

# Clusterfinder Use Case

Art Carlson (MPE)

AstroGrid-D project meeting

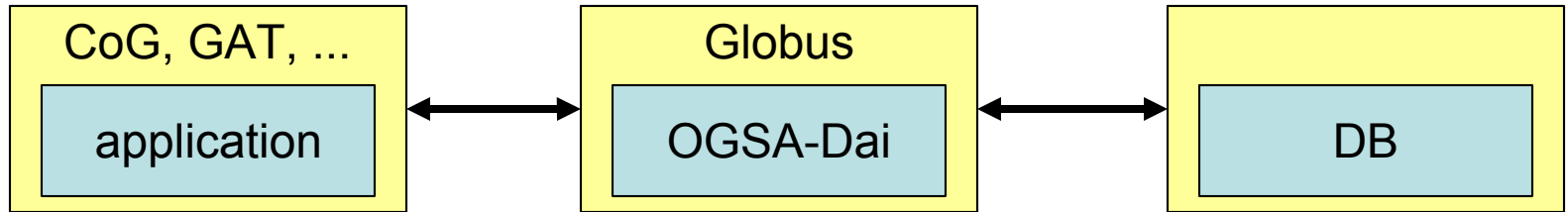
Garching, 14 Nov 2006

As a reminder, the clusterfinder processes catalogs of galaxies (SDSS) and x-ray photons (RASS) to identify clusters of galaxies.

- Data access
- Logistics
- Workflow
- Visualization

# Data Access

- Input from data bases at remote sites
- Access through OGSA-Dai
  - Authentication and authorization provided
  - Delivery of results to third-party sites
  - ...
  - Relatively easy to set up and interface (at least with Java and Python)
  - Not yet integrated because of delays setting up Globus at MPE



# rosatMPAService.properties

## 1-Select the name of the data service resource.

dai.resource.id=rosatMPADAI

## 2-Select the type of the data resource that forms the data service

## resource.

dai.data.resource.type=Relational

## 3-Provide information about the data resource. This includes:

dai.product.name=SQL Server

dai.product.vendor=Microsoft

dai.product.version=

dai.data.resource.uri=jdbc:microsoft:sqlserver://130.183.85.140:1433;DatabaseName=ROSAT

dai.driver.class=com.microsoft.jdbc.sqlserver.SQLServerDriver

## 4-Enter the initial credential that users need to provide to

## access the data resource.

## If no credentials need to be provided then leave blank.

dai.credential=

## 5-Enter the database username and password that will be used to log

## into the database.

dai.user.name=xxx

dai.password=xxx

# SimpleSQLQueryExample.java

```
import org.w3c.dom.Document;
import uk.org.ogsadai.client.toolkit.GenericServiceFetcher;
import uk.org.ogsadai.client.toolkit.Response;
import uk.org.ogsadai.client.toolkit.activity.ActivityRequest;
import uk.org.ogsadai.client.toolkit.activity.sql.CSV;
import uk.org.ogsadai.client.toolkit.activity.sql.SQLQuery;
import uk.org.ogsadai.client.toolkit.activity.sql.WebRowSet;
import uk.org.ogsadai.client.toolkit.service.DataService;
/**
 * This example creates a Grid Data Service and performs a
 * simple SQL query. The results are returned within the
 * response document.
 */
public class SimpleSQLQueryExample {

    // Copyright statement
    private static final String COPYRIGHT_NOTICE =
        "(c) IBM Corp. 2002 - 2005. (c) The University of Edinburgh 2002 - 2005.";
    public static void main(String[] args) throws Exception {

        // set up service URL and resource ID
        String handle = "http://gavosrv1.xray.mpe.mpg.de:8080/wsrf/services/gavo/DataService";
        String id = "rosatMPADAI";

        // Locate a Data Service
        DataService service = GenericServiceFetcher.getInstance().getDataService(handle, id);
        System.out.println("Ready to connect to data service at " + service.getURL());

        // Perform a simple SQLQuery
        String sql = "select * from dbo.fgetphotonsfromrect( 45, -1, 46, 0) where energy_cor between .5 and 2. ";

        System.out.println("\nPerforming SQL query: " + sql);
        SQLQuery query = new SQLQuery(sql);
        CSV csv = new CSV(query.getOutput());
        ActivityRequest request = new ActivityRequest();
        request.add(query);
        request.add(csv);
        Response response = service.perform( request );
        System.out.println("Response:\n" + response.getAsString());
    }
}
```

# Logistics

- Regular directory structure is essential for automated workflow, but
  - is a lot of work to create and maintain
  - doesn't scale
  - structure of problem more table than tree
- → DMC?

# Workflow

- Compilation, tests, execution, postprocessing – all done with make
- make has issues. (ant is better.)
- → scons?
  - Combines a modern build tool
  - with a full-fledged programming language (python)

# Visualization

- IDL
  - Licensing issues
  - → GDL?
  - → Or a fortran-callable package?
- Not yet integrated in workflow

