Opal - Enabling the Web 2.0 Paradigm for Biomedical and Metagenomics Applications

Sriram Krishnan, Wes Goodman, Luca Clementi, Zhaohui Ding, Kohei Ichikawa, Shinji Shimojo, Susumu Date, Philip Papadopoulos, Larry Smarr, Paul Gilna, Peter Arzberger, Wilfred Li

Introduction

Opal is developed by NBCR (National Biomedical Computation Resource) as a Java-based toolkit that automatically wraps any scientific application with a Web services layer that is fully integrated with Grid Security Infrastructure (GSI) based security, cluster support, and data management (http://nbcr.net/services/opal/). It is designed to make the deployment of applications as Web services as simple as possible. This is critical because computational biologists should focus on more sophisticated biological modeling problems, instead of spending their time rewriting applications to take advantage of the emerging Grid computing technologies. With Opal, only a simple configuration file is required to deploy an existing application as a Web service, and the Grid middleware is completely transparent to the end-user. In addition, an Opal-based application service may be accessed using any Web services enabled client, because of the use of standard WSDL and SOAP-based APIs. Opal also provides the basic HTTP/S access to results, as well as any metadata that describes how the inputs and outputs may be handled. This 'deploy once, and used by many' feature of a Web service is a key ingredient in lowering the cost of entry to the Grid.

Web 2.0 Principles

- Use of the Web as a platform
- Harness collective intelligence of the community
- Focus on cost-effective scalable services, not packaged software
- Enable lightweight programming models
- Provide a rich user experience

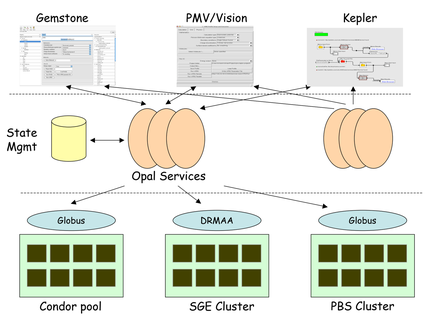
* Tim O'Reilly. "What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software". http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

Our Philosophy

- Focus on application-level services
- Provide access to virtualized resources
- Enable access via multiple user interfaces
- Leverage the right tools - open source, commodity software

* S. Krishnan and K. Bhatia. "SOAs for Scientific Applications: Experiences and Challenges". Accepted for publication to the 3rd IEEE International Conference on e-Science and Grid Computing, Dec 2007

Scientific SOA: Big Picture

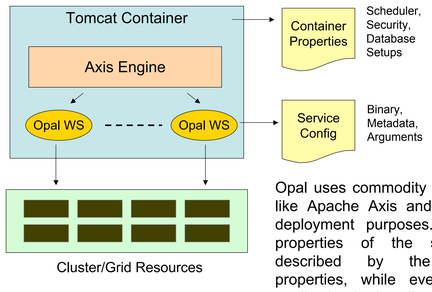


The multi-tier Services Oriented Architecture (SOA) consists of the raw resources at the bottom, the application and security services in the middle tier, and the various user interfaces on top, which shield the end-users from the complexity of the underlying infrastructure.

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



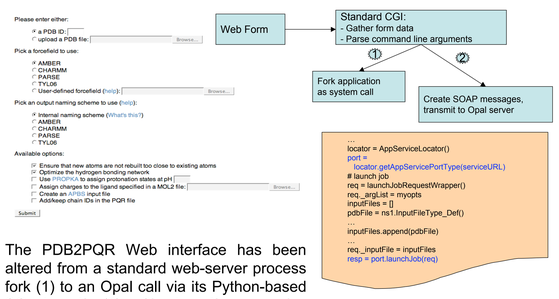
Opal uses commodity technologies like Apache Axis and Tomcat for deployment purposes. The static properties of the system are described by the container properties, while every scientific application is described by an XML service configuration.

Deploying an Application: PDB2PQR

```
<appConfig xmlns="http://nbcr.sdsc.edu/opal/types"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <metadata>
    <usage><![CDATA[python pdb2pqr.py [options] --ff={forcefield} {path} {output-path} ]]></usage>
    <info xsd:type="xsd:string">
     <![CDATA[
        The required arguments are as follows:
        <forcefield>
          The forcefield to use -- currently AMBER (AMBER99, Wang J, et al, 2000), CHARMM
          (CHARMM27, MacKerell AD Jr, et al, 1998), PARSE (PARSE, Sitkoff D, et al, 1994),
          and TYL06 (Tan C, et al, 2006) are supported.
              ...
        Optional command-line arguments are:
        --help (-h)
          Display the usage information
              ...
     ]]>
    </info>
  </metadata>
  <binaryLocation>/usr/local/src/apbs-0.3.1/tools/manip/psize.py</binaryLocation>
  <parallel>false</parallel>
</appConfig>
```

For each deployed application, Opal expects an XML configuration file. This defines application metadata, such as correct argument syntax, location for the application's binary, and the application's parallelization.

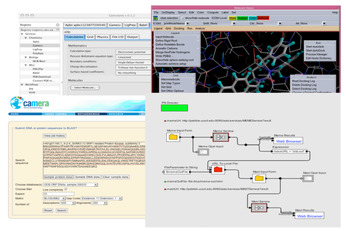
PDB2PQR: Opal Web Interface



The PDB2PQR Web interface has been altered from a standard web-server process fork (1) to an Opal call via its Python-based SOAP API (2). Client APIs are also available in other languages such as Perl and Java, and can be easily incorporated into existing Web interfaces.

Graphical User Interfaces

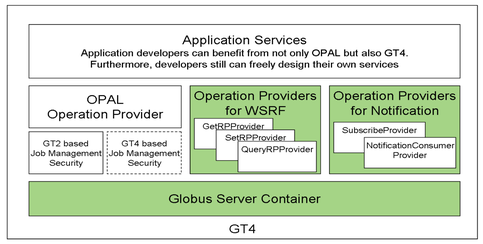
Gemstone: http://gemstone.mozdev.org ADT: http://mgltools.scripps.edu



GridSphere: http://www.gridsphere.org Kepler: http://kepler-project.org

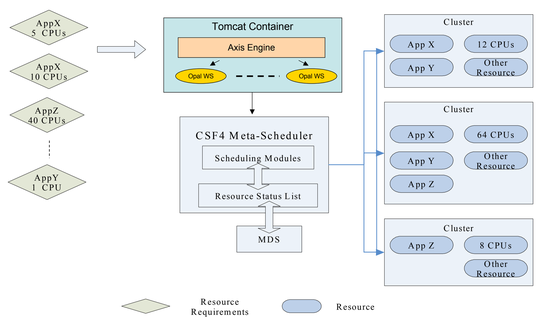
Several graphical interfaces written in a wide variety of languages are available for scientific applications deployed via Opal. Gemstone is a Mozilla Firefox based Rich Internet Application (RIA). AutoDockTools (ADT) provides a Python-based rich interface for protein docking. GridSphere is a portal-based environment used by the NBCR and CAMERA projects for providing Web interfaces to scientific applications, while Kepler is a workflow tool for composition of complex scientific pipelines.

WSRF Compliance: Opal-OP



Opal-OP, developed by Osaka University, Japan, is an extension of the base Opal toolkit for WSRF compatibility. Opal-OP services can reside within a GT4 container and leverage all of GT4 functionality, while being accessible via WSRF primitives.

Ongoing Work: Meta-Scheduling



The CSF4 Meta-Scheduler provides a mechanism to transparently schedule jobs across multiple Grid resources. As part of our ongoing work, we are working on leveraging CSF4-based Meta-scheduling to launch Opal jobs. The end-user would still use the simple Opal APIs to submit jobs, while being completely oblivious of the Meta-scheduling being implemented in the background.

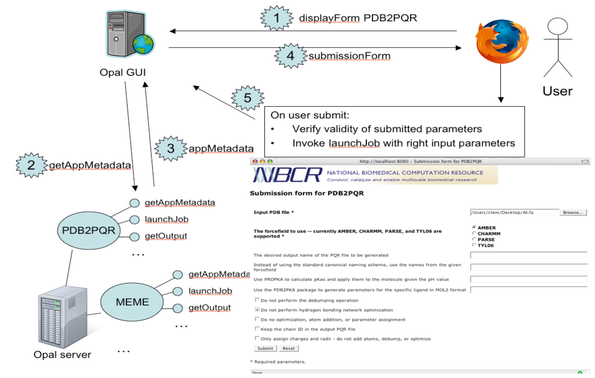
Argument Validation

```
Usage: pdb2pqr.py [options] --ff=<forcefield> <path>
<output-path>
Required Arguments:
    <forcefield> : The forcefield to use - currently
                   amber, charmm, and parse are
supported.
    <path>       : The path to the PDB file or an ID
                   to obtain from the PDB archive
    <output-path>: The desired output name of the
                   PQR file to be generated
Optional Arguments:
    --help  (-h): Display the usage information
    .....
Optional Arguments from Extensions Directory:
    --chi        : Print the per-residue backbone chi
                   angle to {output-path}.chi
.....
```

```
<!-- list of flags -->
<flags>
  <flag>
    <id>help</id>
    <tag>--help</tag>
  </flag>
  .....
</flags>
<!-- list of tagged parameters -->
<taggedParams>
  <separator>=</separator>
  <param>
    <id>forcefield</id>
    <tag>--ff</tag>
    <paramType>STRING</paramType>
    <required>true</required>
    <value>AMBER</value>
    ....
  </param>
  ....
</taggedParams>
<!-- list of untagged parameters, in order -->
<untaggedParams>
  <param>
    <id>pdb</id>
    <paramType>STRING</paramType>
    <ioType>INPUT</ioType>
    <required>true</required>
  </param>
  .....
</untaggedParams>
```

Work is being done to add argument validation to Opal via an XML parameter specification. Application developers can specify the structure of command-line arguments, and the Opal service will validate arguments before a job is launched. Errors can be caught before the job is queued on the Grid, resulting in less time wasted when an application call is not properly constructed. The parameter specification can also be automatically translated into interfaces.

Automatic Interface Generation



We are working on automatic interface generation from the XML parameter specification described above. When a user wishes to invoke an Opal service, the Web interface dynamically downloads the parameter specification and renders it via a Web form. This enables an application provider to make updates dynamically, and push it instantaneously to the end-user without having to install any new software.

Conclusions

- More information, downloads, documentation:
  - http://nbcr.net/services/
  - Opal Tutorial on September 26, 4PM at NCSA 1030
- Questions and comments welcome!
  - Email Sriram at: sriram@sdsc.edu
  - Web Services Forum at: https://nbcr.net/forum/