Opal: Wrapping Scientific Applications as Web Services

Sriram Krishnan*, Brent Stearn, Karan Bhatia, Kim Baldridge,
Wilfred Li, Peter Arzberger
*sriram@sdsc.edu

Motivation

- Enable access to scientific applications on Grid resources
  - Seamlessly via a number of user interfaces
  - Easily from the perspective of a scientific user
- Enable the creation of complex scientific workflows
  - Possibly with the use of commodity workflow toolkits

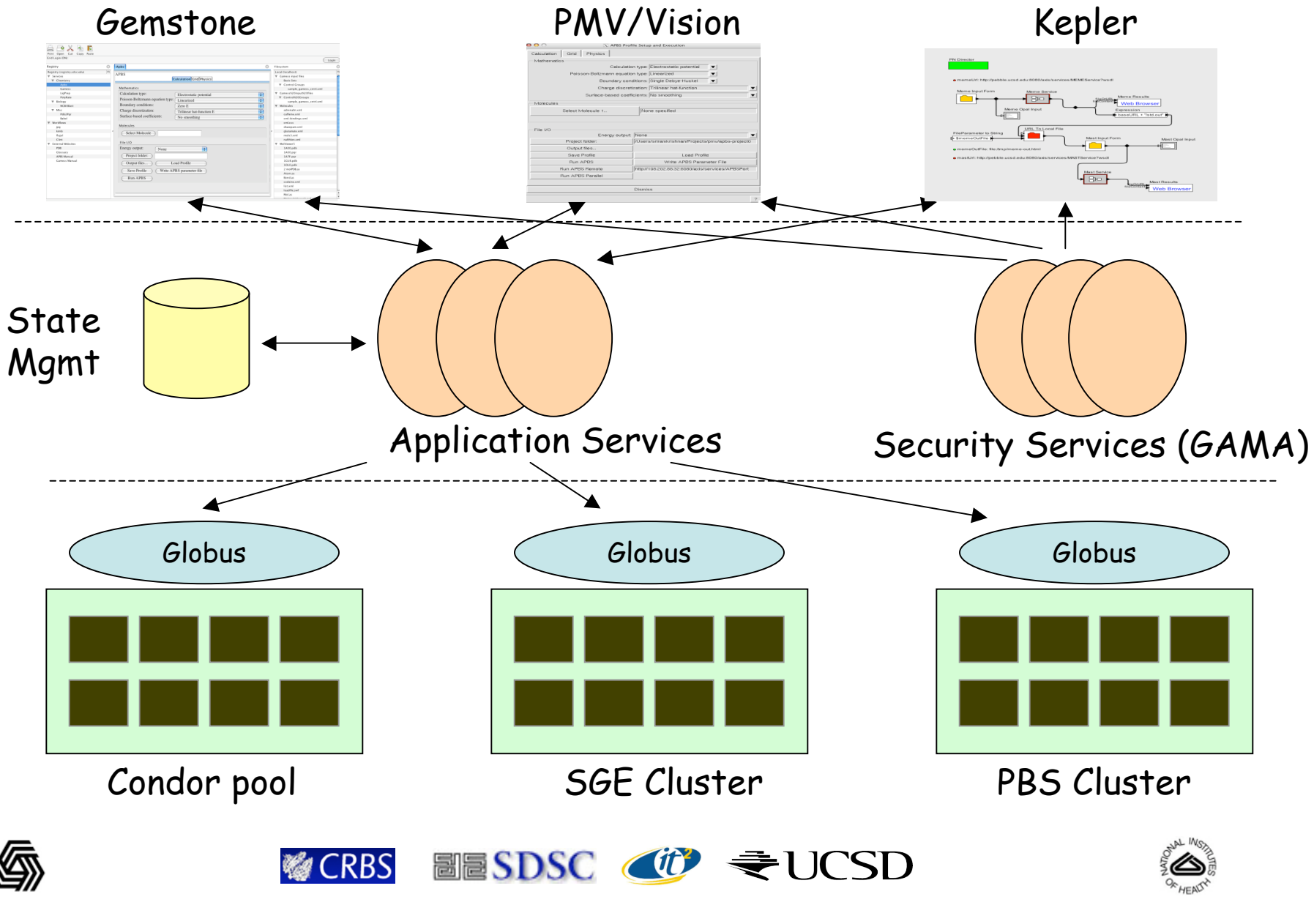
Some Problems

- Access to Grid resources is still very complicated
  - User account creation
  - Management of credentials
  - Installation and deployment of scientific software
  - Interaction with Grid schedulers
  - Data management

Towards Services Oriented Architectures (SOA)

- Scientific applications wrapped as Web services
  - Provision of a SOAP API for programmatic access
- Clients interact with application Web services, instead of Grid resources
  - Used in practice in NBCR, CAMERA, GLEON, among others

Big Picture


Gemstone
PMV/Vision
Kepler
State Mgmt
Application Services
Security Services (GAMA)
Globus
Globus
Globus
Condor pool
SGE Cluster
PBS Cluster

Scientific SOA: Benefits

- Applications are installed once, and used by all authorized users
  - No need to create accounts for all Grid users
  - Use of standards-based Grid security mechanisms
- Users are shielded from the complexities of Grid schedulers
- Data management for multiple concurrent job runs performed automatically by the Web service
- State management and persistence for long running jobs
- Accessibility via a multitude of clients

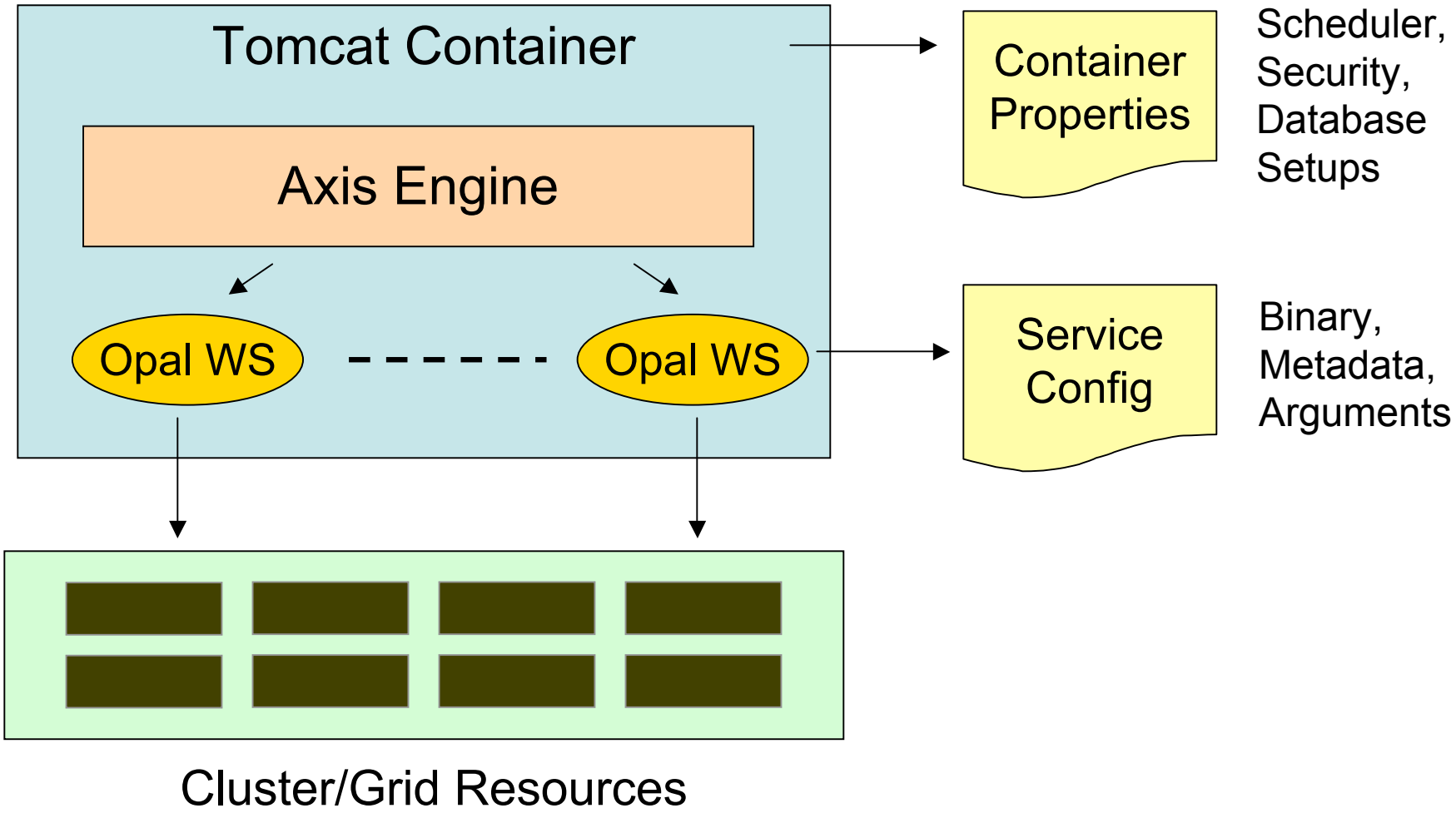
Possible Approaches

- Write application services by hand
  - Pros: More flexible implementations, stronger data typing via custom XML schemas
  - Cons: Not generic, need to write one wrapper per application
- Use a Web services wrapper toolkit, such as Opal
  - Pros: Generic, rapid deployment of new services
  - Cons: Less flexible implementation, weak data typing due to use of generic XML schemas

The Opal Toolkit: Overview

- Enables rapid deployment of scientific applications as Web services (< 2 hours)
- Steps
  - Application writers create configuration file(s) for a scientific application
  - Deploy the application as a Web service using Opal's simple deployment mechanism (via Apache Ant)
  - Users can now access this application as a Web service via a unique URL

Opal Architecture



Implementation Details

- Service implemented as a single Java class using Apache Axis
  - Application behavior specified by a configuration file
  - Configuration passed as a parameter inside the deployment descriptor (WSDD)
- Possible to have multiple instances of the same class for different applications
  - Distinguished by a unique URL for every application
- No need to generate sources or WSDL prior to deployment

Sample Application Configuration

```
<appConfig xmlns="http://nbcr.sdsc.edu/opal/types"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <metadata>
    <usage><![CDATA[psize.py [opts] <filename>]]></usage>
    <other xsd:type="xsd:string">
       <![CDATA[
         --help             : Display this text
         --CFAC=<value>     : Factor by which to expand mol dims to
                              get coarse grid dims
                              [default = 1.7]
       ...
       ]]>
    </other>
  </metadata>
  <binaryLocation>/homes/apbs_user/bin/psize.py</binaryLocation>
  <defaultArgs>--GMEMCEIL=1000</defaultArgs>
  <parallel>false</parallel>
</appConfig>
```

Sample Container Properties

```
# the base URL for the tomcat installation
# this is required since Java can't figure out the IP
# address if there are multiple network interfaces
tomcat.url=http://ws.nbcr.net:8080

# database information
database.use=false
database.url=jdbc:postgresql://localhost/app_db
database.user=<app_user>
database.passwd=<app_passwd>

# globus information
globus.use=true
globus.gatekeeper=ws.nbcr.net:2119/jobmanager-sge
globus.service_cert=/home/apbs_user/certs/apbs_service.cert.pem
globus.service_privkey=/home/apbs_user/certs/apbs_service.privkey

# parallel parameters
num.procs=16
mpi.run=/opt/mpich/gnu/bin/mpirun
```

Application Deployment

- Deployment Descriptor (WSDD):

```
<service name="PsizeServicePort" provider="java:RPC"
        style="document" use="literal">
     <parameter name="appConfig"
                value="/home/apbs_user/opal/etc/psize_config.xml"/>
     <parameter name="scope" value="Application"/>
     <parameter name="className"
                value="edu.sdsc.nbcr.opal.AppServiceImpl"/>
     ...
</service>
```

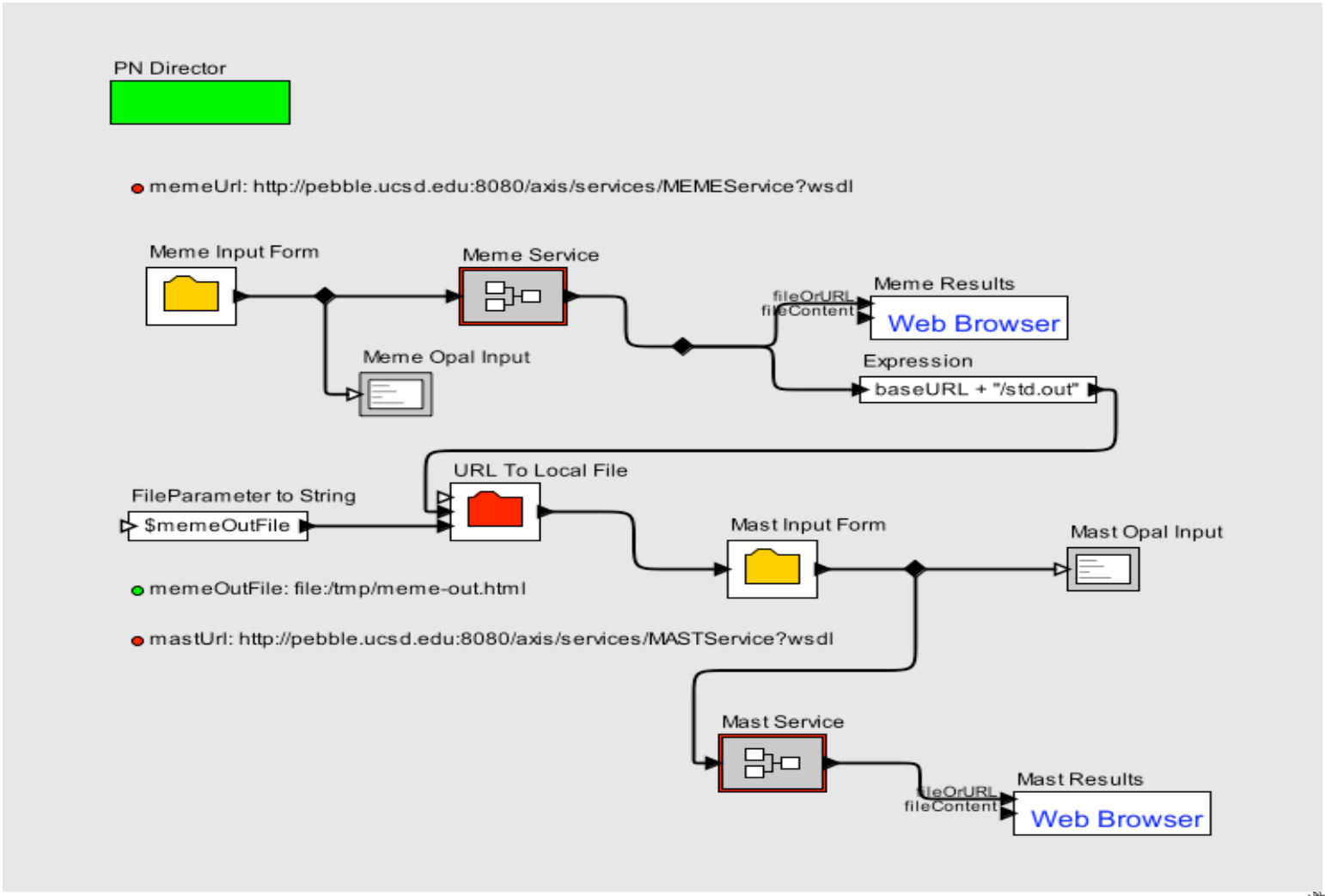
- To deploy onto a local Tomcat container:

```
ant -f build-opal.xml deploy -DdeployDesc=<deploy.wsdd>
```

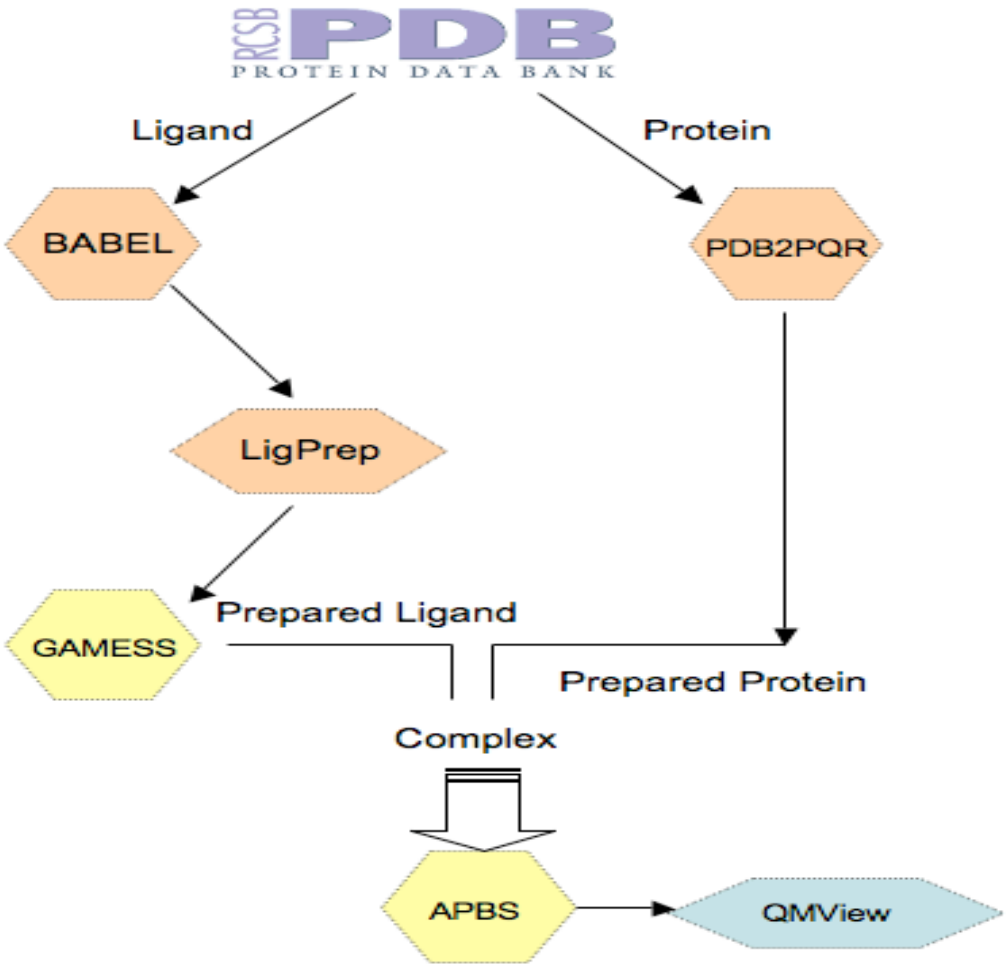
Service Operations

- Get application metadata: Returns metadata specified inside the application configuration
- Launch job: Accepts list of arguments and input files (Base64 encoded), launches the job, and returns a jobID
- Query job status: Returns status of running job using the jobID
- Get job outputs: Returns the locations of job outputs using the jobID
- Destroy job: Uses the jobID to destroy a running job

Kepler Workflow



Gemstone Access to Molecular Science



Future Work

- WSRF Integration
  - State management using standard Grid mechanisms
  - Asynchronous status notifications via WS-Notification
  - Better lifetime management for job I/O
    - Currently the job outputs reside on the server until they are deleted, typically by a cron job
- Alternate mechanisms for I/O staging
  - GridFTP, RFT, GASS
- Interface generation from application metadata
  - Presentation logic to describe interfaces
  - Business logic for Web service invocations from the interface

Concluding Remarks

- Opal enables rapidly exposing legacy applications as Web services
  - Provides features like Job management, Scheduling, Security, and Persistence
- More information, downloads, documentation:
  - http://nbcr.net/services/