AnnotationSupport

Description

AchimHuegen, May 16 2006

Proposed Solution

A module is a plain java class that annotates its methods. Each annotated method corresponds with a service point, configuration point or contribution.

The first example demonstrates the definition of three services and their wiring.

```
public class ExampleModule1
{
  @Service(id = "Calculator", serviceModel="singleton")
  public Calculator getCalculator()
  {
    return new CalculatorImpl(service(Adder.class), service(Subtracter.class));
  }

  @Service(id = "Subtracter", serviceModel="singleton")
  private Subtracter getSubtracter()
  {
    return new SubtracterImpl();
  }

  @Service(id = "Adder", serviceModel="singleton")
  private Adder getAdder()
  {
    return new AdderImpl();
  }
}
```

The method body of a service point method is at the same time factory method, responsible for the service construction and its wiring. Note, that the service-Method offers a typed access to other services!

Autowiring Properties:

```
public class ExampleModule
{
  public Calculator getCalculator()
  {
    CalculatorImpl calculator = new CalculatorImpl();
    autowireProperties(calculator);
    return calculator;
  }
}
```

Autowiring Construction:

```
public class ExampleModule
{
  public Calculator getCalculator()
  {
    Calculator calculator = autowireConstruct(CalculatorImpl.class);
    return calculator;
  }
}
```

Define configuration points with default values:

```
public class ExampleModule
{

  @Configuration(id = "Translators")
  public List getTranslators()
  {
    List translators = new ArrayList();
    translators.add(new StringTranslator());
    translators.add(new IntegerTranslator());
    return translators;
  }
}
```

Contribute to a configuration point:

```
public class ExampleModule
{

  @Contribution(configuration-id = "Translators")
  public void getTranslators(List translators)
  {
    translators.add(new SmartTranslator());
  }
}
```

Wiring of configurations:

```
public class ExampleModule
{
  public TranslatorManager getTranslatorManager()
  {
    TranslatorManagerImpl manager = new TranslatorManagerImpl();
    manager.setTranslators(configuration("translators", List.class);
    return manager;
  }
}
```

Configurations can be any POJO:

```
public class ExampleModule
{
  @Configuration(id = "strutsModule")
  public ModuleConfig getStrutsModule()
  {
    return new ModuleConfigImpl();
  }

  @Contribution(configuration-id = "strutsModule")
  public void contributeToStrutsModule(ModuleConfig config)
  {
    config.addActionConfig(new ActionConfig());
    config.addFormBeanConfig(new MyFormBeanConfig());
  }
}
```

Add an interceptor:
... to be specified

Details

The methods *service*, *configuration*, *autowireConstruction*, *addInterceptor* etc. could be inherited from an ancestor. They make use of java 5 generics get rid of casting.

Convention over configuration

Most annotations could be replaced by conventions in the default case:
For example the method name *getServiceCalculator* would be sufficient to define a new service point with the name *calculator*. *contributeToStrutsModule* could indicate a contribution to the configuration *strutsModule* which is defined by *getConfigurationStrutsModule*.