A Service API for Deployment

Steve Loughran
HP Laboratories
steve_loughran@hpl.hp.com

Goals

- Widely usable deployment API
- Public face of CDDLM Basic Services
- Integrate with portal, resource manager
- Language agnostic
- Use core WS-I SOAP, if possible, for interoperability
- Be extensible

Explore what works, what doesn't

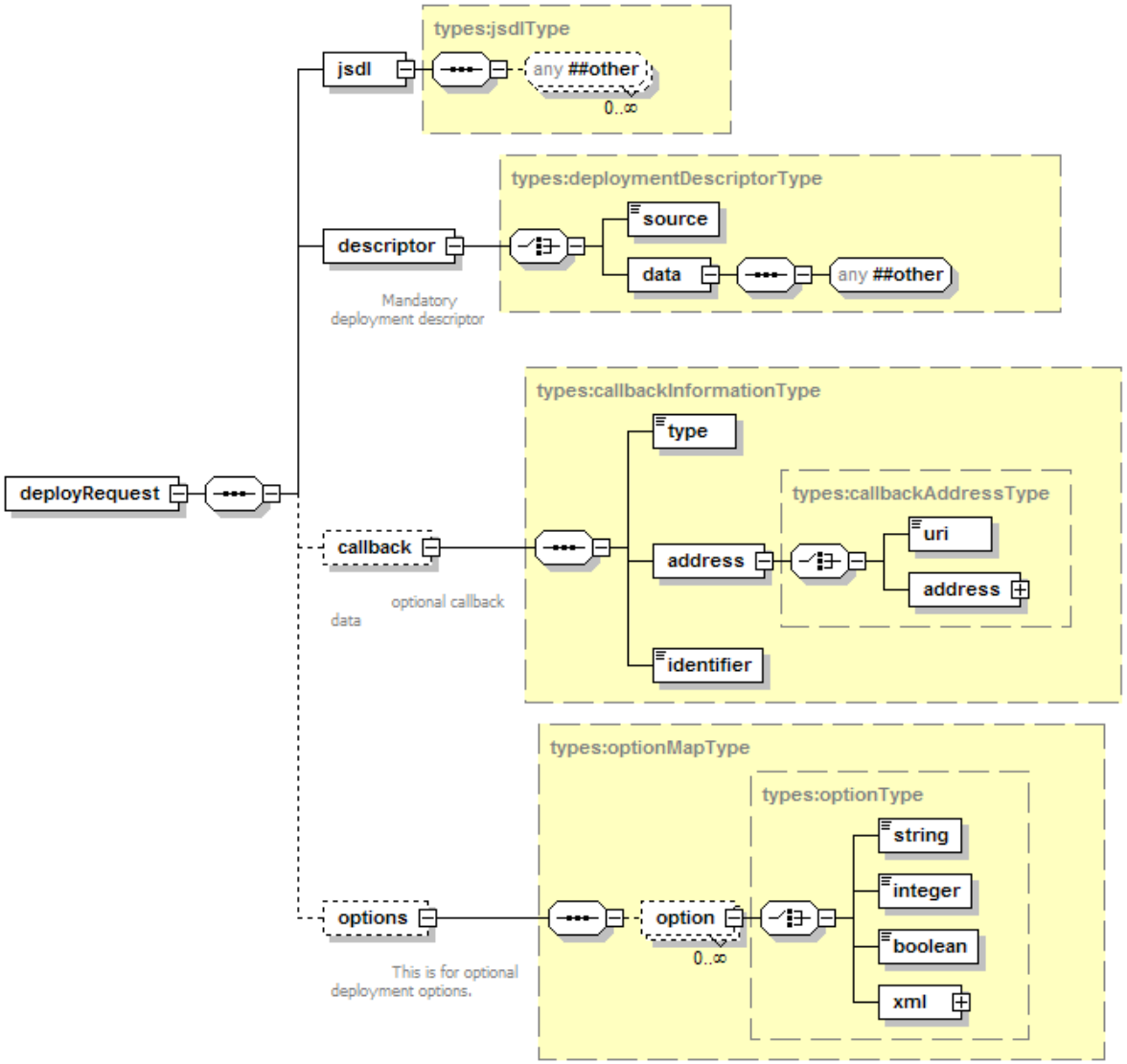
XML Layer

- SOAP1.1, WS-I level
- Seven operations:

```
deploy
undeploy
serverStatus
applicationStatus
listApplications
setCallback
(lookupApplication)
```

- ~2000 lines of XML Schema!

Deploy



[JSDL metadata]

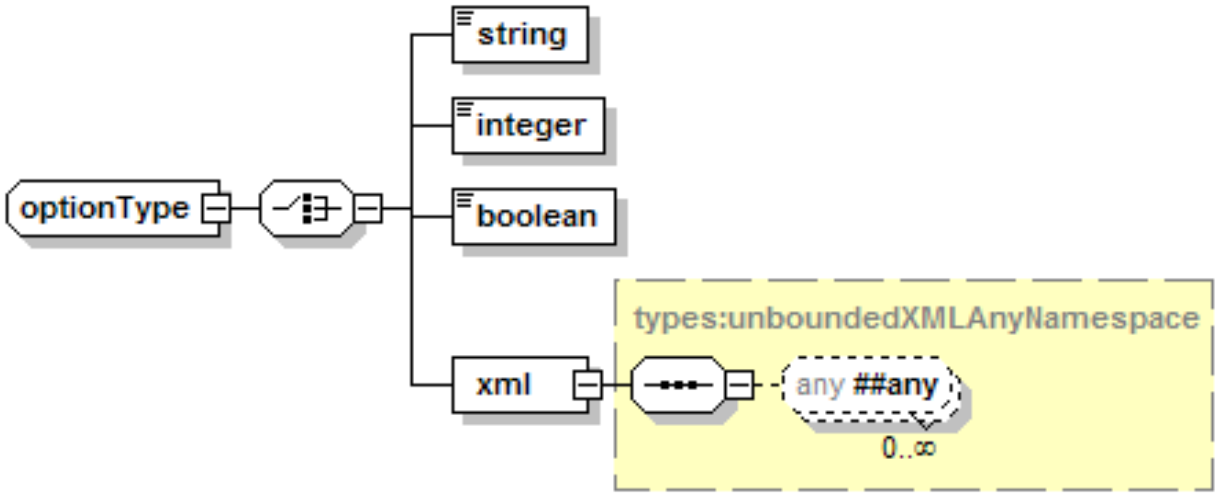
XML descriptor

[callback]

[options]

=> URI

Deployment supports optional extensions



- a list of options
  - `name`: URI
  - `mustUnderstand`:boolean
- SOAP Header semantics

Defined optional extensions

`validateOnly`: validate descriptor, do not deploy

`name`: unique name for the application

`propertyMap`: name-value pairs of late binding information

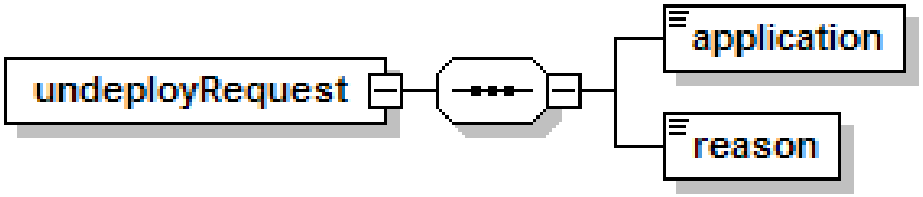
extractable from configuration file (e.g. by SmartFrog `PROPERTY`) keyword.

Naming

- URI for deployed applications.
  - Unique to that machine
  - use wall time and counter to keep unique over restarts

- Option for explicit 'application name'
  - SmartFrog runtime permits cross application resolution
  - Proved to brittle for testing, let alone deployment
  - Retained as a vestigial/optional feature
  - Recommend: use real naming/dir/location services

undeploy



- idempotent
- asynchronous
- not an error to undeploy a nonexistent application

applicationStatus

```
<applicationStatusResponse>
  <reference>http://cddlm/job2</reference>
  <name>job2</name>
  <state>running</state>
  <stateInfo xsi:nil="true" />
</applicationStatusResponse>
```

- probes application health (blocking call)
- optional arbitrary `stateInfo` section
- Recommend: make fault information explicit

Callbacks raised on lifecycle events

- Message contents:
  - Application URI
  - Application state
  - Caller supplied identifier
  - Application Status
  - `xsd:any`
- Callback information can be in a `deploy` request
- `setCallback` operation sets a callback on a running application
- `setCallback=null` to unsubscribe.

Trouble with Callbacks

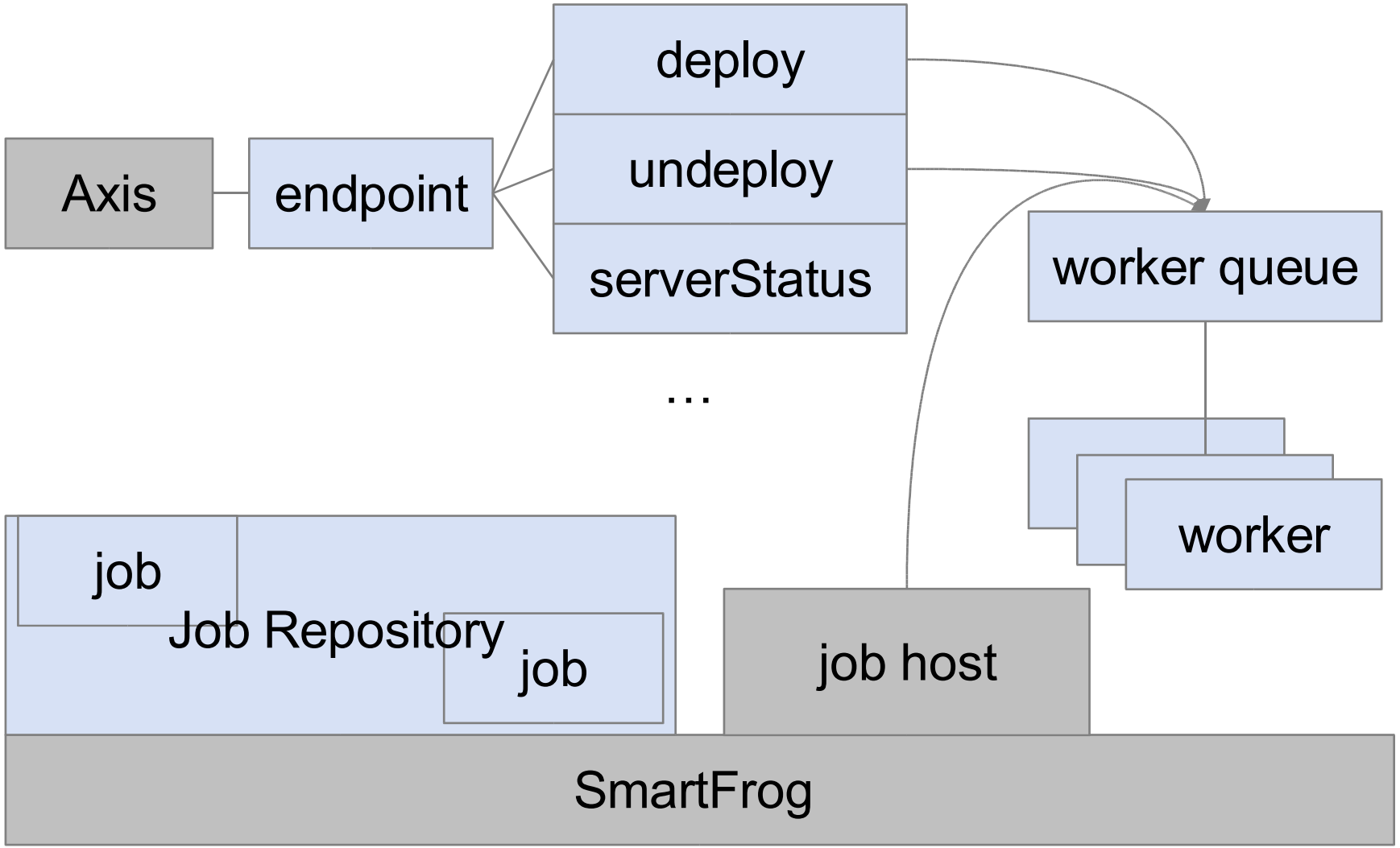
- No standard callback mechanism
- WS-Notification v. complex
- WS-Notification/WS-Eventing convergence?
- Firewalls?
- Jabber/XMLPP goes through firewalls!

- Prototype just has simple direct callback endpoint
- notifications are queued; failures ignored.

Implementation

- Java based
- Uses Apache Axis 1.2beta for SOAP
- Built on SmartFrog
- Hosted in SmartFrog (Axis Component, CDDLM component supplies a new endpoint)
- Mildly Asynchronous (worker threads for notification callbacks)
- Deploys SmartFrog descriptors
- Parses and Validates XML-CDL descriptors

Architecture



Issue: Faults

- Not using WS-BaseFaults: *too inflexible "BaseFault does NOT include open element extensibility"*
- requires explicit typing of all possible fault content
- requires explicit naming of all throwable faults

the "checked exception" pattern

- using classic SOAPFault
- constants.xml declares fault codes

Not yet addressed

- Logging
  - need dynamic reconfigure of running logs
  - feeding of log data (buffered?) to monitors
  - caching of data for polling
- Code/data file provisioning
  - assume portal copies files to shared file system
  - could handle attachments, though that is an interop black spot.
- Security

JAX-RPC considered wrong

- JAX-RPC is the standard Java SOAP API
- Built around O/X mapping "serialization"
- Some support for arbitrary XML in MessageElement class (JAX-M) derivative:

```
public Document getAsDocument() {
 String s = getAsString();
 Reader reader = new StringReader(s);
 InputSource s=new InputSource(reader)
 return XMLUtils.newDocument(s);
}
```

This API is hopeless for supporting arbitrary XML

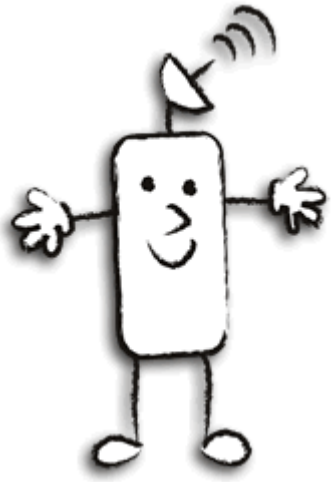
Thinking of an alternative: Alpine

- doc/lit, SOAP1.2, Java1.5 only
- No O/X binding: use XOM or XMLBeans
- No client/server distinction: processing chain
- Multi transport
- non blocking
- JMX management
- Fast, lightweight
- For experts only

Next Steps

1. integrate with a front end (the Iceni one?)
2. integrate with a resource manager
3. incorporate feedback/experience
4. work on trouble areas: callbacks, faults, security, logging
5. implementation: XML-CDL support as it evolves

product placements:



Mindreef SOAPscope 4.0