PojoServices

Problem Description

AchimHuegen, July 6 2004, 1.0beta

Services must be interfaces. Plain java objects are not supported.

- Maintenance of interfaces is extra work (two units, linked javadoc)
- Prototyping is more difficult
- Adoption of HiveMind for new developers is more difficult

Proposed Solution

Allow the use of java classes in the interface attribute of a service-point.

Example:

```
<service-point id="POJO" interface="java.text.SimpleDateFormat">
            <invoke-factory service-id="hivemind.BuilderFactory" model="primitive">
                    <construct class="java.text.SimpleDateFormat">
                            <string>MM/yy</string>
                    </construct>
            </invoke-factory>
</service-point>
```

In fact it is very easy to implement, just remove the interface check from ServicePointImpl#lookupServiceInterface and hivemind is ready for POJOs. Better: Let the serviceModel decide whether it want to support non-interface classes.

Interceptor support should be extended to support both POJOs (without final methods) and interfaces. This can be achieved by switching from inheritance to composition. For example a LoggingInterceptor shouldn't specialize AbstractLoggingInterceptor but hold a reference to a LoggingInterceptor (non abstract counterpart of AbstractLoggingInterceptor) and delegate all the work to this instance.

Discussion

HowardLewisShip: **-1** HiveMind already has a facility for creating POJOs (BeanFactory). However, HiveMind is also supposed to be about *best practices* including coding to interfaces. I'm not interested in *nice-to-haves* I'm interested in **specifically useful** constructs.

AchimHuegen: The POJOs that a BeanFactory produces are no singletons (ignoring the caching) and can not be used for configuring another service. The POJOs itself can not be configured. Some of these problems could be solved by your "something nifty with translators" improvements in combination with PrototypeBeanFactory. So I would appreciate a +1 for PrototypeBeanFactory.

If you don't want to include POJO services in the distribution, what about leaving the decision to support POJOs to the ServiceModel? The HiveMind user could still use POJOs if he really want to ignore the best practises by implementing a new service model.

KnutWannheden: I don't want to start a religious war here, but implementing this request is IMHO important enough to take the risk 😉

It's the best practices argument which got me thinking. Being an XP and TDD enthusiast I quite like the DoTheSimplestThingThatCouldPossiblyWork practice. Quote: *... pick something you can do and do quickly, so that you can get on with the other things you really need to do. Then do that thing professionally and well, complete with all appropriate refactoring.* In context I think defining a POJO as a service is the simplest thing that could possibly work. The exact service interface may not crystallize until later on in development. Refactoring tools are very important when employing this practice. Imagine a HiveMind refactoring in Eclipse linked to the Extract Interface refactoring: It would both extract an interface from the POJO class and update the `service-point` definition in the descriptor. Or how about a warning marker next to a POJO service in the descriptor?

In summary I think coding to interfaces is most definitely a best practice, but as an *enforced practice* I think it is impeding the getting-there. It's a matter of EnablingAttitude.