JUDCon

JBoss Users & Developers Conference

2012:Boston

# jBPM & Drools go Enterprise

Maciej Swiderski

Build comprehensive BPM platform on top of jBPM and Drools that will truly accelerate your business

## Sounds nice but what's that?

How easy is to:

- Introduce new (version of) process?
- Change logic of a process?
- Upgrade your environment?
- Migrate your active processes?

## ... the goal is to be able...

- Add new (version) processes without affecting already running instances
- Alter business logic invoked by the processes independently
- Run different versions of the engine at the same time

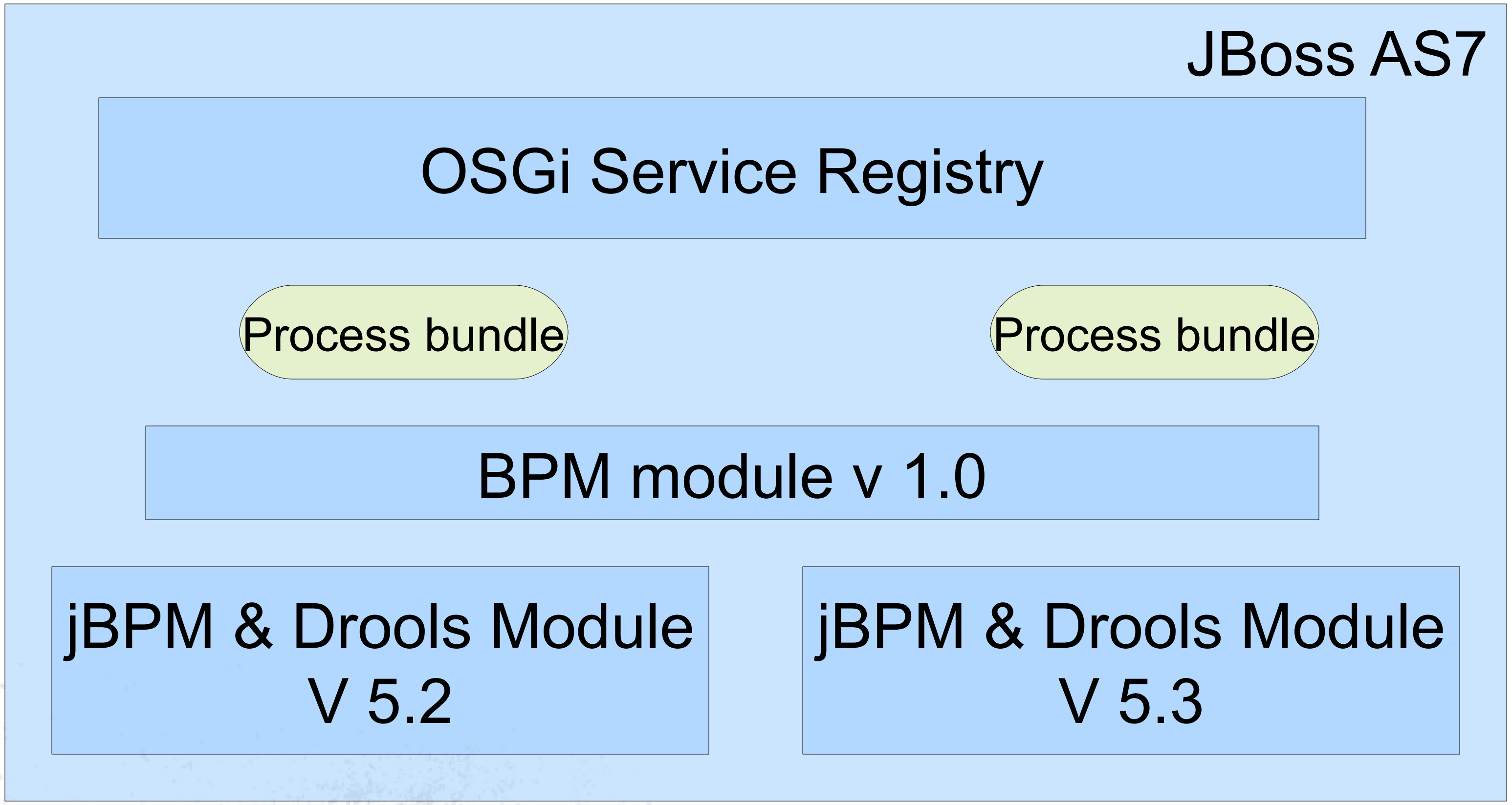
## ... the developer goal is to be able...

- Do not maintain knowledge sessions on application/client side
- Do not worry about version if not needed
  - Process version
  - Engine version
- Simplify usability of the engine from application/client

## JBoss AS7 to the rescue

- JBoss Modules
  - jBPM and Drools configured as JBoss Module with all their dependencies
- OSGi
  - Engine factories registered in OSGi service registry with version properties
  - Engines registered in OSGi service registry with custom properties
  - Client code packaged as OSGi bundles

## Platform overview



## Platform components

- BPM module
  - Registers resolver manager
- JBPM & Drools module
  - Registers ExecutionEngineFactory
  - Registers resolvers
- Platform bundle
  - Bootstraps and registers ExecutionEngine
- Client application
  - Uses ExecutionEngine via resolvers

## BPM module

- Simple abstraction layer on top of jBPM and Drools APIs to make clients independent of the version
- Registers ResolverManager in OSGi service registry as single point of interaction for ExecutionEngine look ups

## ResolverManager

```
public interface ExecutionEngineResolverManager {

    void register(String owner, ExecutionEngineResolver resolver);

    void unregister(String owner, UUID resolverUniqueId);

    ExecutionEngineResolver find(RequestContext context);

    ExecutionEngine findAndLookUp(RequestContext context);

    Collection<ExecutionEngineResolver> getResolvers();
}
```

## jBPM & Drools module

- jBPM and Drools components bundled in a single JBoss Module together with all dependencies
- OSGi enabled
- Registers ExecutionEngineFactory that is exposed to process bundles to construct ExecutionEngines for given version of jBPM and Drools
- Registers resolvers supported by given version

## ExecutionEngineFactory

```
public interface ExecutionEngineFactory {
    public ExecutionEngine newExecutionEngine(
                                ClassLoader bundleClassLoader);
    public ExecutionEngine newExecutionEngine(
                          ClassLoader bundleClassLoader,
                          ExecutionEngineConfiguration config);
    public ExecutionEngine newExecutionEngine(
                                ClassLoader bundleClassLoader,
                           ExecutionEngineConfiguration config,
                                             Object callback);
    public ExecutionEngine newExecutionEngine(
                                ClassLoader bundleClassLoader,
                           ExecutionEngineConfiguration config,
                        ExecutionEngineMapperStrategy strategy,
                                             Object callback);
}
```

## Process bundle

- Main component that makes use of the platform and delivers functionality
- Produces ExecutionEngine which is:
  - Wrapper around KnowledgeBase
  - Provides session management based on business keys using configurable strategies
- Registers ExecutionEngine under various properties making it discoverable by resolvers

## ExecutionEngine

```
public interface ExecutionEngine {
    public Object getKnowledgeBase();
    public SessionDelegate getStatelessSession();
    public SessionDelegate getSession(String businessKey);
    public SessionDelegate getSessionById(int id);
    public Object getHumanTaskConnector();
    public UUID getUUID();
    public String buildCompositeId(String id);
    public void disposeSession(SessionDelegate session);
}
```

## Client application

- Ultimate client of the platform
- Makes use of ExecutionEngine and ResolverManager to perform work
- Independent of the platform and process version
- Communicates only through OSGi service registry

## Client application code

```
// get reference to Resolver manager
ServiceReference srf = this.context.getServiceReference(
                ExecutionEngineResolverManager.class.getName());
ExecutionEngineResolverManager resolverManager =
    (ExecutionEngineResolverManager)this.context.getService(srf);

//find right resolver and directly look up the engine
RequestContext reqContext = new HttpRequestContext(request);
ExecutionEngine engine = resolverManager.findAndLookUp(reqContext);

// get session by business key and start process on it
String compositeProcessInstanceId = engine.getSession("business-
key").startProcess(”process-id”);
```

## Resolvers

- Resolver is responsible for finding the right ExecutionEngine based on given context
- Default policy - first resolver that accepts the context will do the look up in OSGi service registry
- Platform delivers some resolvers out of the box but process bundles can introduce custom resolvers as well

## Available default resolvers

- UUID based resolver that accepts context if:
  - Explicitly contains UUID property of the engine
  - Contains composite id property (for instance processInstanceId)
- Version based resolver that accepts context if:
  - Explicitly contains version property
- Valid time resolver that will accepts the context if:
  - No version parameter is given

## Session management

- In case where more than one session is in use there is a need to keep track of the identifiers and in some cases even relationship between process instance and session instance
- By default this need must be secured on application side
- On clustered environment things get more complicated (avoid concurrent usage of the same session)

## SessionMappingStrategies

- Platform is equipped with strategies that are capable of maintaining sessions identifiers based on some business key
- Application refers to the session with custom business key like for instance user id or department instead of the internal session id
- Strategies are pluggable and every ExecutionEngine instance can utilize different implementation

## Available session mapping strategies

- SerializableMap strategy – dedicated strategy for standalone installation that will simply persist the map of known values to the disc
- Clustered strategy – dedicated strategy that will employ Inifinispan as distributed cache with configured data store

## Make your engine configurable

- ExecutionEngine will emit notification on number of events so the process bundle can react on them:
  - Knowledge base creation
  - Knowledge session creation
  - Work item registration
  - Dispose of the session
  - etc

# ExecutionEngineCallback

```
public interface ExecutionEngineCallback {
    public void preKnowledgeBaseCreate(KnowledgeBuilder builder);
    public void postKnowledgeBaseCreate(KnowledgeBase kBase);
    public void preKnowledgeSessionCreate(Environment environment,
                                          KnowledgeSessionConfiguration config, KnowledgeBase kBase);
    public void postKnowledgeSessionCreate(StatefulKnowledgeSession session, String businessKey);
    public void postKnowledgeSessionCreate(StatelessKnowledgeSession session, String businessKey);
    public void preKnowledgeSessionRestore(Environment environment,
                                           KnowledgeSessionConfiguration config, KnowledgeBase kBase);
    public void postKnowledgeSessionRestore(StatefulKnowledgeSession session, String businessKey);
    public void preSessionDispose(StatefulKnowledgeSession session, String businessKey);
    public void postSessionDispose(StatefulKnowledgeSession session, String businessKey);
    public void preWorkItemRegister(StatefulKnowledgeSession session, String businessKey,
                                    KnowledgeBase kBase, WorkItemHandler handler);
    public void postWorkItemRegister(StatefulKnowledgeSession session, String businessKey,
                                     KnowledgeBase kBase, WorkItemHandler handler);
}
```

## Multi tenancy

- Multi tenancy is achieved by:
  - Separate process bundles registered with dedicated properties
  - Additional resolver that will understand tenant configuration

## Still in evaluation phase...

- The work is still in evaluation stage so everything can change and hopefully if it does it's for the good
- Please submit your input, requirements, ideas
- There are some limitation currently that are being investigated and most of them have workarounds :)

## Thanks for your attention

Questions? Comments?

More than welcome – get in touch via:

Email: mswiders@redhat.com

IRC: chat.freenode.net#jbpm