

***“Don't call us, we'll call you”***

P. Calafiura

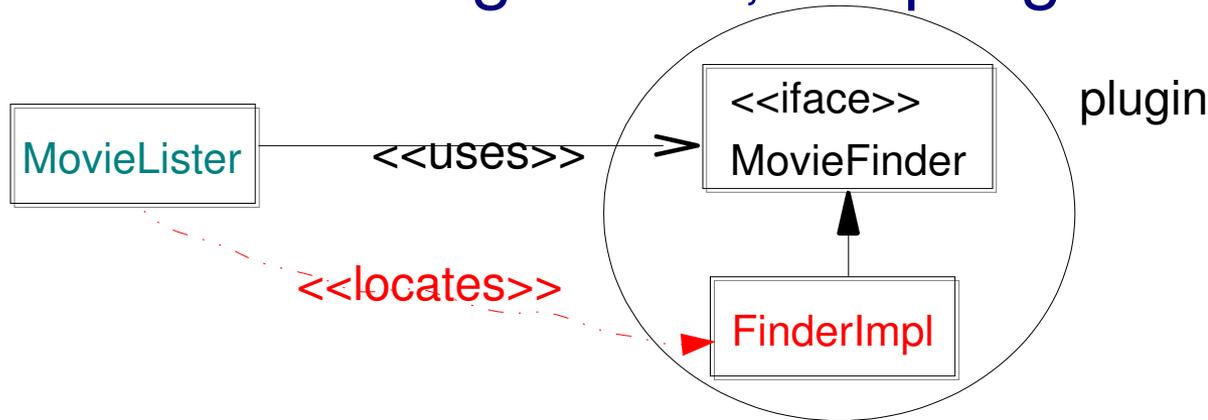
HPCN group Jul 06

# Disentangling the Framework

- Heard me talk twice about disentangling cross-cutting concerns using AOP
  - logging, persistence, thread-safety
- Today focus on a sneakier entanglement issue with component frameworks:  
**application assembly**
  - separate component configuration/use
  - reduce component coupling to framework

# The Problem

- Good component architectures use plugins
  - run time configuration, coupling via iface

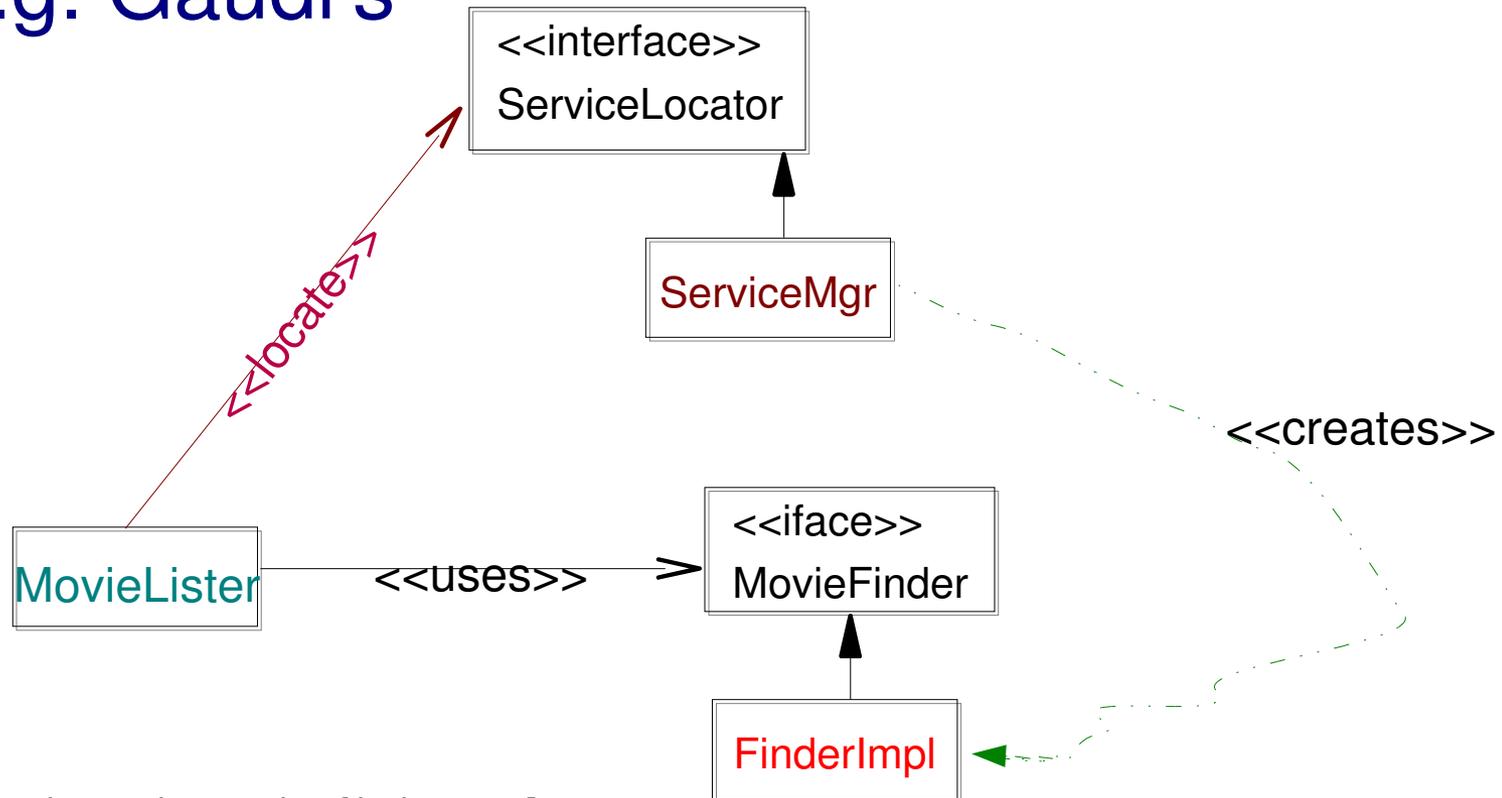


- **Issue:** how does `MovieLISTER` get hold of a `MovieFinder` implementation? Can't do

```
public MovieLISTER() {  
    finder = new ColonDelimitedMovieFinder("movies1.txt");  
}
```

# Service Locator

- e.g. Gaudi's



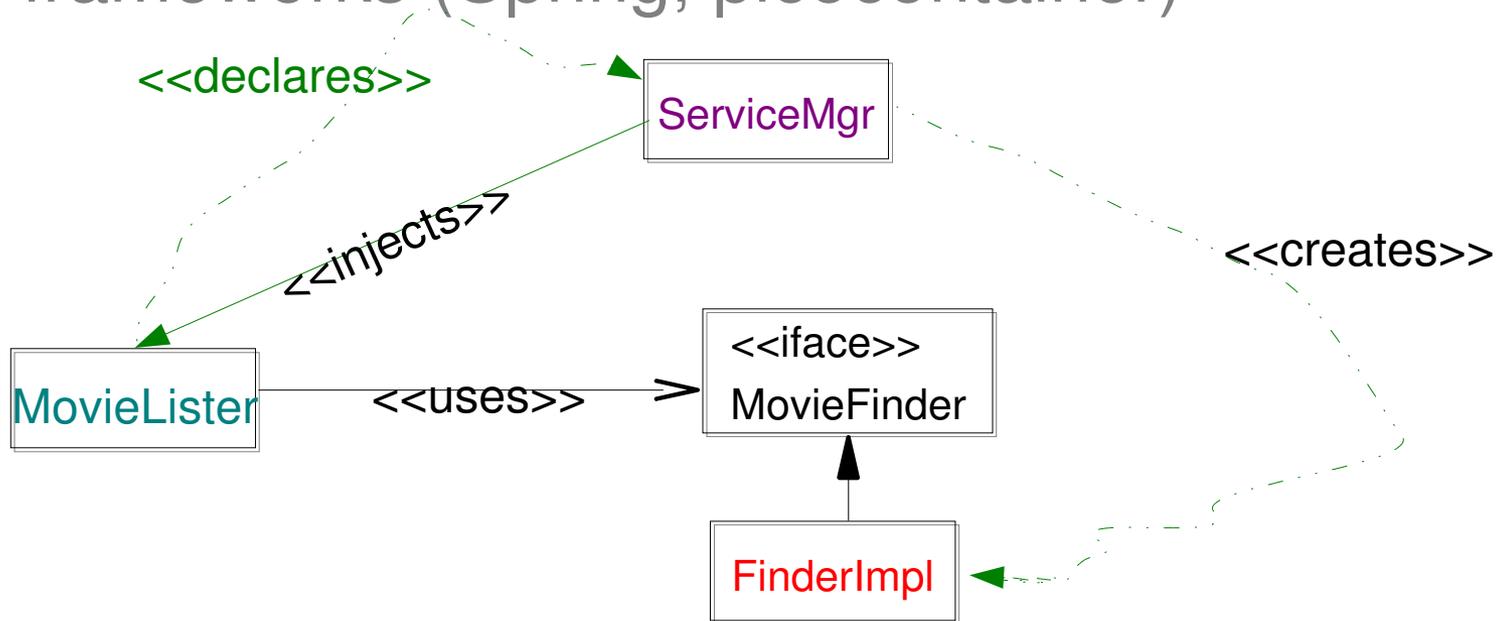
```
// Retrieve the movie finder Tool
s = toolSvc()->retrieveTool(m_movieToolType, m_movieToolInstanceName, m_movieTool);
if (s.isFailure()) {
    log << MSG::ERROR << "Could not retrieve " << m_movieToolName << " from ToolSvc. ";
    ....
}
```

# Service Locator Discussion

- Service Locator leaves responsibility of assembling the application (creating components, connecting them) to component code
- Framework does not know what the component needs before it can run
- Dynamic, “jit” application assembly
  - components created if/when needed

# Dependency Injection

- aka “don't call us, we'll call you”
  - pioneered by java “lightweight containers” frameworks (Spring, picocontainer)



# Injection Techniques

## Constructor Injection (picocontainer)

- component created in usable state
- readability issues if many interdependencies
- inheritance can get in the way

## Setter Injection (Spring)

- easier to read, more flexible, incomplete constructor

## Interface Injection (Avalon)

- fwk uses declared injection interfaces to figure out the dependencies and to inject the correct dependents
  - remember D0 framework?

# Declaring Interdependencies

Component must declare what to inject

Code or configuration files?

- API is needed for many non-trivial cases
  - “I’ll need this service but only for simulation jobs”

```
private MutablePicoContainer configureContainer() {
    MutablePicoContainer pico = new DefaultPicoContainer();
    Parameter[] finderParams = {new ConstantParameter("movies1.txt")};
    pico.registerComponentImplementation(MovieFinder.class,
                                        ColonMovieFinder.class, finderParams);
    pico.registerComponentImplementation(MovieLister.class);
    return pico;
}
```

# Declaring Interdependencies II

Component must declare what to inject

Code or configuration files?

- Configuration file (XML of course...) promotes a clearer separation among configuration and use

```
<beans>
  <bean id="MovieLister" class="spring.MovieLister">
    <property name="finder">
      <ref local="MovieFinder"/>
    </property>
  </bean>
  <bean id="MovieFinder" class="spring.ColonMovieFinder">
    <property name="filename">
      <value>movies1.txt</value>
    </property>
  </bean>
</beans>
```

# The POJO Movement

## Reaction to 1<sup>st</sup> gen java frameworks: EJB

- work with POJOs (Plain Old Java Obj)
  - described to framework by configuration files/API
- one example Hibernate (presented here)
- influenced by AOP and generic programming
- Application assembly only one example of cross-cutting concern
  - transaction management, persistence, thread-safety

# What's in it for us dinosaurs?

## ATLAS discussing new techniques to locate components

– currently:

```
// Retrieve the movie finder Tool
s = toolSvc()->retrieveTool(m_movieToolType, m_movieToolInstanceName,
                           m_movieTool);
```

tool locator

String Properties

local pointer

- client must know how and when to call the locator, must tell athena when done with tool (**nobody does**)
- athena does not see the string properties and the local tool pointer (**hence can't release tool**)





# Conclusions

## Many ways to skin a cat

- Common goal
  - separate configuration of services from their use
- ServiceLocator vs Injector
  - flexibility vs transparency
- Configuration File vs Injection API
  - for athena, python properties are bit of both
- I like Injection because allows framework to see the information flow into components

# References

- Martin Fowler Injection paper
  - <http://www.martinfowler.com/articles/injection.html>
- ACM Queue issue on components
  - C. Richardson “Untangling Enterprise Java”  
ACM Queue, Vol 4 No 5 pp 36-44