January 29, 2002

Introduction to Aspecual Collaborations
```java
public static void main(String[] args) {
    System.out.println("Hello World!");
}
```
We generate a very similar Java, which is then compiled.

Having to parse it.

Double braces break the 100% Java, but that is just to avoid

- Hedges can be expected as well
- Doesn’t hinder instantiation of the class
- Is provided rather than overridden
- Is expected is like abstract, but
- Looks like Java with funny reserved words.

**Syntax**
Just out of curiosity: the compiled Java
point dot to say it

Give X a body (main).

Linking it all together
host collaboration. (line 4)

export any members we want to appear in the interface of the

members. Members of an imported

NB: Signatures have to be exactly the same. (line 8)

We provide an implementation of the expected dot method.

We export the members. Members of an imported

We provide an implementation of the expected dot method.

We map the constituent names of the constituent

We insert hw-greet and hw-main in to the host collaboration

helptomeworld. (line 8)

We insert hw-greet and hw-main in to the host collaboration

Attachment

```python
{ 5
  export
  main:
  provide dot with signal:
} Main
X = `Greeter`, hw-main
attach `hw-greet`, hw-main
```
attachment, but two inserted colIab.

members, we call it an attachment. This example has one

Each time we insert some collaborations and deal with their
Running it:
provided member : a member that is not expected
append from signature
expected member : a member that hasn't been specified yet
members : fields and methods in a participant
constituent collaboration : the collaborations we are inserting
host collaboration : the collaboration we are creating
features
participant: a generalization of a Java class with additional
Java package
collaboration: a closed set of participants. a generalization of

Quick Vocabulary Summary
Look at various composition strategies

maintain the back pointers

add back pointers from Bar to Var

add getters and setters

We are going to:

```cpp
{ } participate Bar
    { } Bar
    Bar p;
} participate Var
i collab vars;
```

Another example
Getters and Setters
time to automate this yet.

You shouldn't have to make A and B yourself, but I haven't had

Apology

this is why Bar \( b \) can be provided to Target Field

\[
\{ \text{Var, Source, while B, Target} \} = \text{A, } \text{B}
\]

• Insert \( \text{getset} \) and \( \text{vars into these} \)

• labulously make placeholders: \( \text{A and B,} \)

• we make a new result collaboration

• \( \text{getset is simple} \)

Get and Set Explanation
```java
{ public
} Participant T
{ {

    return rv;
    target.back = this;
    { System.err.println("Dropping old back-pointer");
        if (target.back != null)
            target = target.get();
        retval = sm.invoke();
    }
}}

exception T.get();
}
```

Adding and Maintaining Backpointers
These are created automatically. These cannot be mapped by signature of the aspectual method. The classes to implement the names RetVal and Set Meth are taken from the declared aspectual method must return. This returns a RetVal object, which is the host method is captured as a Java object, with one after advice. 

When, in this case, we invoke the host method first: this is Explicitly invoke the host method – can choose not to, or join point is method invocation. Methods only – no aspectual invocations on field refs. If: our

Don't match the method they will be wrapping

Invoked implicitly – unlike expected methods

Aspectual methods
the user (ie they are local to the collaboration).
Attaching the aspectual method
• `SetMethod.invoke()` calls `setB` with the intended argument.
  
  return is void, so that is unnecessary.
  
  `Retrofit` and extract any returned object – in this case, the
  needs to be passed to `setB`. In addition, we need to unpack
  `Certificate` object that
  ```
  ● Our generated method also creates the `Certificate` object that
  signature as `setB`, but which calls `setB`.
 Instead, we automatically generate a method with the same
  ● different signatures.
  We want to replace `setB` with the `setB` method, but they have
  ```

Around `setB` do `setB`
public class M
{
    public static void main(String[] args)
    {
        // Since collaborating are Java, just import
        import version3dp.*;
        package test;
    }

    with-back-pointer + sl::getTarget().back();

    System.err.println(sl + "hasTarget" + sl::getTarget() + " with-back-pointer + sl::getTarget().back();

    System.err.println(sz + "hasTarget" + sz::getTarget() + " with-back-pointer + sl::getTarget().back();

    System.err.println(sz + "hasTarget" + sz::getTarget() + " with-back-pointer + sl::getTarget().back();

    System.err.println(t); // invokes not just the setter, but the aspecual method

    t = new Target();
    source sz = new Source;
    source sl = new Source;

    System.err.println(t); // which is why the participants were public

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

```java
{  
    {  
      let _s not provide the setter
      export get as getTarget;
      provide field with back;
    } The + = variantIdp, Target, getTarget, Source
    {  
      export setTarget;
      export getTarget;
      src + = variantIdp, Source, getTarget, Target
      {  
        attach variantIdp, getset, Target
      } public participant Target
    } source {  
      public participant Source
      colab variantIdp, getset, Target
      and setters to it;
    }
    That export of back as a variable is kinda ugly. Let's add setters
    Re-use
    }
```
We've accumulated behavior bit by bit.

vars, vars, vars, vars, vars, vars.

We have built up more and more functional collaborations:

Composition Review: Accumulation
Composition alternative: parallel
collaboration cannot be exported.

Attachment names are only visible within the host.

Each attachment.

We can refer to members from other attachments by naming clauses, so each needs two clauses.

Each collaboration can be added at most once per attach.

Can get messy, keeping track of what goes where.

We add it all at once.

Composition review: Parallel.
Composition alternative: parallel variation
The host collaboration's members are accessed as if they were inserted in a previous attach clause. The name of the

Composition Review: Parallel 1.5

1. The host collaboration is the same as the collaboration.

2. So it's parallel too.
Composition alternative: the good way
to provide it (as only exported members are visible).

- To export the expected field, else we would be unable to maintain them behind the scenes.
- We’ve built up a composite collaboration which only expects a

```javascript
{ }
export getaccessible as getSource;
    } Target += Bar; this
    } provide field with
    } Source += Var, Src
    } attach vars, getTarget
{ } public participant Target
{ } public participant Source
} collab varNgoodway;
```

the good way, cut, and review
to be visible outside the host collaboration.

Every member of an attached collaboration must be exported.

Collaborations are attached to a host by attach clauses.

Collaborations are compiled separately.

Recap
Homework!
Some pages:

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

Aspect J container example
```java
public abstract int check();

public abstract int weight;
;
System.out.println("SimpleObject", +name +"weights" +weight);
}

public int check()
{
    return res;
}

res = int weight;
;
res = newSimple);
}

public static Simple make(String u, int w)

public Participant Simple extends Item
{

    return weight;

    System.out.println("SimpleObject", +name +"weights" +weight);
}

public int check()
{
    return res;
}

res = int weight;
;
res = newSimple);
}

public static Simple make(String u, int w)

public Participant Simple extends Item
{

    return weight;

    System.out.println("SimpleObject", +name +"weights" +weight);
}
```
```java
{{
    contents.add(i);
    {
        contents = new Vector();
    }
}()
}}

public void addItem(Item i)
{
{
    return res;
    res.capacity=c;
    res.name = u;
    Container res = new Container();
}}

public static Container make(String u,int c)
{
    int capacity;
    Vector contents;
}

public Participant Container extends Item
```
```java
public class ContainerContinued {

    public int check() {
        int total = 0;
        Iterator it = contents.iterator();
        System.out.println("Container " + name + " loaded");
        while (it.hasNext()) {
            Item child = it.next();
            total += child.total;
        }
        System.out.println(("Container " + name + " weighs \"total\"") + total);
        return total;
    }
}
```
```java
{  
    ct.check();
    ct.addItem(banana);
    ct.check();
    ct.addItem(orange);
    ct.addItem(pencil);
    ct.addItem(apple);
    ct.addItem(kiwi);

    Simple banana = Simple.make("banana", 1);
    Simple kiwi = Simple.make("kiwi", 1);
    Simple orange = Simple.make("orange", 1);
    Simple pencil = Simple.make("pencil", 1);
    Simple apple = Simple.make("apple", 1);

    Container c3 = Container.make("container-3", 1);
    Container c2 = Container.make("container-2", 1);
    Container c1 = Container.make("container-1", 1);

} } 
static public void main(String[] args) throws Exception

    } }
```
write any constructor. (A known bug.)

Notice how we use factory methods rather than constructors with arguments. This is because our ignores constructors

This is of course just a plain Java implementation.
2. One to cache the calculated weight of containers.

I. One to keep track of which items are contained in which

I've also implemented a few aspectual collaborations:
```java

{ expected Child child;
  partial requirement on what can be wrapped //
} Participant Child.setMethod
{
  {{ parent getParent();
    return parent;
  } parent parent;
} Participant Child
{
  {{
    return cm.invoke();
  } cm Child parent = { this;
    System.err.println("set\"child\" ");
  } } actual RV Child.setMethod(cm)
} Participant Parent
} collab backlink;
```
```java
{ }  

{ }

System.err.println("using-cached-value");
if (cached == null) cached = e.invoke();
}

} (EM e)  

{{
    return
    }
}

RV2 cache(FM e)  

{{
    return
    }
}

RV2 cache(FM e)  

{{
    c = c.get();
    c.clearCache();
}

while (c != null)  

    C = this;
    RV2Cache wrapper = e.invoke();
}

aspects

RV invalidate(FM e)  

{{
    cached = null;
    System.err.println("clear-cache");
}

void clearCache()
    RV2 cache:
}

Participant C extends I  

{{
    get();
}

Participant I extends C  

{ }

collab cache:
```
```c
{1
{6

provide get with getParent:

Item += Child; I

{7

around result:check do cache:
around result:invalidate:
around result:valid addItem (Item child) do childset:

} C Parent

} C
attach backlink, caching

attach it to the non-caching implementation (result)
```
output when run as this example does, and show that it produces the same

manually work implement an attachment (either of my

that

create a composite collaboration of the two aspects, and attach

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

assignment