

# Information Service Workshop

## *Cactus Metadata Management*

Thomas Radke (AEI)

5<sup>th</sup> AstroGrid Meeting at MPE Garching  
14./15. November 2006

- Corresponding to our concrete application use cases there are different types of Cactus metadata
  - 1) metadata announced by running simulations
  - 2) metadata generated during automatic integration tests
- since the metadata to be described are very application-specific, each type has its own RDF vocabulary
- vocabularies are defined as OWL documents in RDF/XML
  - ◆ `cctk-schema`
  - ◆ `cactus-integration-tests-schema`
- classes and properties are used to describe the metadata



# Cactus Simulation Metadata



Metadata announced from within running simulations

- ◆ executable name + version
- ◆ job owner, run date, run host, nprocs, cwd
- ◆ name and contents of the parameter file
- ◆ all thorns with all parameters and their values
- ◆ user-defined dynamic runtime information
  - current physical simulation time and iteration
  - events (horizon found, NaN detected, etc.)
  - termination condition





# Automatic Integration Tests (I)



Combination of interdependent unit tests for a given Cactus configuration (thornlist, config options, #procs)

1. freshly checkout development version of Cactus and thorns
2. create a configuration with given machine-specific options: compiler settings, optimisation, MPI, external packages
3. build the configuration (run `make` to produce an executable)
4. build all utility programs
5. run all available testsuites on the given number of processors and evaluate their status

All tests are done within a perl script which is run as a nightly cron job and sends the metadata to an RDF server.



- The script sequentially runs all unit tests and generates an RDF/XML document with the following metadata
  - ◆ unique integration test description:
    - configuration name, thornlist, options
    - user, host, #procs
    - datetime stamp
  - ◆ status of each unit test (passed / failed / skipped), `stderr` log for failed tests
  - ◆ status of all Cactus testsuite runs, in case of failure either with `difflog` or `stdout/stderr` log

- `rdfs:Class` defines a class
  - class instances represent subjects and objects in an RDF statement
- `rdf:Property` defines a property
  - property instances are actual attributes of subjects
- Get a copy of the schema and have a look at it

```
gsiscp -P 2222 \  
astrogrid.aei.mpg.de:~tradke/IS-Workshop/cactus-integration-tests-schema .
```



# An Example RDF/XML File



```
<cactus:IntegrationTest rdf:about="#WaveToy-mike4.cct.lsu.edu-1-tradke-2006-11-02T07%3a52%3a02"
  cactus:name="WaveToy-mike4.cct.lsu.edu-1-tradke-2006-11-02T07:52:02"
  cactus:host="mike4.cct.lsu.edu"
  cactus:nprocs="1"
  cactus:user="tradke"
  cactus:configuration="WaveToy">
  <cactus:datetime rdf:datatype="&xsd;datetime">2006-11-02T07:52:02</cactus:datetime>
  ...
  <cactus:configOptions></cactus:configOptions>
  <cactus:unitTest rdf:nodeID="checkout"/>
  <cactus:unitTest rdf:nodeID="build-utils"/>
  <cactus:unitTest rdf:nodeID="config"/>
  <cactus:unitTest rdf:nodeID="build"/>
  <cactus:unitTest rdf:nodeID="testsuites"/>
</cactus:IntegrationTest>

...

<cactus:UnitTest rdf:nodeID="testsuites">
  <cactus:status>passed</cactus:status>
  <cactus:summary>25 tests in total, 20 passed, 5 failed</cactus:summary>
  <cactus:testsuite rdf:nodeID="testsuites.WaveMoL.gaussian"/>
  <cactus:testsuite rdf:nodeID="testsuites.WaveToy1DF77.wavetoy1df77"/>
  ...
</cactus:UnitTest>
```





# Querying Integration Test Results



```
# list all unit test results from the 20 most recent Cactus integration tests

PREFIX cactus: <http://somehost.org/SCHEMA_PATH_PLACEHOLDER/cactus-integration-tests-schema#>
SELECT ?name ?Configuration ?User ?Host ?NumProcs ?StartedAt \
       ?CheckoutStatus ?ConfigStatus ?BuildStatus ?BuildUtilsStatus ?TestsuitesStatus
WHERE {
  ?test      cactus:name          ?name ;                               # unique ID of this test, used as reference
            cactus:configuration ?Configuration ;
            cactus:user          ?User ;
            cactus:host          ?Host ;
            cactus:nprocs        ?NumProcs ;
            cactus:datetime      ?StartedAt ;
            cactus:unitTest      ?checkout ;
            cactus:unitTest      ?config ;
            cactus:unitTest      ?build ;
            cactus:unitTest      ?buildutils ;
            cactus:unitTest      ?testsuites .
  ?checkout cactus:name          'checkout' ;
            cactus:status        ?CheckoutStatus .
  ?config   cactus:name          'config' ;
            cactus:status        ?ConfigStatus .
  ?build    cactus:name          'build' ;
            cactus:status        ?BuildStatus .
  ?buildutils cactus:name        'build-utils' ;
            cactus:status        ?BuildUtilsStatus .
  ?testsuites cactus:name        'testsuites' ;
            cactus:status        ?TestsuitesStatus .
} ORDER BY DESC(?StartedAt) LIMIT 20
```





# Querying Testsuite Results



```
# list all testsuite results (alphabetically sorted by testsuite name)
# from a specific integration test referred to by <testID>

PREFIX cactus: <http://somehost.org/SCHEMA_PATH_PLACEHOLDER/cactus-integration-tests-schema#>

SELECT ?Testsuite ?Status ?Summary ?Difflog ?Log

WHERE {
  ?test      cactus:name      'WaveToy-mike4.cct.lsu.edu-1-tradke-2006-11-02T07:52:02' ;
            cactus:unitTest  ?unittest .

  ?unittest  cactus:name      'testsuites' ;
            cactus:status    'passed' ;
            cactus:testsuite ?testsuite .

  ?testsuite cactus:name      ?Testsuite ;
            cactus:status    ?Status ;
            cactus:summary   ?Summary ;
            cactus:difflog   ?Difflog .

  OPTIONAL { ?testsuite cactus:log ?Log } .

} ORDER BY DESC(?Testsuite)
```





# Cactus Portal Demonstration



And now  
a quick demonstration  
of how integration test results  
can be queried within the Cactus portal





# Practical Tips



For designing and coding  
the Cactus Metadata Management software  
I used the following tools and services





# Joseki: The Jena RDF Server



- Joseki is part of Jena, the RDF framework for Java.
- It contains both an RDF parser and a SPARQL query validator/processor.
- It can run both as a web service or command-line tool.
- It's available as open-source under a BSD-style license.
- Download from <http://www.joseki.org/> or get it via  

```
gsiscp -P 2222 \  
  astrogrid.aei.mpg.de:~tradke/IS-Workshop/joseki-3.0-beta-2.zip .
```
- **documentation:** `konqueror doc/documentation.html`



- shell wrapper script `parseRDF.tcsh` to validate an RDF/XML file:

```
#!/bin/tcsh
```

```
setenv JOSEKIROOT <your Joseki installation dir>
```

```
java -cp `${JOSEKIROOT}/bin/joseki_path` jena.rdfparse -s -t $*
```

- Usage: `parseRDF.tcsh metadata.rdf`
- error messages are written to stdout  
if `metadata.rdf` doesn't validate correctly



# Joseki: SPARQL Query Processor



- start the Joseki SPARQL query engine as web service:

```
#!/bin/tcsh
setenv JOSEKIROOT <your Joseki installation dir>
setenv PATH      ${PATH}:${JOSEKIROOT}/bin
cd ${JOSEKIROOT}
rdfserver --port 24100 >& joseki.log
```

- Connect with your browser:

mozilla <http://localhost:24100/>

- provides a general purpose SPARQL processor (same as <http://sparql.org/sparql.html>) as well as a SPARQL query validator





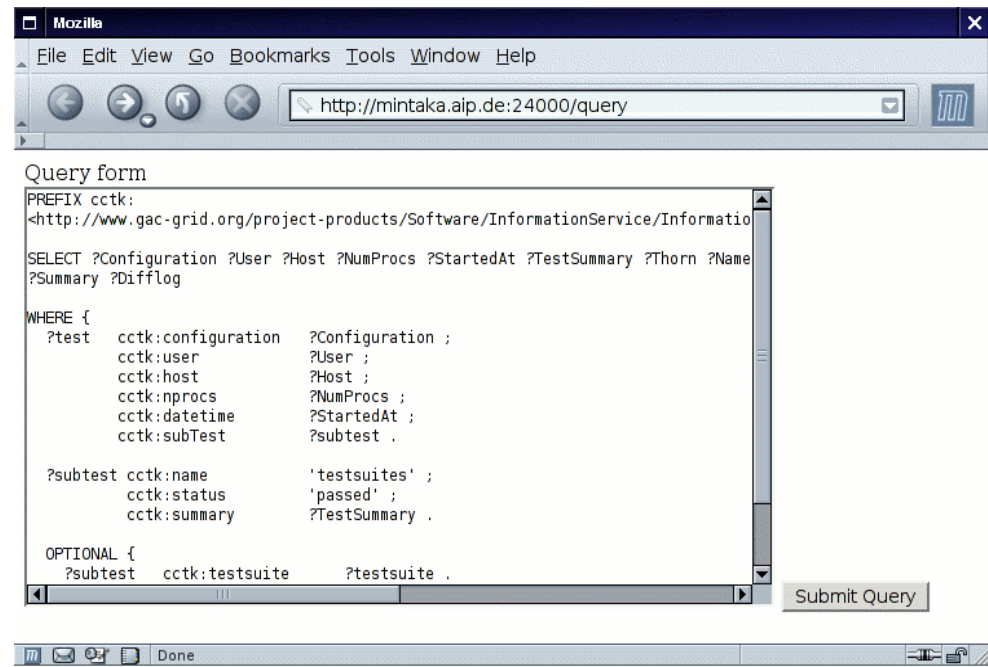
# The IS Web Service Query Interface



- AstroGrid's *IceCore* Information Service provides a web service interface for SPARQL queries, eg.

<http://mintaka.aip.de:24000/query/>

where you can input a SPARQL query in an HTML form



- HTML form doesn't let you load or save a query, and editing a query is inconvenient
- alternative solution: edit and store your query as a plain text file and turn its contents into a SPARQL query URL
- I wrote a small perl helper script which does that:

```
gsiscp -P 2222 \  
  astrogrid.aei.mpg.de:~tradke/IS-Workshop/sparql-to-url.pl .
```
- Usage: `sparql-to-url.pl <rdfhost:port>`  
eg. `cat query.rq | \  
 sparql-to-url.pl mintaka.aip.de:24000`
- paste the output as URL into your browser