OpenArm Agent

1. Configuring the OpenArm Agent

There are basically 2 aspects to configuring the agent functionality of OpenArm:

- Write code to instantiate the OpenArm Agent and the ARM Factories
- Configure the agent by providing either a) a configuration file or b) an instance of the interface OpenArmConfiguration.

1.1. Instantiate the Agent and the ARM Factories

Instantiating and configuring the OpenArmAgent is actually pretty simple. You instantiate an instance of OpenArmTransactionSimpleFacade, passing the constructor:

- either the fully qualified path of the configuration file
- or an instance of `OpenArmConfiguration`

That's it! Here's an example:

```
                    final OpenArmTransactionSimpleFacade openArm = new OpenArmTransacti
                    Application");
```

Typcially, you will want to instantiate and hold a single instance of this class for your eintire application.

Later, when you want to instrument a method, you obtain an instance of ArmTransaction from the Facade like this:

```
                    final ArmTransaction transactionMonitor = this.openArm.getArmTransa
```

... where "someMethodName" will be used as the "name" of the "transaction" that ARM will be measuring.

Directly instantiating the raft of Factory objects that the ARM standard requires is a bit more involved. Basically, it involves the following:

- You get an instance of ArmTransactionFactory from the OpenArmAgent
- You use the ArmTransactionFactory to instantiate an instance of ArmApplicationDefinition
- You pass the ArmApplicationDefinition to the ArmTransactionFactory to get instances of

ArmTransactionDefinition and ArmApplication

Later, when you want to instrument a method, you pass both the ArmApplication and the ArmTransactionDefinition instances to the ArmTransactionFactory to obtain an instance of ArmTransaction. You use the ArmTransaction to instrument your "transaction", whatever that might mean for your application.

In many cases, you will probably want to define and instantiate any number of ArmTransactionDefinitions; one for each distinct type of transaction in your application. In contrast, you will typically only have one instance each of ArmApplication, ArmApplicationDefinition and ArmTransactionFactory.

Here's an exmaple of how this might look:

```
    private ArmTransactionFactory armTransactionFactory;
    private ArmApplicationDefinition armApplicationDefinition;
    private ArmApplication armApplication;

    private ArmTransactionFactory initializeArmTransactionFactory(final String fileName
        final OpenArmAgent openArmAgent = new OpenArmAgent(fileName);
        final ArmTransactionFactory factory = openArmAgent.getTransactionFactory();
        return factory;
    }

    private void initializeArmApplicationGlobals(final String fileName) {
        armTransactionFactory = initializeArmTransactionFactory(fileName);
        armApplicationDefinition = armTransactionFactory.newArmApplicationDefinition("T
        armApplication = armTransactionFactory.newArmApplication(armApplicationDefiniti
    }

    public ArmTransactionDefinition getArmTransactionDefinition(final String transactio
        final ArmTransactionDefinition transDef = armTransactionFactory.newArmTransacti
                armApplicationDefinition,
                transactionName,
                null,
                armTransactionFactory.newArmID(null));
        return transDef;
    }

    public ArmTransaction getArmTransactionMonitor(final ArmTransactionDefinition trans
        final ArmTransaction transactionMonitor = armTransactionFactory.newArmTransacti
        return transactionMonitor;
    }
```

Note:

There are a few things worth mentioning about this sample code.

- Note that, other than the instance of OpenArmAgent, everything else is typed as an instance of one of the standard ARM interfaces. Thus, only the code in the method

#initializeArmTransactionFactory is dependent on OpenArm.

- Note that the ArmTransactionFactory, ArmApplicationDefinition, and ArmApplication are held as instance variables. Typically, you will only have one instance of each of these objects per application context.
- Note, on the other hand, that there are factory methods provided for obtaining instances of ArmTransactionDefinition and ArmTransaction -- this is also typical. The usage pattern here is to obtain a new ArmTransactionDefinition, and use it to obtain a new ArmTransaction, often on a per method basis.
- Note also the somewhat sick looking method signatures, where lots of parameters are actually optional. The ARM interfaces prescribe these method signatures, whether they make sense in all use cases or not. The jury's still out on this one -- at the moment, we handle the plethora of "optional" parameters by passing lots of `null` references. However, it might be prettier to hide this complexity, by adding a raft of overloaded signatures for each of the signatures that ARM prescribes. We'd still have to pass around a lot of `null` references, but we could hide that in the implementation, and keep them from cluttering up end-user code...
- Finally, note the use of the variable named `transactionMonitor` . This variable is typed as an `ArmTransaction` , and is the main object that one works with when instrumenting a method. The Open Group's literature, and the ARM Spec, refer to this entity as *the* transaction -- so why not name the variable simply `transaction` ? Well, because the thing this variable represents is **not** your transaction -- only you know what that term means, in the context of your application. The thing that I've chosen to name a `transactionMonitor` is a tool that you use to instrument and watch over *your* transaction. So I think `transactionMonitor` is a better name -- clearer in intent and meaning. I'll consistently name these things this way, in all of the code samples.

1.2. Provide the configuration for the OpenArmAgent

This is complex enough to deserve its own page ...