# DRP: Dynamic Resource Provisioning

*Ioan Raicu, Ian Foster*

## 1. Overview

This short report on dynamic resource provisioning is in part the result of the CEDPS meeting on October 24th-25th by the Scalable Services breakout session where the following people were present: Ian Foster, Carl Kesselman, Kate Keahey, Keith Jackson, David Konerding, Tim Freeman, Ravi Madduri, and Ioan Raicu (if I missed anyone, please feel free to add them).

After discussing three different systems (AstroPortal [1], GridFTP [2], and Condor Brick [3]) that all needed to do dynamic resource provisioning, we came to the conclusion that there are several things in common across all these three systems, and that other feature are desirable:

- Typical system components and other related components:
  - User(s) to produce work
  - Work Dispatcher
  - Resource Monitor (observes state information at the work dispatcher)
  - Resource Manager (resource provisioner based on information from Resource Monitor)
  - Worker Starter (used to bootstrap worker code by the Resource Manager)
- Coarse grained resource provisioning (i.e. GRAM, etc); typical resources allocated for hours to days
- Fine grained work dispatching (i.e. WS calls, Condor glidein, proprietary protocol based on UDP/TCP/IP, etc); typical work completion in seconds to minutes
- Variable resource pool size; increasing/decrease the number of allocated resources based on load and/or other metrics
- It would be desirable to have a generic system that could be customizable per application to provide the dynamic resource provisioning; the customization needed should be kept to a minimum, and mostly involve only configuration of the resource monitor (which state information to monitor), and the configuration of the resource manager (what policies for resource provisioning it should use depending on the resource monitor observations).

The main motivations behind having the capability for dynamic resource provisioning are:

- Allows for finer grained resource management, including the control of priorities and usage policies
- Optimize for the grid user's perspective: reduces delays on per job scheduling by utilizing pre-reserved resources
- If a metric of success for the Grid VO is resource utilization, then using a DRP will likely increase the resource utilization; note that the VO does not know of the actual state of the allocated resources (idel vs busy), so once they have been allocated by a user, they are by all practical means busy
- Opens the possibility to customize the resource scheduler per application basis, including the use of both data resource management and compute resource management information for more efficient scheduling
- Reduced complexity to the application developer as the details of the dynamic resource provisioning are abstracted away behind a web service interface

This document has the following organization:

1) an overview of the requirements of a dynamic resource provisioning system
2) a description of the proposed architecture DRP: Dynamic Resource Provisioning (if anyone else has a better name and/or acronym, please feel free to offer suggestions)
3) a specification of the DRP Web Service interface
4) some details on possible implementations
5) usage scenarios
6) conclusions we can draw about dynamic resource provisioning

## 2. DRP Architecture

This section covers the description of the proposed architecture DRP: Dynamic Resource Provisioning. Figure 1 has a pictorial representation of DRP; similarly as we outlined in the overview section, the basic architectural components are with more concrete functionality as seen in Figure 1:

- **User:** User(s) to produce work
- **DRP Utilizing Web Service, Work Dispatch:** Work Dispatcher
- **DRP Web Service, Resource Monitor:** observes state information at the work dispatcher via Resource Properties exposed by the DRP Utilizing Web Service
- **DRP Web Service, Resource Manager:** Resource Provision resource via GRAM based on information from the Resource Monitor
- **Resource Pool, Worker Starter:** used to bootstrap worker code by the Resource Manager via GRAM

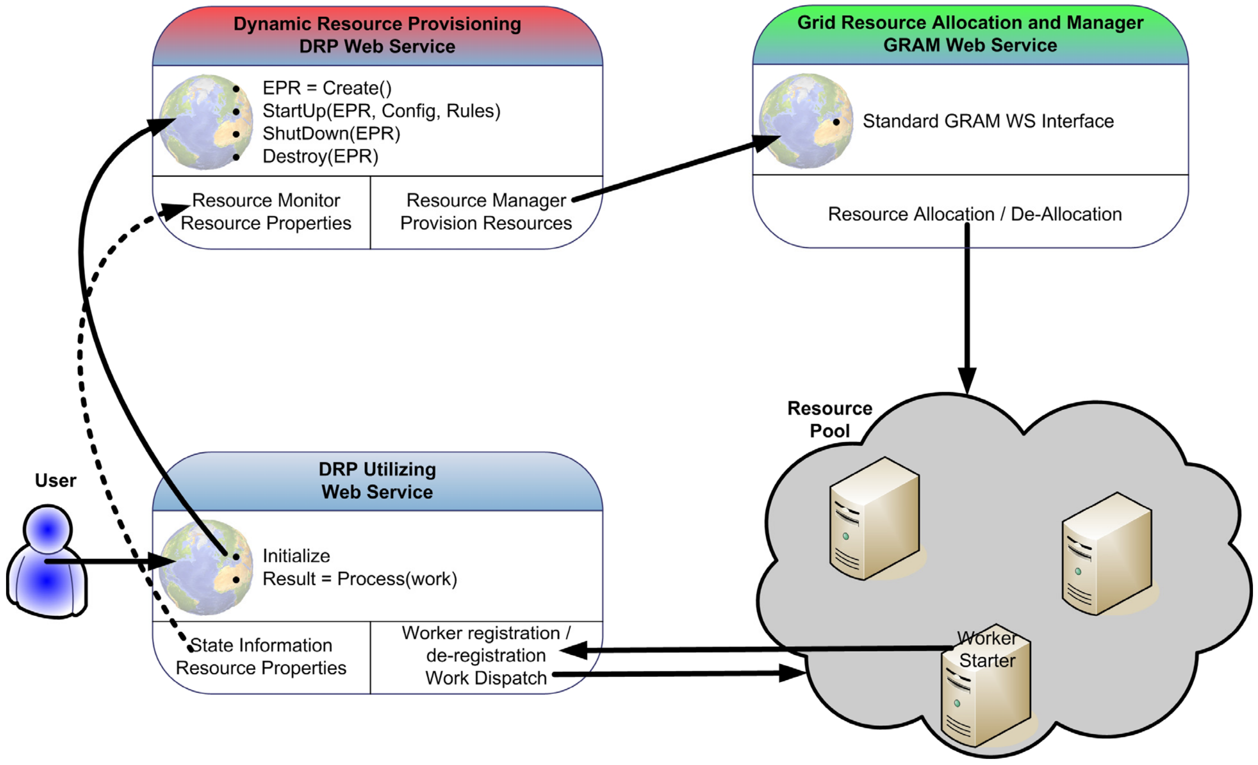

**Figure 1: DRP: Dynamic Resource Provisioning Architecture**

It is worthwhile to note that there is only one core component to DRP, specifically the DRP Web Service (DRP WS). The DRP WS implements the necessary interface (see the next section for more details) for arbitrary web services to be able to take advantage of dynamic resource provisioning mechanisms without exposing all the details if the arbitrary web service wanted to implement the same functionality from scratch. To make the DRP WS functional, there are two more pieces that must be in place: the worker starter (which could be absorbed by the worker code which is a component of the arbitrary web service) and the state information that is being exposed for the DRP WS Resource Monitor to read. Therefore, if an arbitrary web service has the worker code implemented that can execute on a remote resource, then the only things that need to be modified to the arbitrary web service are:

- Initialize the DRP WS with the appropriate information necessary for the DRP WS to make the appropriate resource provisioning decisions; see the next section on what this information is
- Expose state information via Resource Properties

## 3. DRP Web Service Interface

The DRP WS Interface has been outlined in Figure 2.

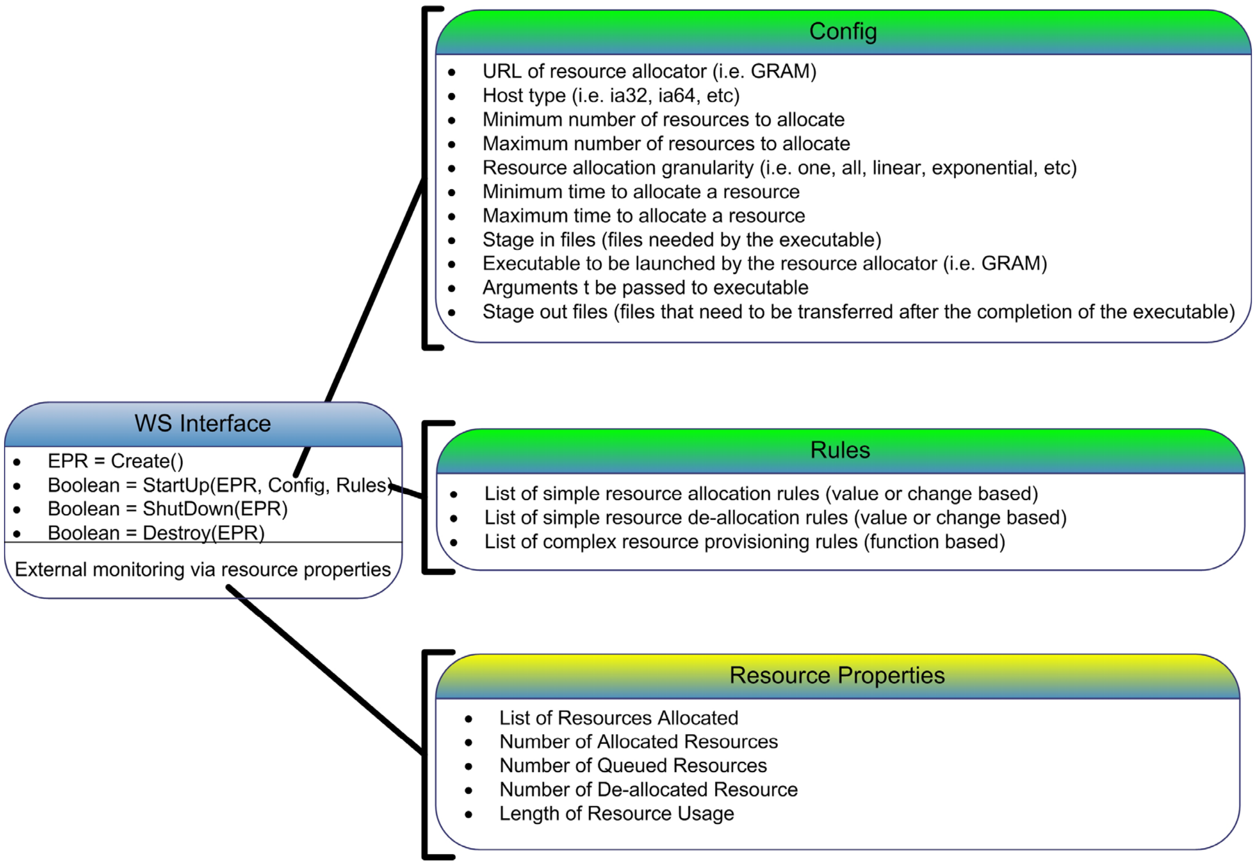
Config
URL of resource allocator (i.e. GRAM)
Host type (i.e. ia32, ia64, etc)
Minimum number of resources to allocate
Maximum number of resources to allocate
Resource allocation granularity (i.e. one, all, linear, exponential, etc)
Minimum time to allocate a resource
Maximum time to allocate a resource
Stage in files (files needed by the executable)
Executable to be launched by the resource allocator (i.e. GRAM)
Arguments t be passed to executable
Stage out files (files that need to be transferred after the completion of the executable)
WS Interface
EPR = Create()
Boolean = StartUp(EPR, Config, Rules)
Boolean = ShutDown(EPR)
Boolean = Destroy(EPR)
External monitoring via resource properties
Rules
List of simple resource allocation rules (value or change based)
List of simple resource de-allocation rules (value or change based)
List of complex resource provisioning rules (function based)
Resource Properties
List of Resources Allocated
Number of Allocated Resources
Number of Queued Resources
Number of De-allocated Resource
Length of Resource Usage

**Figure 2: DRP Web Service Interface**

The full WSDL definitions have been included in the following 4 pages. There are two WSDL files, one denoting the definition of the DRP_Factory while the other containing the definition of the DRP_Service.

The DRP_Factory contains the create() interface, which returns back an EPR specific to the newly created resource. The EPR can be used later for the rest of the WS calls to both the factory and the service. The DRP_Factory also contains the Destroy() interface, which can be used to destroy the specified resource, which completely relinquishes all allocated resources, both remote and local pertaining to the specified EPR.

The DRP_Service contains the StartUp() interface, which is primarily used to initialize the DRP resource so it can be ready to perform the resource provisioning. The arguments to the StartUp() call are the EPR (denoting the specific DRP resource to use, the Config object (shown in the Config section in Figure 2), and the Rules object (shown in the Rules section in Figure 2). It might be worth elaborating more on the Rules object, and how rules can be easily derived, expressed, and effectively represented so that the desired resource provisioning takes place based on the observed state information. For administrative purposes, or even for the DRP utilizing web service to gain knowledge about the resource provisioning state, several resource properties should be exposed by the DRP WS, as shown in the Resource Properties in Figure 2.

Please see the following pages for the specific WSDL definitions to help clarify the DRP WS Interface, the Config and Rules objects, and the resource properties exposed.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="DRP Factory"
    targetNamespace="http://www.globus.org/namespaces/DRP/DRP Factory"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.globus.org/namespaces/DRP/DRP Factory"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!--============================================================

                          T Y P E S

      ============================================================-->
    <types>
        <xsd:schema targetNamespace="http://www.globus.org/namespaces/DRP/DRP Factory"
            xmlns:tns="http://www.globus.org/namespaces/DRP/DRP Factory"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import
                namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
                schemaLocation="../../ws/addressing/WS-Addressing.xsd" />
            <!-- REQUESTS AND RESPONSES -->

            <xsd:element name="createResource">
                <xsd:complexType/>
            </xsd:element>
            <xsd:element name="createResourceResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element ref="wsa:EndpointReference"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>

        </xsd:schema>
    </types>
    <!--============================================================

                          M E S S A G E S

      ============================================================-->
    <message name="CreateResourceRequest">
        <part name="request" element="tns:createResource"/>
    </message>
    <message name="CreateResourceResponse">
        <part name="response" element="tns:createResourceResponse"/>
    </message>
    <!--============================================================

                          P O R T T Y P E

      ============================================================-->
    <portType name="FactoryPortType">
        <operation name="createResource">
            <input message="tns:CreateResourceRequest"/>
            <output message="tns:CreateResourceResponse"/>
        </operation>
    </portType>
</definitions>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="APService"
    targetNamespace="http://www.globus.org/namespaces/DRP/DRP Service"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.globus.org/namespaces/DRP/DRP Service"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrlw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
    xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:import
        namespace=
        "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
        location="../../wsrf/properties/WS-ResourceProperties.wsdl" />
    <wsdl:import
        namespace=
        "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
        location="../../wsrf/lifetime/WS-ResourceLifetime.wsdl" />
    <types>
        <xsd:schema targetNamespace="http://www.globus.org/namespaces/DRP/DRP Service"
            xmlns:tns="http://www.globus.org/namespaces/DRP/DRP Service"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <!-- Requests and responses -->
            <!-- determined number of separate allocation calls (i.e. GRAM calls) to get the desired number of resources -->
            <xsd:complexType name="Granularity">
                <xsd:sequence>
                    <!-- to allocate 16 resources, it would allocate them 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1  in each allocation call until all desired resources have been allocated-->
                    <xsd:element name="one" type="xsd:boolean"/>
                    <!-- to allocate 16 resources, it would allocate them 16 in each allocation call until all desired resources have been allocated-->
                    <xsd:element name="all" type="xsd:boolean"/>
                    <!-- to allocate 16 resources, it would allocate them 1, 2, 3, 4, 5, 1  in each allocation call until all desired resources have been allocated-->
                    <xsd:element name="linear" type="xsd:boolean"/>
                    <!-- to allocate 16 resources, it would allocate them 1, 2, 4, 8, 1  in each allocation call until all desired resources have been allocated-->
                    <xsd:element name="exponential" type="xsd:boolean"/>
                </xsd:sequence>
            </xsd:complexType>
            <!-- configuration of the values needed to communicate with the resource allocator (i.e. GRAM), as well as setting upper and lower bounds on the number of resources and time to allocate the resources -->
            <xsd:complexType name="Config">
                <xsd:sequence>
                    <xsd:element name="gramURL" type="xsd:string"/>
                    <xsd:element name="hostType" type="xsd:string"/>
                    <xsd:element name="minNumResourceAllocate" type="xsd:int"/>
                    <xsd:element name="maxNumResourceAllocate" type="xsd:int"/>
                    <xsd:element name="resourceAllocateGranularity" type="tns:Granularity"/>
                    <xsd:element name="minTimeResourceAllocate" type="xsd:int"/>
                    <xsd:element name="maxTimeResourceAllocate" type="xsd:int"/>
                    <xsd:element name="stageIn" type="xsd:string"/>
                    <xsd:element name="executable" type="xsd:string"/>
                    <xsd:element name="arguments" type="xsd:string"
                     minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="stageOut" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <!-- needed in the SimpleRuleAllocate to determine what operation to apply to the MarkValue; one element from below must be true, whie the rest must be false  -->
            <xsd:complexType name="Operand">
                <xsd:sequence>
                    <xsd:element name="lessThan" type="xsd:boolean"/>
                    <xsd:element name="lessThanOrEqual" type="xsd:boolean"/>
                    <xsd:element name="equal" type="xsd:boolean"/>
                    <xsd:element name="notEqual" type="xsd:boolean"/>
                    <xsd:element name="greaterThanOrEqual" type="xsd:boolean"/>
                    <xsd:element name="greaterThan" type="xsd:boolean"/>
                    <xsd:element name="difference" type="xsd:boolean"/>
                </xsd:sequence>
            </xsd:complexType>
            <!-- information relevant to access a resource property from a given service -->
            <xsd:complexType name="RPinfo">
                <xsd:sequence>
                    <xsd:element name="serviceURL" type="xsd:string"/>
                    <xsd:element name="pollTime" type="xsd:int"/>
                    <xsd:element name="notificationUse" type="xsd:boolean"/>
                    <xsd:element name="name" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <!-- tests between the markValue and the actual value of the resource property identified by the name using the markType operand  -->
            <xsd:complexType name="SimpleRule">
                <xsd:sequence>
                    <xsd:element name="RP" type="tns:RPinfo"/>
                    <xsd:element name="markValue" type="xsd:double"/>
                    <xsd:element name="markType" type="tns:Operand"/>
                </xsd:sequence>
            </xsd:complexType>
            <!-- Complex function support (i.e. y = x^2 + 3*x + 10); this feature needs to be defined -->
            <xsd:complexType name="ComplexFunction">
                <xsd:sequence>
                    <xsd:element name="undefined" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <!-- Complex Rule Allocate: contains a list of resource property names that can be used in the ComplexFunction to determine the amount of resources to allocate  -->
            <xsd:complexType name="complexResourceProvision">
                <xsd:sequence>
                    <xsd:element name="RPs" type="tns:RPinfo"
                     minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="function" type="tns:ComplexFunction"/>
                </xsd:sequence>
            </xsd:complexType>
            <!-- the collection of rules needed for resource provisioning  -->
            <xsd:complexType name="Rules">
                <xsd:sequence>
                    <xsd:element name="simpleResourceAllocate" type="tns:SimpleRule"
                     minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="simpleResourceDeAllocate" type="tns:SimpleRule"
                     minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="complexResourceProvision" type="tns:ComplexRule"
                     minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:complexType>
            <!-- the object arguement to the startUp() WS call which starts up the DRP core logic for resource provisioning  -->
            <xsd:element name="StartUp">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="config" type="tns:Config"/>
                        <xsd:element name="rules" type="tns:Rules"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <!-- the object response for the startUp() WS call return true if everything started OK, or false if there were any errors (i.e. invalid credentials, incorrect GRAM URL, non-existent executable, etc...)  -->
            <xsd:element name="StartUpResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="valid" type="xsd:boolean"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <!-- the object arguement to the shutDown() WS call which stops the DRP core logic (i.e. de-allocate any allocated resources, etc...)  -->
            <xsd:element name="ShutDown">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="valid" type="xsd:boolean"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <!-- the object response for the shutDown() WS call return true if everything stoped OK, or false if there were any errors (i.e. resources that were not de-allocated, etc...)  -->
            <xsd:element name="ShutDownResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="valid" type="xsd:boolean"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <!-- Resource properties -->
            <!-- Resource description; could include physical resource description as well (i.e. CPU type, CPU count, CPU speed, memory size, NIC speed, HD capacity, HD available, etc...) -->
            <xsd:element name="ResourceDescription">
                <xsd:complexType>
                    <xsd:sequence>
```

```
                        <xsd:element name="name" type="xsd:string"/>
                        <xsd:element name="address" type="xsd:string"/>
                        <xsd:element name="type" type="xsd:string"/>
                        <xsd:element name="timeAllocated" type="xsd:string"/>
                        <xsd:element name="timeExpire" type="xsd:string"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="AllocatedResources">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="array" type="tns:ResourceDescription"
                         minOccurs="0" maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="NumAllocatedResources" type="xsd:int"/>
            <xsd:element name="NumQueuedResources" type="xsd:int"/>
            <xsd:element name="NumDeAllocatedResources" type="xsd:int"/>
            <xsd:element name="LengthResourceUsage" type="xsd:int"/>
            <xsd:element name="APResourceProperties">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element ref="tns:AllocatedResources" minOccurs="1" maxOccurs="1"/>
                        <xsd:element ref="tns:NumAllocatedResources" minOccurs="1" maxOccurs="1"/>
                        <xsd:element ref="tns:NumQueuedResources" minOccurs="1" maxOccurs="1"/>
                        <xsd:element ref="tns:NumDeAllocatedResources" minOccurs="1" maxOccurs="1"/>
                        <xsd:element ref="tns:LengthResourceUsage" minOccurs="1" maxOccurs="1"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    </types>
    <message name="StartUpInputMessage">
        <part name="parameters" element="tns:StartUp"/>
    </message>
    <message name="StartUpOutputMessage">
        <part name="parameters" element="tns:StartUpResponse"/>
    </message>
    <message name="ShutDownInputMessage">
        <part name="parameters" element="tns:ShutDown"/>
    </message>
    <message name="ShutDownOutputMessage">
        <part name="parameters" element="tns:ShutDownResponse"/>
    </message>
    <portType name="APPortType"
        wsdlpp:extends="wsrpw:GetResourceProperty
                    wsrlw:ImmediateResourceTermination
                    wsrlw:ScheduledResourceTermination"
        wsrp:ResourceProperties="tns:APResourceProperties">
        <operation name="startUp">
            <input message="tns:StartUpInputMessage"/>
            <output message="tns:StartUpOutputMessage"/>
        </operation>
        <operation name="shutDown">
            <input message="tns:ShutDownInputMessage"/>
            <output message="tns:ShutDownOutputMessage"/>
        </operation>
    </portType>
</definitions>
```

## 4. Usage Scenario

If needed, this can be filled in with how a specific application would use the DRP WS.

## 5. Implementation Details

Once the DRP WS architecture and interface is accepted, we could start talking about actual implementation details (i.e. programming language, the use of notifications or polling, persistent state information (flat files or databases), transport level security, message level security, or no security, etc).

## 6. Conclusion

There is not much to say here yet, other than the fact that the consensus is that this kind of functionality (i.e. DRP) is needed! In the overview section, we covered the motivation for defining such a DRP service; to re-iterate, they were:

- Allows for finer grained resource management, including the control of priorities and usage policies
- Optimize for the grid user's perspective: reduces delays on per job scheduling by utilizing pre-reserved resources
- If a metric of success for the Grid VO is resource utilization, then using a DRP will likely increase the resource utilization; note that the VO does not know of the actual state of the allocated resources (idel vs busy), so once they have been allocated by a user, they are by all practical means busy
- Opens the possibility to customize the resource scheduler per application basis, including the use of both data resource management and compute resource management information for more efficient scheduling
- Reduced complexity to the application developer as the details of the dynamic resource provisioning are abstracted away behind a web service interface

To be fair, we should also include some of the disadvantages that might be packaged together along with such functionality as DRP; note that theses disadvantages are mostly due from the desired functionality, and not any particular implementation of the DRP:

- All jobs submitted by different members need to map to the same user (this might be an implementation detail, but currently in all the existing prototypes of a DRP like functionality [1, 2, 3], they all map all the users to the same user account from the Grid VO's perspective); adequate logging and accounting at the DRP WS could probably give the Grid VO all the information they need for accounting purposes as long as the grid VO would trust the DRP WS
- Initial startup overhead
- Work could be halted unfinished when the original time lease on a particular resource expires if the time lease not being exposed to the work dispatcher
- From the Grid VO's perspective, if they were to actually look at the allocated resource utilization, it is likely that resources allocated via DRP like functionality would be under utilized, as allocated resources could be idle for long periods of time, yet be unavailable to be reclaimed back into the general pool of available resources; this could be resolved by the user of the DRP functionality properly setting the resource provisioning policy to release idle resources, where idle can be defined by the particular application; the problem with this is that we are then trading off between the resource responsiveness and actual resource utilization, but at least allowing the user to make this trade-off is probably the correct approach

## 7. References

There are probably more relevant references than these, but these are a good starting point as they were the center of our discussion at the Scalable Services break-out session.

[1] Ioan Raicu, Ian Foster, Alex Szalay. "Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets", SuperComputing 2006.

[2] John Bresnahan, Ian Foster. "An Architecture for Scavenging Compute Cluster Bandwidth," MS Thesis, University of Chicago, Department of Computer Science, 2006.

[3] Gaurang Mehta, Carl Kesselman, Ewa Deelman. "Dynamic Deployment of VO-specific Schedulers on Managed Resources," USC Information Sciences Institute, 2006.