NotXMLProposal

Problem Description

HowardLewisShip, April 26 2004, 1-0-alpha-4

One of the tenets of HiveMind is that deployment descriptors for J2EE are in verbose XML, and that this is a bad thing.

And yet, the descriptors for HiveMind are themselves XML and can be quite verbose.

Several people, especially ErikHatcher, have **repeatedly** pointed this out to me.

As things currently stand, XML is used for several basic reasons:

- It's easy to use an XML parser to read the kind of hierarchical data inside a descriptor.
- It's easy to generate HiveDoc directly from the XML.
- It is easier to use existing SAX parsers (now part of the JDK) than to write our own parser.

Proposed Solution

We can define our own special purpose langage for the descriptors. The language could be more readable and succinct than the equivalent XML. More work would have to be put into the HiveDoc, to generate the HiveDoc directly from the descriptor objects, or to generate an intermediate XML format from the descriptor objects.

I've been working on SimpleDataLanguage (SDL), which is isomorphic to a well-defined subset of XML. Rather than get into the messy details, here's an example of a HiveMind module deployment descriptor expressed as SDL:

```
module (id=myapp.ui.toolbar version="1.0.1")
{
  configuration-point(id=ToolbarActions)
  {
    schema
    {
      element (name=item)
      {
        attribute (name=label required=true)
        attribute (name=icon required=true)
        attribute (name=action-class)
        attribute (name=service-id)

        conversion (class=myapp.ui.toolbar.ToolbarAction)
        {
          map (attribute=action-class property=action translator=object)
          map (attribute=service-id property=action translator=service)
        }
      }
    }
  }

 contribution (configuration-id=ToolbarAction)
 {
  item (label=Open icon=Open.gif service-id=OpenService)
 }

 service-point (id=OpenService interface=myapp.ui.toolbar.ToolbarAction)
 {
   invoke-factory(service-id=hivemind.BuilderFactory)
   {
     construct(class=myapp.ui.toolbar.impl.OpenActionImpl log-property=log messages-property=messages)
     {
      set(name=frob value=grognard)
      set-service(name=dialogViewer service-id=myapp.ui.DialogViewer)
      set-configuration(name=pipeline configuration-id=myapp.ui.FileOpenPipeline)
     }
   }

  add-interceptor(service-id=hivemind.LoggingFactory)
 }
}
```

Remember, XML started as a document format, where using punctuation for blocking delimiters was a bad idea, since such characters could show up in the document text. Our use of HiveMind deployment descriptors is a much more constrained environment, more like scripting than like pure documents. For example, quoting of attributes is not necessary for most attributes, just the ones with whitespace (or other characters outside a specific subset) in the value.

Also, many parts of XML were intended to make it easier to write an XML parser (though, over time, things have twisted such that a fully conformant XML parser is now an enourmous beast). Again ... not our concern.

This solution involves nailing down precisely (in BNF) what our format will look like and writing that parser. I have been experimenting with JavaCC, and have put together an SDL parser that emits SAX parse events. With very minor changes to the DescriptorParser class we could convert our XML descriptors to SDL. We could even support both formats!

Discussion

JoeDuffy, April 26 2004

What about using something like Groovy to configure and wire everything up? This has the obvious downside that you're working at an imperetive vs. declarative level, probably interacting with the HiveMind object model in some fashion, but has the benefit of being a familiar syntax, very lightweight, and is reusing an existing technology (rather than inventing a proprietary grammar).

HowardLewisShip: Groovy may not be fully stable for months and represents a significant new dependency at a time when I'd like to remove dependencies. I'm also not sure how we could get the desired level of (line precise) exception reporting. However, I'd like folks to noodle some more on the ideas of using scripting languages to perform some of the work (especially with respect to the conversion of XML objects). Maybe that is the right approach to removing XML from HiveMind ... that's thiking outside the box.

In the meantime, I've updated the example above to reflect a slightly more verbose but significantly more consistent approach (based on additional thoughts I had composing this blog entry). Keeping the design isomorphic to a subset of XML had advantages ... I can just make my parser spew out SAX events, maybe even feed it into a SDL (Simple Data Language) to XML to HTML pipeline.

HarishKrishnaswamy: I am completely against the idea of a proprietary language. I think it will certainly hinder adoption. I would like to see a Java scripting language being used for this purpose. I agree Groovy is certainly not stable but Beanshell is. I have already wipped up some configuration for work using Beanshell with line precise error reporting. And all that is needed is a simple helper class and a very small Beanshell script to execute the configuration one node at a time. The script will look something like this:

```
evalModule(url, helper, interpreter, callstack)
{
    setAccessibility(true);

    reader = null;

    try
    {
        reader = new InputStreamReader(url.openStream());
        parser = new Parser(reader);
        parser.setRetainComments(true);

        while (!parser.Line()/* eof */)
        {
            node = parser.popNode();

            node.setSourceFile(url.toString());

            // Cache the line number for error reporting purposes
            helper.setCurrentLineNumber(node.getLineNumber());

            node.eval(callstack, interpreter);
        }
    }
    finally
    {
        if (reader != null)
            reader.close();
    }
}
```

And the configuration file could look like this:

```
servicePoint("service-id", ServiceInterface.class);

// Singleton service constructed via constructor injection
singleton(implementation("service-id", ServiceImplementation.class, new Object[]{1.34, "some string", service
("some-service-id")}));

// Pooled service constructed via setter injection
dependencies = new HashMap();
dependencies.put("property1", property1Value);
dependencies.put("anotherService", service("anotherServiceId"));
pooled(implementation("service-id", ServiceImplementation.class, dependencies));
```

And something along the same lines can be used for configuration too.

servicePoint, service, implementation, singleton, pooled ... are all methods in our helper class that will do the needful.

Although I haven't yet tried out to build a doc from the config file, I am pretty sure its very simple. Beanshell already has a Bshdoc.bsh script that generates javadoc like doc.

ChristianEssl: I like Harish's idea of using Beanshell. It is indeed less verbose than xml. Maybe somehow it could be possible to replace the dependencies map and constructor-array with a closure, which directly constructs the implementation. To me this would be the main advantage of using an alternative config format. Apart of this can someone point out what the disadvantages of xml are except of the verbosity and that ejb (and most other frameworks) use it too. Especially for contributions I think xml is much better than scripting, because you basicly define your own specialized easy-to-use language. Is there some thought of mixing scripting with xml?

HowardLewisShip: You might notice I'm unconvinced about using scripting. The core issue is the XML. The way Sun uses XML (lots of elements, no attributes) is super-duper verbose. The XML formats used by HiveMind deployment descriptors are more straight-forward, but are still hard on the eyes ... the extra verbosity inherent in all those open and close brackets, the unnecessary quotes, the long verbose close tags — those all become distractions to the eye, obscuring the real intent. The SimpleDataLanguage I've been proposing here is (especially for Java coders) more natural, because of the use of curly braces to denote enclosure of elements. And yet, your could easily write a translator that converts back and forth between a subset of XML and SDL.

The examples above, using BeanShell, were, in my opinion, even more obscure than the XML versions. In addition, the code seems to expect each line to be a fully executable statement, which is going to make anything reasonable in terms of configuration even more awkward.

HarishKrishnaswamy: Christian, I am not sure how you can construct the service from with the configuration file using simple closures. Could you elaborate?

Howard, Beanshell is simply plain Java with some syntax sugar, so it is not necessary that every line should be an executable statement. I think the use of a scripting language is two-fold - make the config file smaller and simpler, and also simplify the framework internals. The SDL, although looks better than XML, still requires a lot of typing.

HowardLewisShip: I think we can optimize the format somewhat; the SDL version is isomorphic to the XML version, but even the XML version could be made more succinct. We seem to be in a situation of *declarative* (SDL/XML) vs. *procedural* (scripting). To me, scripting opens up a whole can of worms ... and sacrifices one of the **key** features of HiveMind: line precise error reporting. Groovy builders are a step up from pure BeanShell statements but still miss the mark. I think HiveDoc will also suffer. I have nothing against something like:

```
. . .  invoke-factory (service-id=foo.bar.BeanShellServiceFactory)
{
  script {
<<

return new MyServiceImpl(. . .);

>> }
} . . .
```

The << . . . >> syntax fills the same role as CDATA sections in XML ... everyting inside the delimiters is passed through as-is. I like the idea of scripting, but I like line precise error reporting much, much more. I also think scripting introduces further problems if and when there is tool support (the tool has to parse the script, which is not nearly so regular as the SDL or XML).

HarishKrishnaswamy: Why do you think line precise error reporting has to be sacrificed with a scripting language? why something like the script above would n't work? With a scripting language the parser is already built, the tool will simply have to work with the AST (much simpler than working with the raw grammer). Yes, I could use a BeanShellServiceFactory, but I am still not seeing the benefit of SDL.

HowardLewisShip: I must be, from ignorance, misunderstanding the

```
while (!parser.Line()) { node = parser.popNode() ... }
```

stuff. I was looking at that as "read a line, execute a line", but that's obviously not what it is doing. I'd be interested in seeing a real, formatted example, and more detailed notes about how line-precise error reporting is maintained. I'm not completely stubborn, but if you want to change my mind (especially considering who ends up doing the most work) you have to work *really hard* ... but when this has happened in the past, I will fully embrace the final solution (witness: implicit components, etc., in Tapestry 3.0 — which started with a flame by Marc Fluery).

IliaHonsali: I suggest to keep using xml for conf files, the main purpose of xml is to store data not to be easy to view, so, once hivemind come stable "someone" will make gui to mask the xml (with eclipse it'is easy), xml is also something usual that not scare newbies like me.

HowardLewisShip: My current leaning is to allow multiple formats; the existing XML support, potentially the SDL format and the YAML format. However, the idea of linking up a scripting language is intriguing ... I just want the contingent that is pushing the idea to provide a full and viable solution. Meanwhile, out-of-band, EricHatcher pointed to an developerWorks article that really sums up the situation while raising (for me, at least) the question: who is this XML for: HiveMind, or us?

KnutWannheden: To me the crucial question is whether the module (or deployment) descriptor should be *descriptive* or *imperative* (as seen in the make vs. Ant war). For HiveMind I tend to favor a descriptive form. But it should be possible to allow both. We'll just have to decide which one goes into the framework and which one into the library 🙂

GeoffLongman: Speaking as a Hivemind user, XML and SDL look fine. I have not warmed up to YAML at all. Using a true scripting language like Groovy turns me off completely. Leave the descriptors as *descriptors* that describe or declare something and are not executable. Speaking as somebody who *migh t someday* commit to building an Eclipse plugin for Hivemind, XML is the path of least resistance for me. A declarative markup like SDL or YAML would be ok too if the parser support were at a level conducive to developers tools. An DOM equivalent (AST tree) with line *and* character offset *and* range precise information is needed. FYI, no open source XML parsers were this precise out of the box for Spindle.

ChristianEssl: XML is for me. If I look at the tremendous work Howard has put into the XMLParser I'd say a JavaCC grammar would be even easier for HiveMind - less validation. XML is for me because I somehow got to know what an element and an attribute is, how to start and end the document, how to escape things, how to add comments, the meaning of whitespace, what are valid names, knowing which block-close belongs to which start without a lot of counting and finally knowing that others know that (and certainly much more) too. Apart of this looking at the example Harish gave I think he actually meant an language whith only expression-instructions. Well that's not everyones taste, but I'd call it declarative and line pricese reporting can be maintained this way. It has the advantage that it's relatively easy to use combined with JavaDoc. Further it would be very easy to implement convinience methods.

HowardLewisShip: JavaCC generates a token stream and each token knows it start and end line number and column. I think it will be much easier to support this than with XML. I suspect we'll be able to easily get that information out of the parser and into the plugin. Like Tapestry, a plugin shouldn't be all that necessary ... the SDL stuff takes the teeth out of XML, making it look quite pleasant.

HarishKrishnaswamy: Ok, here's the same example with little more details.

```
/**
 * Service point definition
 */
servicePoint("service-id", ServiceInterface.class);

/**
 * Singleton service constructed via constructor injection
 */
singleton
(
    implementation
    (
        "service-id",
        ServiceImplementation.class,
        new Object[] {1.34, "some string", service("some-service-id")} // constructor arguments
    )
);

/**
 * Pooled service constructed via setter injection
 */
dependencies = new HashMap();
dependencies.put("property1", property1Value);
dependencies.put("anotherService", service("anotherServiceId"));

pooled
(
    implementation
    (
        "service-id",
        ServiceImplementation.class,
        dependencies // Setter properties
    )
);
```

This file will be read and built by a builder class that will look something like this:

```
public class BshBuilder
{
    ...
    public void buildModule(URL moduleUrl, Registry registry)
    {
        try
        {
            Interpreter interpreter = new Interpreter();

            helper = new BshBuilderHelper(registry);
            helper.setCurrentModuleName(moduleUrl.toString());

            interpreter.set("$helper$", helper);

            interpreter.eval("importObject($helper$)");  // This is a mixin command

            interpreter.eval("importCommands(\"path/to/evalModule.bsh\")");

            interpreter.set("$interpreter$", interpreter);
            interpreter.set("$callstack$", new CallStack(interpreter.getNameSpace()));

            interpreter.set("$url$", moduleUrl);
            interpreter.eval("evalModule($url$, $helper$, $interpreter$, $callstack$)");
        }
        catch (Exception e)
        {
            handleInterpreterException(e);
        }
    }
    ...
}
```

evalModule.bsh is the same script that I had posted previously. The BshBuilderHelper will look something like this:

```
public class BshBuilderHelper
{
    ...
    String _currentModuleName;
    int    _currentLineNumber;

    public void servicePoint(String serviceId, Class serviceInterface)
    {
        Location location = new Location(_currentModuleName, _currentLineNumber);
        // Register the service point along with the location
    }

    public Object implementation(String serviceId, Class serviceImplementation, Object[] constructorArgs)
    {
        Location location = new Location(_currentModuleName, _currentLineNumber);
        // Register the service implementation along with the location
        return service;
    }

    public Object service(String serviceId)
    {
        // Get service
        return service;
    }

    public Object singleton(Object service)
    {
        // Make service a singleton
        return service;
    }
    ...
}
```

BshBuilder receives the module descriptor sets up the BeanShell interpreter by mixing in the BshBuilderHelper object and executes the evalModule script. The evalModule script reads the module descriptor one statement at a time, sets the line number of the executing node in the helper, and executes it. The helper does the needful. And that's pretty much all for handling the descriptors.

Of late, I have really subscribed into KISS and Enabling Attitude principles and these ideas are simply a repercussion of that.

KnutWannheden: Harish, your example is IMO getting very close to a purely descriptive form (like the SDL or XML approach). Of course someone could (ab)use the BSH design to write an absurdly cryptic module descriptor. This is where I gather you say the Enabling Attitude principle comes in to play. IMO the only thing with the XML descriptor which doesn't conform to this principle is the XML syntax itself, and that's what SDL should solve. But then again with the plethora of XML processing / spewing tools I think XML also has some nice advantages. Also I think one of the main purposes of using a descriptive syntax is to make the descriptor itself readable. And, as previously noted, if users put arbitrary Java code into the descriptor I think it could prove difficult for HiveDoc to produce something useful.

HarishKrishnaswamy: Couple things I forgot to mention: Producing HiveDoc is just as simple because the comments could be a part of the AST and we could annotate them for HiveDoc. And secondly, with this approach I don't think we need a special tool like Spindle for the descriptors; the eclipse scrapbook page is good enough, IMO.

I certainly like the SDL far better than XML for reasons I have already mentioned, but my point of using scripting language was to save us the trouble of creating another language and make it more easily adoptable. If I were new to HiveMind, I wouldn't want to worry about the schema and factory and friends. I would simply want to say here's my service point and here's the implementation, you do your thing and get me the service. Or here's the configuration point and here's all the contributions. Don't get me wrong, I still like all the concepts in HiveMind, its just the usage 😉 And just to make it clear, there are no flames here!

ColinSampaleanu: I don't know if you're looking for outside opinions, but I hate the idea of inventing a new format like SDL. I agree it's a a bit cleaner and easier to type than XML, but ultimately it doesn't seem to me the gain is that great that it makes up for the fact that it's completely proprietary and nobody is able to leverage the work of anybody else (i.e. other tools able to read the same format). For this reason, YAML or even a scrpting solution like Groovy or BeanShell are preferable to me, along with XML which should remain a standby.

ChristianEssl: I like BeanShell more and more. Especially Harish's format looks quite readable and the Eclipse support is especially cool. What I still don't like is the way of constructing service-impl. As said it's not only unconvinient but also prevents the use of other ServiceImplementationFactories. I'd like to see this happen through a script as well. Sorry that I did not respond to the closure question, but I was thinking of a solution and I've now at least an idea how it maybe could work:

Instead of giving the implementation method a map or array, it should be given the name of a method defined in the script. The method would have the same signature and the same function as ServiceImplementationFactory.createCoreServiceImplementation. Now the implementation() method would look up the corresponding BshMethod serialize it and store it (encoded) in an Element. Than it would setup the implementation-descriptor so that a special BeanShellServiceImplementationFactory is called with the Element as parameter. During runtime the Factory would just take the BshMethod (cache it) and execute it. (This means of course that the method is not executed in the context of the descriptor-script - so it's not a closure). I think a similar aproach could be used for interceptors.

Beside of this I like BeanShell, because I think most of the verbosity of xml-descriptors for containers (not only for HiveMind) comes from cut-and-paste for common tasks. Ie if you want to add a logging interceptor to five services you have to type the whole implementation stuff 5 times where only the service-id changes. Compare this with BeanShell where you just define a method (in a lib) and pass it an array of service-ids.

DieterBogdoll: To be honest, I don't understand the necessity for a SDL. But if you all need them its okay for me. But I think it is utmost important to **keep** XML as a configuration mechanism.

Mike Henderson: I've been looking hard at Groovy and Groovy Markup gives you something like SDL without having to write a parser. I've implemented a builder class for connecting beans together with Groovy Markup: http://www.behindthesite.com/blog/C663408438/E876434710/index.html

HowardLewisShip: I'm not saying that you **can't** build beans using the scripting language of your choice. I'm simply saying that doing so, you may lose some of the benefits of HiveMind. HiveDoc, for starters. Line precise error reporting, potentially. And it adds to the dependencies. I'd certainly endorse the idea of an add-on library to let you use scripting if that's your way, but I don't want to see the core of HiveMind screwed just to support scripting, which only *some* people think is the One True Path.

SteveGibson: For us, we use a lot of XML already, so being able to keep XML for me is very important. A lot of people are saying they don't want to invent a new language - SDL. Well, to me, this looks a lot like IBMs Stanza format, used in AIX forever, and also the configuration format a company I used to work for came up with - purely because you specify the object you want to create and assign the properties. I think the SDL is very readable, and would probably use it if we didn't already use lots of XML files and *gasp* properties files.

NareshSikha: I hope that the bulk of Hivemind Services and Configurations are written by junior developers who are more concerned about their resume than the strength of the architecture they are fulfilling. This will afford more time for architects and leads to solve the interesting problems. Therefore I think **t ools** should be delivered to help developers author valid module definitions (Eclipse plugins are a good starting point). Then the details of the definition are more or less irrelevant. Barring that, please, please, give junior developers a chance to strengthen their resume by allowing for industry standard means for communicating metadata (XML).

AchimHuegen: Now that I worked with SDL I would like to share my opinion: I don't think the benefit of SDL is big enough to counter-balance the disadvantages. The main differences to XML are the use of curly braces instead of start and end tags and the way of quoting. The use of braces enhances readability on the one hand but reduces it on the other hand for larger files. Apparently the verbose xml format is sometimes even helpful. I couldn't resist to quote the main hivemodule.sdl, sorry howard 😉 :

```
    }  // element construct
  } // parameters-schema
```

Use of quotation marks is simply inverted, which could be a bit confusing:

```
XML:
<bean name="bean2">
  <string>testValue2</string>
</bean>

SDL:
bean (name=bean2)
{
  string { "testValue2" }
}
```

The differences are subtle but must be learned by developers. Furthermore, there are still some open questions like encoding (character sets). I think, the main problem behind "too much xml" is not the xml format itself, but the definition of verbose schemes and the shifting of programming logic to xml files. The definition of proprietary formats by each library is no solution for these problems.

MicahSchehl: XML is great because of it's wide acceptance-- but it is difficult to read and I am often overwhelmed with XML I am not familiar with. I would like to see a project which aims to allow anyone to use syntax that they desire for configuration files. The project would interpret the config file and generate the XML that so many other projects expect. The generation could be done at compile time (generate XML files) or at runtime (SAX events). The project could have default interpreters such as JavaScript or SDL. This is just my first thoughts on this.

RichardClark: BeanShell looks like a fairly dangerous choice, IMO. Both of its licenses (LGPL or Sun's community license) create interesting problems for the deployer in a commercial environment (the LGPL pushes you to set up a source code distribution mechanism, while Sun's license carries detailed documentation requirements to keep the lawyers happy); this could be a show-stopper to adoption in many places. SDL or XML might make a developer work harder to learn HiveMind, but the wrong license will get the lawyers to kill HiveMind's usage cold.