation
• A simple lexical analysis is required
  • Describe tokens
  • Program scanner
• A simple context-free analysis is required
  • Describe grammar and AST declarations
  • Program recursive descent parser and build AST structure
• A simple translation from BoCoLa to Java is required
  • Describe the form in Java to which your operation similar to "`m` **by** `i`" is translated
  • Describe VISITOR pattern for AST and program VISITOR to build this form in Java
• A simple runtime system in Java is available for the preliminary version of BoCoLa
• No contextual analysis is required and no error handling is required

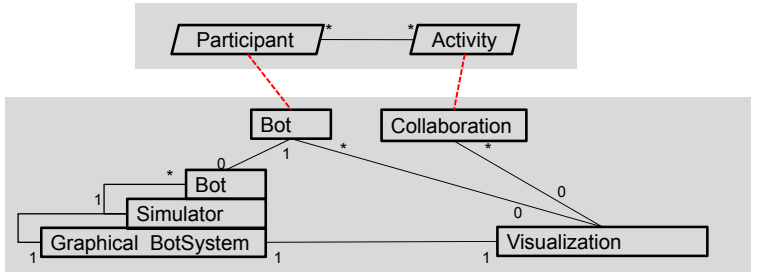
3. Runtime System Modification
• Modify Participant to make a random selection between operations to be executed (instead of the existing queue principle) and program this change
• Design and program the Runtime System also to support the extended version of CoBoLa including your version of the *synchronization* construction
• Discuss (but do not program) an extension of the *synchronization* construction with an arbitrary number of operations as arguments (and not only two)

4. Example Experimentation
• Describe some illustrative examples and program these in BoCoLa using collaborations
• Translate the examples to Java by your BoCoLa translator
• Execute and show the translated program and discuss the result

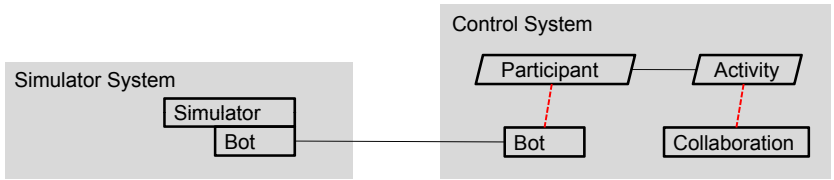
Bot System: Class Extension & Interface Implementation



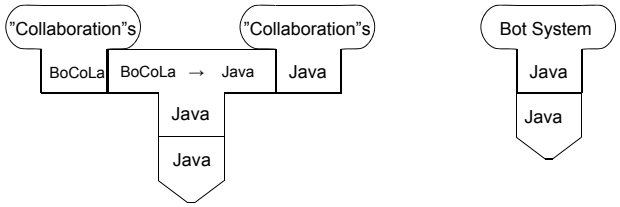
Bot System: Class Model



Bot System: Architecture



Translation and Interpretation



Bot System = "Collaboration"s + Bot System Skeleton + RunTime System

Language

```
collaboration c
between "sequence of bot numbers"
[
   "sequence of move by i or work by i"
]
```

Syntax:
- BoCoLa is a list of collaboration descriptions
- A collaboration specifies a sequence of numbers of collaborating bots followed by a sequence of operations involving these bots

Contextual Constraints:
- For any collaboration the bot numbers in the sequence must be different
- Bot number "i" in the operation "`m by i`" must be declared in the sequence of bot numbers
- The names of collaborations, "c", must be different

Semantics:
- Collaborations are maintained concurrently:
  - A collaboration states *which* bot executes *which* method *when*: Method "m" in the sequence "`m by i`" is executed by bot "i". Method "m" is either "`move`" or "`work`".
  - After maintaining an operation the collaboration proceeds to the next operation in the sequence.
  - Maintaining the operation "`m by i`" means that bot "i" executes method "m".
  - The sequence of operations in a collaboration is implicitly enclosed in a forever loop
- Bots execute concurrently:
  - A bot executes at most one method at a time and without interruptions.

Syntax

BoCoLa = (N, ∑, P, BotsProgram), where

N = { BotsProgram CollaborationList Collaboration BotList Bot
OperationList Operation }

∑ = { **collaboration between by**
; **,** **[** **]**
identifier integer-literal
}

P = { BotsProgram ::= CollaborationList

CollaborationList ::= Collaboration{Collaboration}*
Collaboration::= **collaboration** identifier BotList **[** OperationList **]**

BotList ::= **between** Bot { **,** Bot }*
Bot::= integer_literal

OperationList ::= Operation { **;** Operation }*
Operation ::= identifier2 **by** integer_literal
}

(where identifier2 is either "`move`" or "`work`")

Example

```
collaboration c1
between 1, 2
[
   work by 1
   move by 2
]
```

```
collaboration c2
between 2, 3
[
   move by 3
   move by 2
   work by 3
   move by 2
]
```

```
collaboration c3
between 1, 2, 3
[
   work by 2
   work by 3
   move by 1
   work by 2
   move by 1
   move by 3
]
```

```
collaboration c1
between 1 2
[
   work by 1;
   move by 2
]
```

```
c1: collaboration
between 1 and 2  where
   1 please work
   2 please move
```

```
   [
      c1: 1 2
      1 :: work
      2 :: move
   ]
```

```
collaboration c1
between 1 + 2
[
   work by 1;
   move by 2;
]
```

```
c1: (1 2)
[
   1 ~ work
   2 ~ move
]
```

(c1, (1, 2), ((1, work), (2, move)))

Alternative Syntax?

BoCoLa ::= CL

CL ::= C { C }*
C ::= **collaboration** identifier BL **[** OL **]**

BL ::= **between** B
B::= integer-Literal { **,** integer-Literal }*

OL ::= O { **;** O }*
O ::= identifier2 **by** integer-Literal

BoCoLa ::= { **collaboration** identifier
between{ integer-Literal **,** }*
[{ identifier2 **by** integer-Literal **;** }*
] }*

BoCoLa ::= { C B **[** { O }* **]** }*

C ::= **collaboration** identifier

B ::= **between** { integer-Literal **,** }*

O ::= identifier2 **by** integer-Literal **;**

BoCoLa Grammar (Concrete & Abstract)

BotsProgram ::= CollaborationList	Program
	CollaborationList
CollaborationList ::= Collaboration {Collaboration }*	SequentialCollaboration
Collaboration ::= **collaboration** identifier BotList **[** OperationList **]**	Collaboration
	BotList
BotList ::= **between**Bot { **,** Bot }*	SequentialBot
Bot ::= integer-Literal	Bot
	OperationList
OperationList ::= Operation { **;** Operation }*	SequentialOperation
Operation ::= identifier2 **by** integer-Literal	Operation

AST

```
public abstract class AST {
  …
}

public class Program extends AST {
  …
  public CollaborationList C;
}
```

```
public abstract class CollaborationList extends AST {
  ...
}

public class SequentialCollaboration extends CollaborationList {
  ...
  public CollaborationList CL1, CL2;
}

public class Collaboration extends CollaborationList {
  ...
  public BotList B;
  public OperationList O;
}
```

BotsProgram ::= CollaborationList	Program
	CollaborationList
CollaborationList ::= Collaboration {Collaboration }*	SequentialCollaboration
Collaboration ::= **collaboration** identifier **[** BotList OperationList **]**	Collaboration

RD

BotsProgram ::= CollaborationList

CollaborationList ::= Collaboration {Collaboration }*
Collaboration ::= **collaboration** identifier BotList **[** OperationList **]**

BotList ::= **between**Bot { **,** Bot }*
Bot ::= integer-Literal

OperationList ::= Operation { **;** Operation }*
Operation ::= identifier2 **by** integer-Literal

```
parseBotsProgram() { ... }
parseCollaborationList() { ... }
parseCollaboration() { ... }
parseBotList() { ... }
parseBot() { ... }
parseOperationList() { ... }
parseOperation() { ... }
```

RD

```
parseBotsProgram() {
   ...
   parseCollaborationList();
   ...
}
```

BotsProgram ::= CollaborationList

CollaborationList ::= Collaboration {Collaboration }*
Collaboration ::= **collaboration** identifier BotList **[** OperationList **]**

```
parseCollaborationList() {
   parseCollaboration ();
   while (currentToken.kind == Token.COLLABORATION) {
      parseCollaboration ();
   }
}

parseCollaboration () {
   accept(Token.COLLABORATION);
   accept(Token.ID);
   parseBotList();
   accept(Token.LEFTS);
   parseOperationList();
   accept(Token.RIGHTS);
}
```

RD + AST

```
Program parseBotsProgram() {
   ...
   Program p = parseCollaborationList();
   ...
   return p;
}
```

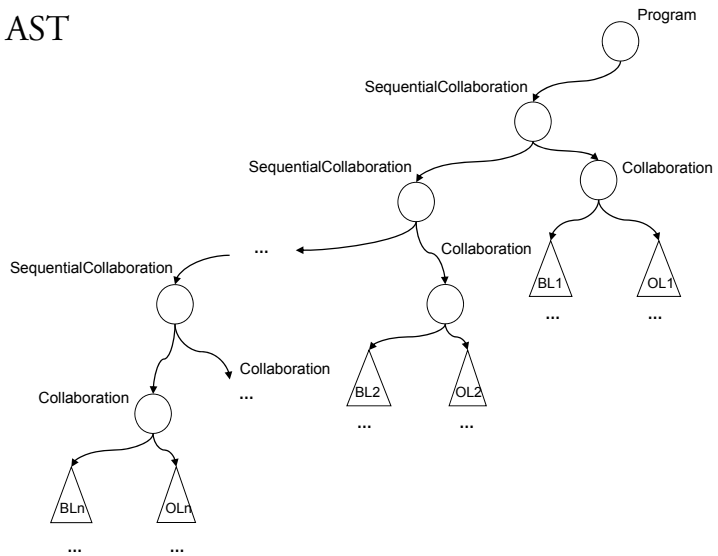
BotsProgram ::= CollaborationList

```
CollaborationList parseCollaborationList() {
   CollaborationList c1 = parseCollaboration ();
   while (currentToken.kind == Token.COLLABORATION) {
      CollaborationList c2 = parseCollaboration ();
      c1 = new SequentialCollaboration(c1, c2, …);
   }
   return c1;
}
```

CollaborationList ::= Collaboration {Collaboration }*
Collaboration ::= **collaboration** identifier BotList **[** OperationList **]**

```
CollaborationList parseCollaboration () {
   accept(Token.Collaboration);
   accept(Token.ID);
   BotList  bl = parseBotList();
   accept(Token.LEFTS);
   OperationList ol = parseOperationList();
   accept(Token.RIGHTS);
   return new Collaboration(bl, ol, …);
}
```

AST



VISITOR

Program

CollaborationList
SequentialCollaboration
Collaboration

BotList
SequentialBot
Bot

OperationList
SequentialOperation
Operation

```
public interface Visitor {
   public object visitProgram (BotsProgram prog, Object arg);

   public object visitSequentialCollaboration (SequentialCollaboration c, Object arg);
   public object visitCollaboration (Collaboration c, Object arg);

   …
}
```

VISITOR

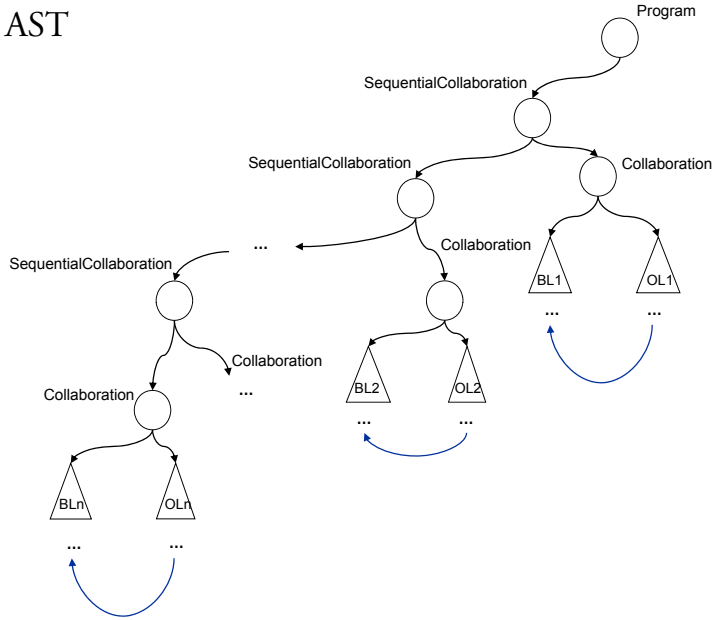
```
public abstract class AST {
  ...
  public abstract Object visit (Visitor v, Object arg);
}
```

```
public class Program extends AST {
  ...
  public Object visit (Visitor v, Object arg)  {
    return (v.visitProgram(this, arg);
}
```

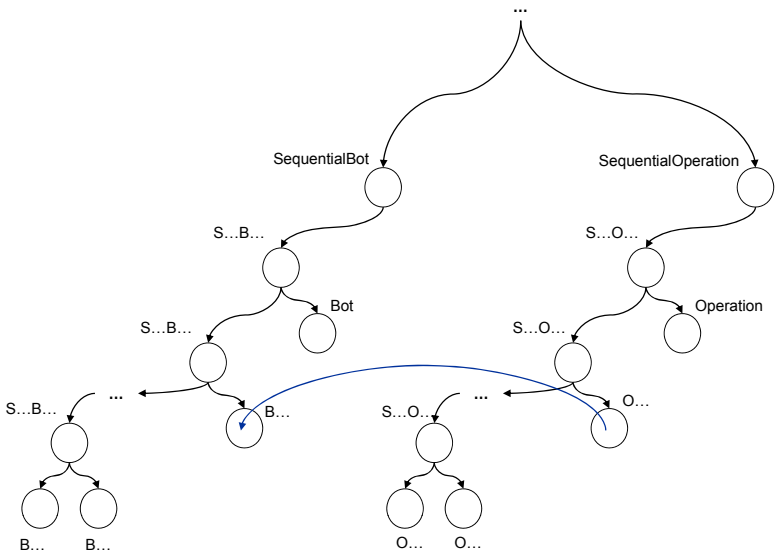
```
public class SequentialCollaboration extends CollaborationList {
  ...
  public Object visit (Visitor v, Object arg)  {
    return (v.visitSequentialCollaboration (this, arg);
}

public abstract class Collaboration extends CollaborationList {
  ...
  public Object visit (Visitor v, Object arg)  {
    return (v.visitCollaboration (this, arg);
}
```

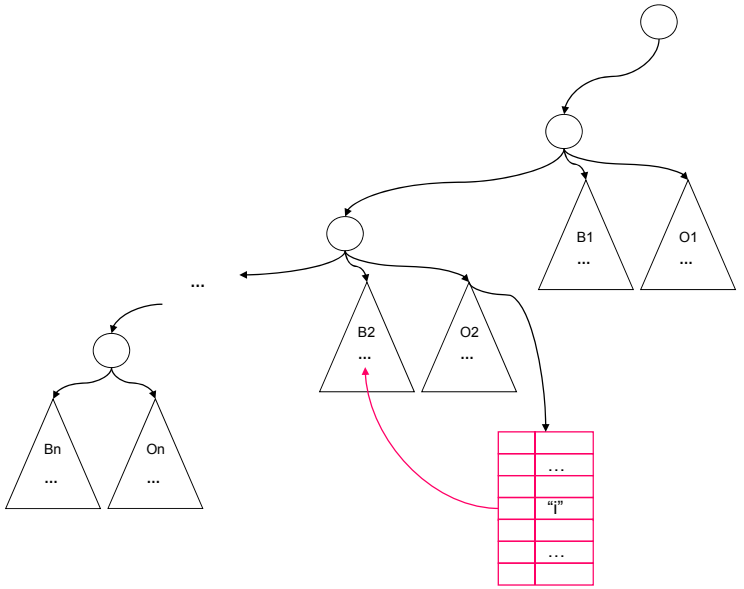
AST



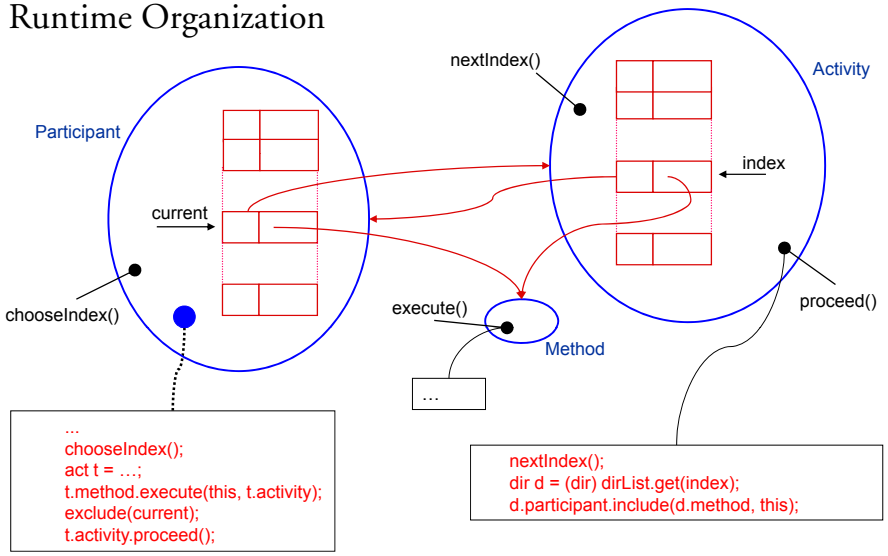
AST



AST



Runtime Organization



Dynamic: actList [activity, method]

Static : dirList [participant, method]

Runtime Organization

```
public abstract class Participant implements Runnable {
  ...
  synchronized public void chooseIndex() { … };
  …
  synchronized public void include(Method m, Activity a) { … };
  …
  synchronized public void exclude(int index) { … };
  …
  public void run()
  {
    while (this != null) {
      ...
      chooseIndex();
      act t = …;
      t.method.execute(this, t.activity);
      exclude(current);
      t.activity.proceed();
    };
  };
  …
  public class act {
    ...
    Method method;
    Activity activity;
  };
  ...
  ArrayList <act> actList = new ArrayList <act>(20);
  int current = -1;
};
```

```
public abstract class Activity {
  …
  public void proceed() {
    nextIndex();
    dir d = (dir) dirList.get(index);
    d.participant.include(d.method, this);
  };
  …
  void nextIndex() {
    …
  };
  …
  public class dir {
    …
    Method method;
    Participant participant;
  };
  …
  ArrayList <dir> dirList = new ArrayList <dir>(20);
  int index = -1;
};
```

```
public abstract class Method {
  public abstract void execute(Participant p, Activity a);
  …
};
```