DescriptorApiRevampProposal

This proposal is far from complete, but I was hoping to get a discussion going before I put too much effort into a proof of concept (or proof of failure). The input should help me complete this proposal.

Problem Description

It is currently (HiveMind 1.1 beta 1) quite difficult and cumbersome to provide module descriptors in formats other than XML, e.g. plain Java code. There is of course the Groovy support, but it has a one-to-one correspondance with the XML format, as it is implemented as a thin wrapper (a *Builder* in Groovy lingua) on top of the XML processing.

The biggest problem is the many constructs in the current API which are specific to the XML syntax. Thus providing a different module descriptor format requires the processing implementation to be expressed in terms of these XML specific constructs. This task is both overly complex and potentially forfeits implementing the processing using constructs better suited to the job.

It is for instance currently not easy to express a module descriptor using plain Java code.

Proposed Solution

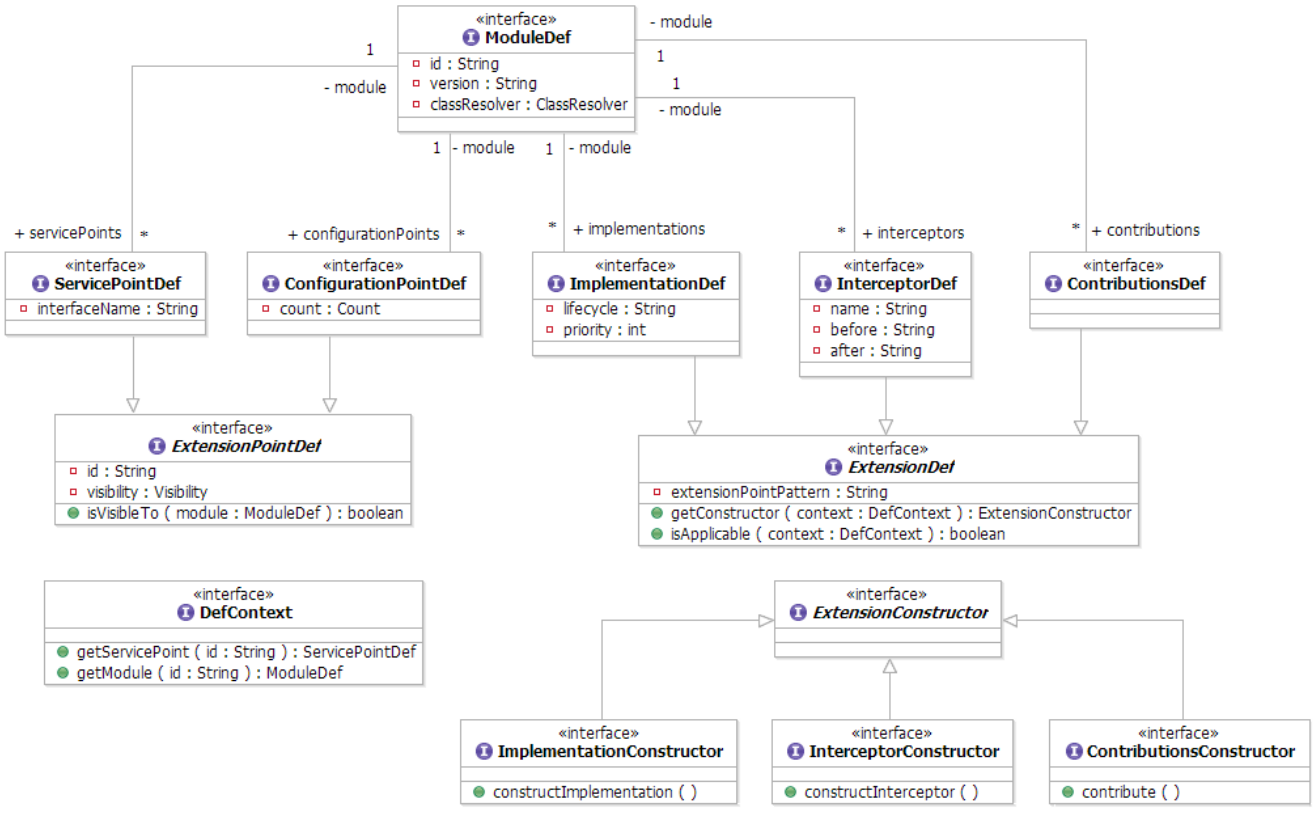
The approach of the proposed solution is to move to a more abstract **Descriptor API** and factor the XML specifics into an XML implementation (based on this API). IMO the following features are XML specific:

- shorthand notations (e.g. <invoke-factory> allowed inside <service-point>) reflected in the current API
- the *schema processing* converting XML elements to POJOs (this also includes ${} symbol resolution)
- sub-modules – these are really just normal modules
- conditional contribution expressions
- service implementation and interceptor factories

Factoring these out will in turn also affect parts of the **Runtime API** and implementation.

Descriptor API

Here is an overview of the proposed Descriptor API (package `org.apache.hivemind.def`).



And the corresponding interfaces in Java:

```
public interface ModuleDef
{
    public String getId();
    public String getVersion();
    public ClassResolver getClassResolver();

    public List getServicePoints();
    public List getConfigurationPoints();
    public List getImplementations();
    public List getInterceptors();
    public List getContributions();
}
```

```
public interface ExtensionPointDef
{
    public ModuleDef getModule();
    public String getId();
    public Visibility getVisibility();
    public boolean isVisibleTo(ModuleDef module);
}
```

```
public interface ExtensionDef
{
    public ModuleDef getModule();
    public String getExtensionPointMatchPattern();
    public boolean isApplicable(DefContext context);
}
```

```
public interface ServicePointDef extends ExtensionPointDef
{
    public String getInterfaceName();
}
```

```
public interface ConfigurationPointDef extends ExtensionPointDef
{
    public Occurances getCount();
}
```

```
public interface ImplementationDef extends ExtensionDef
{
    public String getServiceModel();
    public int getPriority();
    public ImplementationConstructor getConstructor(DefContext context);
}
```

```
public interface InterceptorDef extends ExtensionDef
{
    public String getName();
    public InterceptorConstructor getConstructor(DefContext context);
}
```

```
public interface ContributionDef extends ExtensionDef
{
    public ContributionConstructor getConstructor(DefContext context);
}
```

```
public interface DefContext
{
    public ModuleDef getModule(String id);
    public ServicePointDef getServicePoint(String id);
}
```

Some important notes:

- The *RegistryBuilder* part of the Descriptor API has been left out, as it should remain the same.
- Ordering of interceptors could also be addressed by letting an implementing object implement the HiveMind `Orderable` interface.
- The `getConstructor()` methods in *ImplementationDef*, *InterceptorDef*, and *ContributionDef* establish the link to the Runtime API. The returned objects will be used at runtime to lazily instantiate the actual objects. The referenced interfaces are described in the following.

Runtime API

As mentioned the `getConstructor()` methods on the *Extension* subtypes establish the link to the Runtime API. Here are the referenced interfaces thereof. Note that the types of the method arguments in these interfaces are the internal interfaces and **not** the just described Descriptor API interfaces.

```
public interface ImplementationConstructor
{
    public Object constructCoreServiceImplementation(ServicePoint servicePoint,
            Module contributingModule);
}
```

```
public interface InterceptorConstructor
{
    public Object constructServiceInterceptor(InterceptorStack interceptorStack,
            Module contributingModule);
}
```

```
public interface ContributionConstructor
{
    public Object contributeOrderedElements(OrderedConfiguration configuration,
            Module contributingModule);

    public Object contributeMappedElements(MappedConfiguration configuration,
            Module contributingModule);
}
```

Some notes:

- The interfaces *ImplementationConstructor* and *InterceptorConstructor* already exist (although with different names). The difference is that here the *context* is provided as explicit method arguments.
- The interface *ContributionConstructor* does not have correspondance in the current code.

Discussion