Adapting Apache UIMA to OSGi
Tommaso Teofili | Adobe Basel R&D
Agenda

- The problem:
  - not-OSGi stuff in an OSGi world
  - use case
- Possible adaption approaches discussion
  - an approach for our use case
- Conclusions
Use case: Domeo

- Annotation toolkit
- create & share ontology based annotations
- using AnnotationOntology RDF model
- supporting (semi) automated annotation
  - Apache UIMA based text mining applications integrate via Apache Clerezza platform
  - **but** ... Clerezza is OSGi based while UIMA is not
Apache UIMA

- ASF top level project since 2010
- Unstructured information management architecture
- OASIS standard
- Powering
  - apps: IBM Watson, DeepQA, ...
  - platforms: Behemot, Domeo, U-Compare, ...

![Diagram of the Aggregate Analysis Engine and CAS Annotations]

**CAS Annotations**
- Tokens
- Parts of Speech
- Names
- Organizations
- Places
- Persons
Apache Clerezza

- Apache Incubator project
- Service platform for linked data
- RDF API
- JAX-RS implementation
- OSGi based
- Powering Apache Stanbol semantic content management project
Clerezza – UIMA architecture overview

- JAX RS
- Clerezza platform
- AO
- OSGi container
- Triple store

UIMA Pipeline

AE  AE  AE  AE  AE
The problem

- Apache Clerezza
  - is OSGi based (running on Apache Felix)
  - artifacts delivered as bundles
- Apache UIMA
  - is not OSGi based
  - is not OSGi ready
  - artifacts delivered as standard jars / PEARs
  - using reflection / custom classloading
  - has some OSGi artifacts for UIMA Eclipse plugins
- How to flawlessly let UIMA live and run inside Clerezza?
Adapting to OSGi

- Things to consider:
  - backward compatibility
  - development effort
  - classloading
  - packaging / deployments
  - semantic versioning

- Business requirements are crucial to prioritize the above list
Adaption approaches

- migrate everything to support (also?) OSGi
- migrate something in a single lightweight bundle
- migrate nothing and wrap everything in a single big bundle
- create an adaption layer
- ...
Approach #1 – migrate everything

• pros
  • fully OSGi compatible
  • one time effort
  • fully leveraging OSGi capabilities
• cons
  • backward compatibility issues
  • significant development effort
  • consider integration effort
  • possibly keep / maintain different architectures at once
  • not always feasible
  • need full control
Approach #2 – migrate something

• pros
  ▪ fully OSGi compatible
  ▪ easier limited development effort
  ▪ facade package

• cons
  ▪ need to keep maintaining OSGi and not-OSGi artifacts over time
  ▪ not everything is available to OSGi client architectures
  ▪ need full control on codebase
Approach #3 – just wrap

- **pros**
  - fully OSGi compatible
  - almost no development effort
  - everything is available to OSGi client architectures

- **cons**
  - just not leveraging OSGi capabilities
  - possible classloading issues
  - need to keep managing OSGi and not OSGi artifacts over time
Approach #4 – adaption layer

- **pros**
  - fully OSGi compatible
  - everything is available to OSGi client architectures
  - original not-OSGi stuff is not touched
  - no need to control not-OSGi stuff

- **cons**
  - some application specific development effort needed
  - explicitly managing classloading issues
  - need to keep managing this adaption layer
Managing requirements

- **Domeo concerns**
  - OSGi compatibility
  - ease of development for UIMA users with (almost) no OSGi experience
  - leverage OSGi capabilities

- **Communities concerns**
  - Apache UIMA community was interested in OSGi but:
    - not willing to drop backward compatibility
    - not willing to migrate everything to OSGi right away
  - Apache Clerezza community was happy to have UIMA onboard but:
    - didn’t want to have release blocked by UIMA OSGi adaption
Feasible approaches

- migrate everything > NO
  - need to touch the code base, not backward compatible
- migrate something > to a degree
  - existing UIMA users may need stuff not included in the lightweight pack
- just wrap > NO
  - classloading issues
- adaption layer > YES
  - but need to develop such a layer :-)
UIMA OSGi adaption layer known approaches

- DME for UIMA from IBM AlphaWorks
- Clerezza – UIMA integration module
- UIMA integration for Apache Stanbol
- UIMA OSGi enablement via Maven
- ...

16
UIMA Dynamic Module Enabler
Clerezza – UIMA adaption layer

- simple API on top of UIMA API
  - create pipeline executor, give it a UIMA descriptor, run
- still can use plain UIMA API
- each bundle using UIMA just needs a specific OSGi activator
- lightweight
Consider an existing text mining application using UIMA for some NLP task that needs to be plugged in Domeo

- it needs:
  - ReST endpoint
  - RDF API for dealing with ontologies
  - AnnotationOntology model awareness

- it already has:
  - one or more UIMA descriptors for its NLP pipeline(s)
  - UIMA components (either developed internally or publicly available)

Let’s assume our sample service has one UIMA pipeline with one descriptor and uses Apache OpenNLP to extract person names from text
@Property(name = "javax.ws.rs", boolValue = true)
@Path("/uima")
public class OpenNLPSampleService {

.. 

@POST
@Path("opennlp/person")
@Produces("application/rdf+xml")
public Graph extractPersons(@FormParam("uri") String uriString) {

    UIMAExecutor executor = ... ;
    Map<String, Object> parameters = ... ;
    parameters.put(“graph.name”, uriString);
    executor.analyzeDocument(text, new XMLInputSource(descriptor), parameters);
    }

.. 
}

Clerezza – UIMA : sample service dependencies

- OSGi container
  - OpenNLP sample service
  - Clerezza UIMA utils
  - OpenNLP
  - UIMA SDK

- depends on
  - OSGi
  - not OSGi
Clerezza – UIMA : sample service classloading

- Our sample service
  - has OpenNLP stuff in its classloader
  - gets UIMA stuff from Clerezza UIMA utils bundle
  - nothing new but ...
- UIMA uses XML descriptors in which component specific class names are given in order to instantiate pipelines

```xml
<analysisEngineDescription xmlns="http://uima.apache.org/resourceSpecifier">
  <frameworkImplementation>org.apache.uima.java</frameworkImplementation>
  <primitive>true</primitive>
  <annotatorImplementationName>opennlp.uima.NameFinder</annotatorImplementationName>
</analysisEngineDescription>
```

- UIMA SDK uses the “famous” Class.forName(...)  
- everything should break in our case when instantiating pipelines since Clerezza UIMA utils bundle doesn’t hold OpenNLP stuff in its classloader
Clerezza – UIMA: runtime classloading dependencies

OSGi container

- OpenNLP sample service
- Clerezza UIMA utils
- OpenNLP
- UIMA SDK

needs class from

OSGi
not OSGi
we can leverage
  - OSGi activators
    - upon bundle activations notify “someone” that new UIMA components are available
  - UIMA resource managers
    - upon pipeline execution we can use a resource manager with a custom classloader

Solution:
  - each bundle using UIMA extends a specific OSGi activator (org.apache.clerezza.uima.utils.UIMABundleActivator) to explicitly declare its UIMA components
  - the Clerezza UIMA utils module registers such components classloaders within an aggregate classloader which is passed to the UIMA resource manager at runtime
Clerezza – UIMA : managing classloading issue

Sample service bundle

- UIMABundleActivator
- UIMACOMPONENTX
- OpenNLP Sample Service

Clerezza UIMA utils bundle

- UIMAResourcesClassLoaderRepository
- ResourceManager
- AEProvider
- UIMAEEXECUTOR

ServiceRegistry

OSGi container
Conclusions

- Achievements
  - UIMA codebase not touched
  - fully OSGi compatible
  - lightweight

- Left outs
  - semantic versioning
  - more OSGi capabilities could be leveraged (lots of UIMA API stuff could be exposed with OSGi services)

- Lessons learned
  - not only one good way
  - full OSGi migration is often not simple in the enterprise
  - using BNDTools / maven-bundle-plugin is sometimes not enough for migrating bundles because of application specific classloading issues
Thanks!

- We’re hiring:
  - jobs-basel@adobe.com

- Q&A