1 Modeling Aspects with APPCs: an Example Suite

1.1 Composing Aspects

In the example considered so far, the aspects to be composed are independent form each other. The connector CompositionConn1 is equivalent to specifying three connectors. The order of executing the individual aspects being composed is not important. This is different in the following example.

```
appc DataWithCounter {

 abstract participant Counter {
     void reset();
     void inc();
     void dec();
  }

 abstract participant DataStructure { // it doesn't have to be a Stack

     protected Counter counter;

     expected void initCounter();

         // the method for initializing the count field
         // is let to be bound at deployment time
         // when the concrete type of Lock is known

     expected void empty();
     expected void push(Object a);
     expected void pop();

     replace void empty() {
               counter.reset();
               empty();
             }

     replace void push(Object a) {
             counter.inc();
             push(Object a);
            }

     replace void pop() {
               counter.dec();
               pop();
             }

  }

}


APPC LockedData {

   abstract participant Data {

       Lock lock;

       expected void initLock();
```

```
        // the same as for count

      expected * method_to_wrap(*);

      replace method_to_wrap {
        if (lock.is_unlocked()) {
             lock.lock();
             method_to_wrap(*);
             lock.unlock();
         }
    }

   abstract participant Lock {
     expected void lock();
     expected void unlock();
     expected boolean is_unlocked();
   }
}

// appl

class StackImpl {
   String s = new String();
   void empty() {s = "";}
   void push(char a)
     {s = String.valueOf(a).concat(s);}
   void pop()
     {s = s.substring(1); }
   char top() { return (s.charAt(0));}
   ...
 }


class QueueImpl {...}

class CounterImpl {
   int i = 0;
   void resetImpl() {i=0;};
   ...
 }

class LockImpl {
   boolean l = true;
   boolean is_unlocked() {return l;}
   ...
}

// deployment

connector CounterWithLock {
  Counter = ConterImpl implements LockedData.Data {
     method_to_wrap = {reset, inc, dec}};
     void initLock() {
        lock = new Lock();
     }

  Lock = LockImpl implements LockedCounter.Lock;
```

```
}

connector AddLockedCounter {

  Stack = StackImpl implements DataWithCounter.DataStructure {

       void initCounter() {
          counter = new Counter();
          counter.initLock();
        }
       void push(Object obj) {push(((String) obj).charValue());}
       Object pop() {return (Object) String.valueOf(pop());}
       ...
   };

  Queue = QueueImpl implements DataWithCounter.DataStructure {
     ... };

  Counter = CounterWithLock.Counter implements DataWithCounter.Counter;

// using


class Client {

// creating instances of the old classes
DataStructure.StackImpl stack = new DataStructure.StackImpl();
DataStructure.QueueImpl queue = new DataStructure.QueueImpl();


//adapting ..

stack = AddLockedCounter.adapt(stack);
queue = AddLockedCounter.adapt(queue);

//using

stack.push(...);
...
```

An alternative deployment would be to first deploy DataWithCounter with LockedData, i.e., feed the provided Counter part of DataWithCounter into what is expected from the Counter participant in DataWithCounter. This would result in an "composed" but still "abstract" aspect DataWithLockedCounter, which can then later be deployed with concrete applications, as illustrated by the code below.

```
appc DataWithLockedCounter {
// (??? should I call this appc or connector???)

   abstract participant Counter = DataWithCounter.Counter implements
                                          DataWithLock.Data {
         method-to-wrap = {reset, inc, dec}};

   abstract participant Lock = DataWithLock.Lock;

   abstract participant DataStructure = DataWithCounter.DataStructure;
```

}

It might look a bit strange that the expected interface of DataWithCounter.Counter is specified to implement the expected interface of DataWithLock.Data. In fact there is nothing strange: what the connetions above say, is that whatever implementations will feed reset, inc, dec in the xepected interface of DataWithCounter.Counter these will first be provided to the expected interface of LockedData. The provided interface of the latter at Data will be the new Counter participant in DataWithLockedCounter. The DataStructure participant is simply taken over from DataWithCounter. Now, the deployment of the composed aspect DataWithLockedCounter with the application package will be as follows.

```
connector ApplicatinDataWithLockedCounter {

  Counter = CounterImpl implements DataWithLockedCounter.Counter {
          void initLock() {lock = new Lock();}
          void reset() {resetImpl();}
          ... }

  Lock = LockImpl implements LockedCounter.Lock;

  Stack = StackImpl implements DataWithCounter.DataStructure {

       void initCounter() {
          counter = new Counter();
          counter.initLock();
        }
       void push(Object obj) {push((char) Object);}
       Object pop() {return (Object) pop();}
       ...
   };

  Queue = QueueImpl implements DataWithCounter.DataStructure {

     ... };
```