ServiceManagerRelease

Few things generate so much controversy as the release() method of the ServiceManager. Let's take a look at a few of the common comments made (we see these periodically):

try{}finally{release} is a PITA

Yes it is. It is the price you pay for having a rich set of component lifestyles available to you automagically - we need to deal with all components as if they have a rich lifestyle, since the lifestyle is not exposed to the client (which is a Good Thing).

Note there's several things you can do to make this stuff easier. Consider adopting a code convention where you have private members with a ms_ (or something, for "My Service") prefix, then an abstract base class which implements dispose() and will release all those private members, then call super. dispose() as the last method inside your child class dispose() (if any):

```
 public abstract class AbstractServiceDisposer    // UNTESTED CODE SKETCH
     extends AbstractLogEnabled
     implements Disposable, Serviceable
{
   private ServiceManager m_sm;

   public void service( ServiceManager sm )
   {
     m_sm = sm;
   }

   protected ServiceManager getServiceManager()
   {
     return m_sm;
   }

   public void dispose()
   {
     try
     {
       Field[] fields = this.getClass().getDeclaredFields();
       for( int i = 0; i < fields.length; i++ )
       {
         try
         {
           field = fields[i];
           if( field.getName().startsWith("ms_") )
             m_sm.release( field.get(this) );
         }
         catch( IllegalArgumentException iae )
         {
           // won't happen
         }
         catch( IllegalAccessException iace )
         {
           getLogger().warn( "Can't release all my services: " +
             "SecurityManager won't allow me to reflect on myself",
             iace );
         }
       }
     }
     catch( SecurityException se )
     {
        // we need permissions from security manager
        getLogger().warn( "Can't release all my services: " +
          "SecurityManager won't allow me to reflect on myself",
          se );
     }
}
```

alternative approaches include mixins/AOP/etc. Before you do any of this, words of caution:

http://lsd.student.utwente.nl/jicarilla/TooMuchMagic

GC magic with finalize() can work

GC magic is not part of the avalon-framework contract. Why? It can be somewhat buggy on obscure JDKs, it's expensive, it's complex, its misusing intended use of finalize().

To be on the safe and portable side, don't rely on it being available, nor on it to always work perfectly.

Developers are dumb/lazy

What you can do is create a framework on top of avalon that makes more assumptions about developer laziness that uses mechanisms such as the above. You'll want to coerce your dumb developer friends to all use your special class though.

No-one uses it

Errr...wrong. ServiceManager.release() is used in applications all over the globe. A notable one is cocoon.

what about Releaseable, ReleaseRequired, ReleaseUtil?

Been there, done that, won't work. Marking components as releasable will just mean you'll be checking whether they're marked as such, which is just as ugly.

what if we add a method to the service interface?

A SAXTransformer can't be used after its endDocument() method has been called, so let's add one method to all services that you can call, and which indicates that the component can be reclaimed / put in a pool / disposed.

Comment 1: Adding a release() method to the Component interface would be a double whammy - not only do we bring back the most hated interface in Avalon from its deprecation, we bring it back in tandem with the most hated method. Nice.

Comment 2: The code doesn't get that much better. Instead of:

```
 MyService ms = null;
try {
    ms = lookup(...);
    ...
} finally {
    release(ms);
}
```

you get:

```
 MyService ms = null;
try {
    ms = lookup(...);
    ...
} finally {
    if (ms != null) {
        ms.release();
    }
}
```

which is a step sorta sideways-backward, with an extra null check being required. (Having that null check encapsulated in a ReleaseUtil.release(Object o) is **not** a solution, see above.)

It boils down to...

There's a simple choice:

either

- you restrict yourself to a subset of lifestyles that do not require release() semantics (example: Phoenix, PicoContainer);

or

- you always do the release() semantics, even when they're not really required