Introduction to Aspectual Collaborations

January 29, 2002

foo

- blah
- blah

Hello World

```
1  collab hw_greet;
2  participant Greeter {
3    void sayit () {{ System.out.println("Hello World!"); }}
4  }
5  collab hw_main;
6  participant Main {
7    expected void doit();
8    public static void main(String[] args) {{
9      Main m = new Main();
10     m.doit();
11   }}
12 }
13 collab helloworld;
14 participant X {}
15 attach hw_greet, hw_main {
16   X += Greeter, Main {
17     provide doit with sayit;
18     export main;
19   }
20 }
```

Syntax

- Looks like java with funny reserved words.
- expected is like abstract, but
  - is **provided** rather than overridden
  - doesn't hinder instantiation of the class
  - fields can be expected as well
- Double braces bracket 100% java, but that is just to avoid having to parse it.
- We generate a very similar java, which is then compiled.

Just out of curiosity: the compiled java

```
1  package hw_greet; // NOTICE that each of these compiles and is plain java
2  class Greeter {
3    void sayit () { /*beginverb*/
4      System.out.println("Hello World!");
5    /*endverb*/ }
6  }
7  package hw_main; // NOTICE although they may throw exceptions
8  class Main {
9    /*expected*/ void doit() { /*beginverb*/
10     throw new IllegalStateException("Unprovided Expected Method");
11   /*endverb*/ }
12   public static void main(String[] args) { /*beginverb*/
13     Main m = new Main();
14     m.doit();
15  /*endverb*/ }
16 }
17 package helloworld;
18 class X {}       // NOTICE: X is empty
```

Linking it all together

- Give X a body (`main`).
- point `doit` to `sayit`

Attachment

```
1 attach hw_greet, hw_main {
2   X += Greeter, Main {
3     provide doit with sayit;
4     export main;
5   }
```

- We insert `hw_greet` and `hw_main` in to the **host** collaboration (`helloworld`). *(line 1)*
- We **map** the participant names of the **constituent** collaborations to the host collaboration. *(line 2)*
- We provide an implementation of the expected doit method. NB: signatures have to be exactly the same. *(line 3)*
- We export the **members**. Members of an inserted collaboration are by default unexported. We need to explicitly export any members we want to appear in the interface of the host collaboration. *(line 4)*

- Each time we insert some collaborations and deal with their members, we call it an **attachment.** This example has one attachment, but two inserted collabs.

Running it

```
shalmanser(112): ../acc collab: ../examples/helloworld.collab
added /tmp/classes to CLASSPATH
Processing ../examples/helloworld.collab
processing collaboration hw_greet
  process source
  compiling
  munge
processing collaboration hw_main
  process source
  compiling
  munge
    dumping
      getAllCollabClasses(hw_main)
      writing /tmp/classes/hw_main/Main.class
processing collaboration helloworld
  process source
  compiling
  munge
    dumping
      getAllCollabClasses(helloworld)
      writing /tmp/classes/helloworld/X.class
shalmanser(113): CLASSPATH=/tmp/classes java helloworld.X
Hello World!
```

Quick Vocabulary summary

- collaboration: a closed set of participants. a generalization of java package
- participant: a generalisation of a java class with additional features
- host collaboration : the collaboration we are creating
- constituent collaboration : the collaborations we are inserting
- members : fields and methods in a participant
- expected member : a member that hasn't been specified yet, appart from signature
- provided member : a member that is not expected

Another example

```
1 collab vars;
2 participant Var {
3     Bar b;
4 }
5 participant Bar { }
```

We are going to:

- add getters and setters
- add back pointers from Bar to Var
- *maintain* the back pointers
- Look at various composition strategies

Getters and Setters

```
1  collab getset;
2  participant Source {
3    expected Target field;
4    public Target get() {{ return field; }}
5    public void set(Target t) {{ field = t; }}
6  }
7  participant Target { }
8
9  collab varsNgs;
10 participant V { } // Evil
11 participant B { } // Evil
12 attach getset, vars {
13   V += Var, Source {
14     provide field with b;
15     export get as getB;
16     export set as setB;
17   }
18   B += Bar, Target { }
19 }
```

Get and Set Explanation

- getset is simple
- we make a new `result` collaboration
- laboriously make placeholders: V and B,
- insert getset and vars into these
- V = { Var, Source }, while B = { Bar, Target}
- this is why `Bar b` can be provided to `Target field`

Apology

You shouldn't have to make V and B yourselves, but I haven't had time to automate this yet.

Adding and Maintaining Backpointers

```
1  collab backPtr;
2  participant S {
3    expected T getT();
4    aspectual RetVal setter(SetMeth sm) {{
5      RetVal rv = sm.invoke();
6      T targ = getT();
7      if (targ.back != null) {
8        System.err.println("Dropping old back pointer");
9      }
10     targ.back = this;
11     return rv;
12   }}
13 }
14 participant T {
15   public S back;
16 }
```

Aspectual methods

- Invoked implicitly – unlike expected methods
- Don't match the method they will be wrapping
- Methods only – no aspectual invocations on field refs. IE: our join point is method invocation.
- Explicitly invoke the **host method** – can choose not to, or when. In this case, we invoke the host method first: this is after advice.
- The host method is captured as a java object, with one method: invoke(). This returns a RetVal object, which the aspectual method must return.
- The names RetVal and SetMeth are taken from the declared signature of the aspectual method. The classes to implement these are created automatically. These cannot be mapped by

the user (ie they are local to the collaboration).

Attaching the aspectual method

```
1  collab varsNgsNbp;
2  public participant  Source { }
3  public participant  Target { }
4  attach varsNgs, backPtr {
5    Source += V, S {
6      provide getT with getB;
7      around setB do setter;
8      export getB as getTarget;
9      export setB as setTarget;
10   }
11   Target += B, T {
12     export back;
13   }
14 }
```

Around setB do setter

- We want to replace setB with the setter method, but they have different signatures.
- Instead, we automatically generate a method with the same signature as setB, but which calls setter.
- Our generated method also creates the SetMeth object that needs to be passed to setter. In addition, we need to unpack RetVal and extract any returned object – in this case, the return is void, so that is unnecessary.
- SetMeth.invoke() calls setB with the intented argument.

```
1  package test;
2  import varsNgsNbp.*; // since collaborations are java, just import!
3  class M {
4    public static void Main(String args[]) {{
5      Source s1 = new Source(); // which is why these participants were public
6      Source s2 = new Source();
7      Target t = new Target();
8      s1.setTarget(t); // invokes not just the setter, but the aspectual method
9      System.err.println(s1+" has target "+s1.getTarget()+
10                                " with back pointer "+s1.getTarget().back);
11     s2.setTarget(t);
12     System.err.println(s2+" has target "+s2.getTarget()+
13                                " with back pointer "+s2.getTarget().back);
14
15     System.err.println(s1+" has target "+s1.getTarget()+
16                                " with back pointer "+s1.getTarget().back);
17   }}
18 }
```

Re-use

That export of back as a variable is kinda ugly. Let's add getters and setters to it:

```
1  collab varsNgsNbpNgs;
2  public participant  Source { }
3  public participant Target { }
4  attach varsNgsNbp, getset {
5    Src += varsNgsNbp.Source, getset.Target { // NB fqcn
6      export getTarget;
7      export setTarget;
8    }
9    Trg += varsNgsNpb.Target, getset,Source {
10     provide field with back;
11     export get as getSource;
12     // let's not provide the setter
13   }
14 }
```

And since we don't export back, it is gone from our sight. Good!

Composition review: Accumulation

- We have built up more and more functional collaborations: vars, varsNgs, varsNgsNbp, varsNgsNbpNgs.
- We've *accumulated* behavior bit by bit.

(picture)

Composition alternative: parallel

```
1  collab allAtOnce;
2  public participant  Source { }
3  public participant Target { }
4  attach vars , getset , backPtr { first :
5    Source += Var, getset.Source, S {
6      provide field  with b;
7      around set do setter ;
8      provide getT with get;
9      export get as getTarget;
10     export set as setTarget;
11   }
12   Target += Bar, getset.Target, T {  }
13 }
14 attach getset { second:
15   Source += getset.Target {  }
16   Target += getset.Source {
17     provide field  with  first :back;
18     export get as getSource;
19     export set as setSource;
20  }
21 }
```

Composition review: Parallel

- We add it all at once.
- Can get messy, keeping track of what goes where
- each collaboration can be added at most once per attach clause, so getset needs two clauses.
- We can refer to members from other attachments by naming each attachment.
- Attachment names are only visible within the host collaboration (cannot be exported).

(picture)

Composition alternative: parallel variation

```
1  collab varNinsert;
2  public participant Var {  Bar b; }
3  public participant Bar { }
4  attach getset , backPtr {  first :
5    Var += Source, S {
6      provide field  with varNinsert:b;
7      around set do setter ;
8      provide getT with get;
9      export get as getTarget;
10     export set as setTarget;
11   }
12   Bar += Target, T { }
13 }
14 attach getset {
15   Var += Target { }
16   Bar += Source {
17     provide field  with  first :back;
18     export get as getSource;
19     export set as setSource;
20  }
21 }
```

Composition review: Parallel 1.5

- So it's parallel too.
- The point is that we don't always have to insert into an **empty** host collaboration.
- The host collaboration's members are accessed as if they were inserted in a previous attach clause. The name of the attachment is the same as the collaboration.

Composition alternative: the good way

```
1  collab getsetNbp;
2  participant Src {}
3  participant Trg {}
4  attach getset , backPtr { bkptr:
5    Src += Source, S {
6      export field ; // export of expected method
7      around set do setter ;
8      provide getT with get;
9      export get as getTrg;
10     export set as setTrg;
11   }
12   Trg += Target, T { }
13 }
14 attach getset {
15   Src += Target { }
16   Trg += Source {
17     provide field with bkptr:back;
18     export get as getSrc;
19   }
20 }
```

the good way, cntd, and Review

```
1  collab varsNgoodway;
2  public participant  Source { }
3  public participant  Target { }
4  attach vars , getsetNbp {
5    Source += Var, Src {
6      provide field  with b;
7      export getTrg as getTarget;
8      export setTrg as setTarget;
9    }
10   Target += Bar, Trg {
11     export getSrc as getSource;
12   }
13 }
```

- We've built up a composite collaboration which only expects a field, and will add getters and setters and a back pointer, and maintain them behind the scenes.
- we need to export the expected field, else we would be unable to provide it (as only exported members are visible).

(picture)

Recap

- Collaborations are compiled separately
- Collaborations are attached to a host by attach clauses
- Every member of an attached collaboration must be exported to be visible outside the host collaboration.

Homework!

Aspect J container example

- I have implemented this in ACs.
- luckily, this works at least.
- We have several nested containers, and we want to make sure their weights are not over the limit.
- Several pages:

```
1  collab result;
2  import java.util.*;
3
4  public abstract participant Item {
5    String name;
6    Container container;
7    public abstract int check();
8  }
9
10 public participant Simple extends Item {
11   int weight;
12   public static Simple make(String n,int w) {{
13     Simple res = new Simple();
14     res.name = n;
15     res.weight = w;
16     return res;
17   }}
18   public int check() {{
19       System.out.println("Simple object "+name+" weighs "+weight);
20       return weight;
21   }}
22 }
```

```
1  public participant Container extends Item {
2    Vector contents;
3    int capacity;
4    public static Container make(String n,int c) {{
5      Container res = new Container();
6      res.name = n;
7      res.capacity=c;
8      return res;
9    }}
10
11   public void addItem(Item i) {{
12     if (contents == null) {
13       contents = new Vector();
14     }
15     contents.add(i);
16   }}
```

```
1  // container continued
2    public int check() {{
3      Iterator it=contents.iterator();
4      int total = 0;
5      while(it.hasNext()) {
6        Item child =  (Item)it.next();
7        total+=child.check();
8      }
9      System.out.println("Container "+name+" weighs "+total);
10     if(total>capacity){
11       System.out.println("Container "+name+" overloaded");
12     }
13     return total;
14   }}
15 }
```

```
1  public participant Main {
2    static public void main(String args []) throws Exception {{
3        Container c1=Container.make("Container 1",5);
4        Container c2= Container.make("Container 2",1);
5        Container c3= Container.make("Container 3",1);
6        Simple apple= Simple.make("apple",1);
7        Simple pencil= Simple.make("pencil",1);
8        Simple orange= Simple.make("orange",1);
9        Simple kiwi= Simple.make("kiwi",1);
10       Simple banana= Simple.make("banana",1);
11
12       c3.addItem(kiwi);
13       c2.addItem(c3);  c2.addItem(apple);
14       c1.addItem(orange); c1.addItem(pencil); c1.addItem(c2);
15
16       c1.check();
17       c1.addItem(banana);
18       c1.check();
19   }}
20 }
```

- This is of course just a plain java implementation.
- Notice how we use factory methods rather than constructors with arguments. This is beacause acc ignores constructors write any constructor. (A known bug).

```
CLASSPATH=/tmp/classes java result.Main
Simple object orange weighs 1
Simple object pencil weighs 1
Simple object kiwi weighs 1
Container Container 3 weighs 1
Simple object apple weighs 1
Container Container 2 weighs 2
Container Container 2 overloaded
Container Container 1 weighs 4
Simple object orange weighs 1
Simple object pencil weighs 1
Simple object kiwi weighs 1
Container Container 3 weighs 1
Simple object apple weighs 1
Container Container 2 weighs 2
Container Container 2 overloaded
Simple object banana weighs 1
Container Container 1 weighs 5
```

I've also implemented a few aspectual collaborations:

1. one to keep track of which Items are constained in which containers
2. one to cache the calculated weight of containers.

```
1  collab backlink;
2  participant Parent {
3    aspectual RV childset(ChildSetterMethod csm) {{
4      System.err.println("set child");
5      csm.child.parent = this;
6      return csm.invoke();
7    }}
8  }
9  participant Child {
10   Parent parent;
11   Parent getParent() {{ return parent; }}
12 }
13 participant ChildSetterMethod {
14   // partial requirement on what can be wrapped
15   expected Child child;
16 }
```

```
1  collab caching;
2  participant I { expected C getC(); }
3  participant C extends I {
4    RV2 cached;
5    void clearCache() {{
6      System.err.println("clear_cache");
7      cached = null;
8    }}
9    aspectual RV invalidate(EM e) {{
10     RV retval = e.invoke();
11     C c = this;
12     while (c != null) {
13       c.clearCache();
14       c = c.getC();
15     }
16     return retval;
17   }}
18   aspectual RV2 cache(EM2 e) {{
19     if (cached == null) cached = e.invoke();
20     else System.err.println("using_cached_value!");
21     return cached;
22   }}
23 }
```

attach it to the non-caching implementation (result)

```
1  attach backlink, caching {
2    Container += Parent, C {
3      around result:void addItem(Item child) do childset;
4      around result:addItem do invalidate;
5      around result:check do cache;
6    }
7
8    Item += Child, I {
9      provide getC with getParent;
10   }
11 }
```

```
set child
clear cache
set child
clear cache
set child
clear cache
set child
clear cache
set child
clear cache
set child
clear cache
Simple object orange weighs 1
Simple object pencil weighs 1
Simple object kiwi weighs 1
Container Container 3 weighs 1
Simple object apple weighs 1
Container Container 2 weighs 2
Container Container 2 overloaded
Container Container 1 weighs 4
set child
clear cache
Simple object orange weighs 1
Simple object pencil weighs 1
using cached value!
Simple object banana weighs 1
Container Container 1 weighs 5
```

assignment

My version uses parallel attachment. You should do two things:

- create a composite collaboration of the two aspects, and attach that
- manually work implement an attachment (either of my example, or your own), and show that it produces the same output when run as this example does.