Java WS Core for Developers

Rachana Ananthakrishnan

Jarek Gawor

Session Notes

- Slides available at:
  - http://www.mcs.anl.gov/~gawor/gw
- This session is for developers already familiar with Java WS Core
- Beginners please checkout 'L3: Build a Service Using GT4' lab
  - Thursday 2pm – 5:45pm
- Other relevant sessions at GW
  - COMM12: Mini Symposium - Development Tools for GT4 Service Programming
    - Monday - but slides might be interesting
  - L4: The FileBuy Globus Based Resource Brokering System - A Practical Example
    - Friday 9am - 1pm

Overview

- Two session parts
  1. General programming guidelines
     1. WSDL
     2. Service implementation
     3. Lifecycle management
     4. Resource persistence and caching
     5. Service communication
     6. Background tasks
     7. Debugging and production tuning
  2. Security features of Java WS Core

Java WS Core

- Development kit for building stateful Web Services
- Implementation of WS-Resource Framework (WSRF) and WS-Notification (WSN) family of specifications
- Provides lightweight hosting environment
  - Can also run in Tomcat, JBoss and other application servers
- Support for transport and message level security
- Implemented with 'standard' Apache software
  - Axis 1 (SOAP engine)
  - Addressing (WS-Addressing implementation)
  - WSS4J (WS-Security implementation)
  - and more

Java WS Core
Key Programming Model Concepts

- Service
  - Implements business logic – stateless
  - Can be composed of one or more reusable Java objects called **operation providers**
  - Configured via *server-config.wsdd*
- Resource
  - Represents the state - statefull
- ResourceHome
  - Manages a set of resources
  - Performs operations on a subset of resources at once
  - Configured via *jndi-config.xml*
- A service is usually configured with a corresponding *ResourceHome* that is used to locate the *Resource* objects

Programming Guidelines and Best Practices

Service WSDL

- Do not generate WSDL from existing code
  - Create it by hand, modify existing one, etc. but follow the WSDL guidelines described next
- Tooling is still not perfect
  - Might generate non-interoperable WSDL

WSDL Guidelines

- WSDL has
  - *Document* and *RPC* invocation style
  - *Literal* and *SOAP encoded* mode
- Use **Document/Literal** mode
  - Do not mix *Literal* with *SOAP encoding* in one WSDL
- Always validate your WSDL
  - Java WS Core does NOT validate it
- Follow WS-I Basic Profile 1.1 guidelines
  - Improves interoperability

WSDL Doc/Lit Guidelines

```
<wsdl:message name="AddRequest">
  <wsdl:part name="input" element="tns:AddRequest"/>
</wsdl:message>

<wsdl:message name="SubtractRequest">
  <wsdl:part name="input" element="tns:SubtractRequest"/>
</wsdl:message>


<portType name="CounterPT">
  <operation name="add">
    <input message="AddRequest"/>
    <output message="AddResponse"/>
</operation>
 <operation name="subtract">
    <input message="SubtractRequest"/>
    <output message="SubtractResponse"/>
  </operation>
</portType>
```

At most one **wsdl:part** element

WSDL Doc/Lit Guidelines

```
<wsdl:message name="AddRequest">
  <wsdl:part name="input" element="tns:AddRequest"/>
</wsdl:message>

<wsdl:message name="SubtractRequest">
  <wsdl:part name="input" element="tns:SubtractRequest"/>
</wsdl:message>


<portType name="CounterPT">
  <operation name="add">
    <input message="AddRequest"/>
    <output message="AddResponse"/>
</operation>
 <operation name="subtract">
    <input message="SubtractRequest"/>
    <output message="SubtractResponse"/>
  </operation>
</portType>
```

Must use **element** attribute

WSDL Doc/Lit Guidelines

```
<wsdl:message name="AddRequest">
  <wsdl:part name="input" element="tns:AddRequest"/>
</wsdl:message>

<wsdl:message name="SubtractRequest">
  <wsdl:part name="input" element="tns:SubtractRequest"/>
</wsdl:message>


<portType name="CounterPT">
  <operation name="add">
    <input message="AddRequest"/>
    <output message="AddResponse"/>
</operation>
 <operation name="subtract">
    <input message="SubtractRequest"/>
    <output message="SubtractResponse"/>
  </operation>
</portType>
```

Must reference **unique** elements (for input messages)

Document/Literal - Arrays

- Encoded - SOAP Encoding 


```
<xsd:complexType name="MyArray2Type" >
  <xsd:complexContent>
    <xsd:restriction base="soapenc:Array">
      <xsd:sequence>
        <xsd:element name="x" type="xsd:string"
                     minOccurs="0"  maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute ref="soapenc:arrayType"
                     wsdl:arrayType="tns:MyArray2Type[]"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

- Literal – XML Schema 


```
<xsd:complexType name="MyArray1Type">
  <xsd:sequence>
    <xsd:element name="x" type="xsd:string"
                 minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Service Implementation

- If you have an existing service code
  - Do NOT generate WSDL from it and try to make it work somehow
  - Instead:
    1) Create WSDL by hand (or using some tools)
    2) Validate WSDL
    3) Generate Java code from WSDL
    4) Implement the generated service interface by **delegating** the calls to your existing service code
- In general, always implement the generated service interface
  - Do NOT define your own service methods first
  - In Document/Literal mode service methods will ALWAYS have 1 input parameter

Service Implementation Guidelines

- Service methods should be stateless
- Keep service logic separate from the service façade
  - Use Axis generated types only in the service facade
    - Avoid passing it to other classes, etc.
    - Instead, convert it to your own types
  - Helps to deal with WSDL, SOAP engine changes, etc. without affecting main service functionality
- Some Axis specific issues
  - Service methods should explicitly define all faults that the method can throw as specified in WSDL
    - Otherwise, the faults will not be serialized correctly on the wire
  - Do NOT use full constructors to initialize the Axis generated types
    - The order of parameters keeps changing ☺


```
MyType type =
      new MyType(min, max);
```


```
MyType type = new MyType();
type.setMin(min);
type.setMax(max);
```

Lifecycle: Service

- Services can implement
  - **javax.xml.rpc.server.ServiceLifecycle** interface
    - init(Object)
      - Axis MessageContext and JAAS security subject will be associated with the thread
    - destroy()
      - Axis MessageContext will be associated with the thread
- These methods are called based on the 'scope' of the service
  - Application (one service instance is created and used for all requests)
    - init() – called when first accessed (or on container startup if *loadOnStartup* enabled)
    - destroy() – called on container shutdown
  - Request (new service instance is created on each request)
    - init() – called before each request
    - destroy() – called after each request
  - Session
    - Not supported

Lifecycle: ResourceHome

- ResourceHome can implement
  - **org.globus.wsrf.jndi.Initializable** interface
    - initialize()
      - Called when first accessed (or on container startup if *loadOnStartup* is enabled)
      - Called after all the parameters specified in the configuration file are set
      - Axis MessageContext and JAAS security subject will be associated with the thread (ResourceHome only)
  - **org.globus.wsrf.jndi.Destroyable** interface
    - destroy()
      - Called on container shutdown

Lifecycle: Resource

- Creation – resource creation is service specific
  - No API defined
- Destruction - resource object can implement
  - ***org.globus.wsrf.RemoveCallback*** interface
    - remove()
      - Called by **ResourceHome** only
  - **ResourceHome** calls remove() when
    - Resource is destroyed explicitly
      - Service implements the *ImmediateResourceTermination* port type of WS-ResourceLifetime specification
    - Resource's lease expires
      - Service implements the *ScheduledResourceTermination* port type of WS-ResourceLifetime specification
- Activation – persistent resource objects are usually activated on demand as a requests come in
  - ResourceHome could activate resources in its initialize() method

Resource Persistence

- Persistence mechanism is up to the service developers
  - Java serialization, relational database, xml database, etc.
- Resource objects can implement
  - **org.globus.wsrf.PersistentResource** interface
    - load(ResourceKey)
      - Loads resource state
        - » Does not need to load the entire resource state – only the necessary bits
        - » Rest of the state can be loaded on demand
      - Does not need to be synchronized as called once to bring the resource into memory
    - store()
      - Saves resource state
      - Must be synchronized as might be called from multiple threads at the same time
  - Use with **org.globus.wsrf.impl.ResourceHomeImpl**

Resource Persistence

- Persistence resource object must provide no-argument constructor
  - ResourceHomeImpl attempts to load the resource by
    - Creating new instance of the resource object
    - Calling the load(ResourceKey) method
      - load() either loads the resource state, or
      - Fails with NoSuchResource exception
- Define separate constructors to distinguish between new resource creation and resource activation

Container Registry

- In-memory registry of service and container configuration information
  - Created from the jndi-config.xml files deployed with services
  - Registry is only exists on the server-side
  - Services can use it to pass its own custom configuration
  - Services can use it at runtime to store some information
    - Information stored at runtime will not be persisted – registry is transient
  - Registry is visible to all services
    - Facilities direct communication with other services / resources
- Accessible via standard JNDI API
  - Retrieve configuration data, find ResourceHome of the current and other services

Container Registry

- Registry has a tree-like structure
  - java:comp/env - root of the tree
    - /services – all services are placed under this node
      - /ServiceA – each service also has its own sub-node
        - » home – service-specific resources are leaf nodes
        - » resourceA
      - /ServiceB
        - » resourceB
        - » …
    - resourceC – global resources are leaf nodes under root
    - resourceN
    - …

Obtaining reference to the registry using JNDI

- Usual method 


```
InitialContext ctx = new InitialContext();
```

- Recommended method 


```
import org.globus.wsrf.jndi.JNDIUtils;
...

InitialContext ctx = JNDIUtils.getInitialContext();
```

Works in application servers

Container Registry
Adding Custom JNDI Resources

Java class:

```
public class MyBean {
  private long timeout;

  private MyBean() {
  }

  public void setTimeout(long timeout) {
     this.timeout = timeout;
  }

  public long getTimeout() {
      return this.timeout;
  }
}
```

Resource definition:

```
<resource name="MyBean"
            type="package.MyBean">
  <resourceParams>
    <parameter>
      <name>factory</name>
      <value>
         org.globus.wsrf.jndi.BeanFactory
      </value>
    </parameter>
    <parameter>
      <name>timeout</name>
      <value>120000</value>
    </parameter>
  </resourceParams>
</resource>
```

Container Registry
Adding Custom JNDI Resources

Java class:

```
public class MyBean {
  private long timeout;

  private MyBean() {
  }

  public void setTimeout(long timeout) {
     this.timeout = timeout;
  }

  public long getTimeout() {
      return this.timeout;
  }
}
```

Can implement *Initializable* and *Destroyable* interfaces

Class must have no-argument

Define appropriate getters and setters methods. All basic types are supported. Arrays are not supported

Container Registry
Adding Custom JNDI Resources

Resource definition:

Specifies Java class

All JNDI resource must specify 'factory' parameter with that value (expect 'home' resources)

Each parameter name must correspond to a setter method in the Java class

```
<resource name="MyBean"
          type="package.MyBean">
  <resourceParams>
    <parameter>
      <name>factory</name>
      <value>
        org.globus.wsrf.jndi.BeanFactory
      </value>
    </parameter>
    <parameter>
      <name>timeout</name>
      <value>120000</value>
    </parameter>
  </resourceParams>
</resource>
```

Resource Cache

- Works only with **org.globus.wsrf.impl.ResourceHomeImpl** and **persistent** resources
  - **ResourceHomeImpl** maps resource keys to resource objects wrapped in Java SoftReferences
  - SoftReferences allow the JVM to automatically garbage collect the resource objects if nothing else references them
    - Thus, reduces memory usage and improves scalability
  - However, sometimes with SoftReferences resource objects might get GCed too frequently
    - Resource Cache prevents that by keeping temporary hard references to the resource objects
      - Cache can have size limit or time limit or both
      - Cache uses Least Recently Used (LRU) algorithm

Configuring Resource Cache

```
<service name="CounterService">
   ...
  <resource name="cache" type="org.globus.wsrf.utils.cache.LRUCache">
    <resourceParams>
      <parameter>
        <name>factory</name>
        <value>org.globus.wsrf.jndi.BeanFactory</value>
      </parameter>
      <parameter>
        <name>timeout</name>
        <value>120000</value>
      </parameter>
      <parameter>
        <name>maxSize</name>
        <value>1000</value>
      </parameter>
    </resourceParams>
  </resource>
  ...
```

Specify cache size or timeout or both

Configuring Resource Cache

```
...
<resource name="home" type="...">
  <resourceParams>
    ...
    <parameter>
      <name>cacheLocation</name>
      <value>java:comp/env/services/CounterService/cache</value>
    </parameter>
    ...
  </resourceParams>
</resource>

</service>
```

Add ‘cacheLocation’ parameter that points to the cache resource

Communication Between Services

- Regular invocations
  - Standard HTTP/S calls
  - Service can be remote or local
- Local invocations
  - In-memory, **server-side** only calls between services
    - No HTTP/S transport - uses 'local:///' protocol
      - Extra setup is necessary to use local invocation in Tomcat or other application servers
    - SOAP serialization/deserialization is performed
    - Security is enforced (message level)
- Direct invocations
  - In-memory, **server-side** only calls between services
    - Regular Java method calls achieved using JNDI
      - Can invoke things published in JNDI but cannot invoke actual service method
    - SOAP serialization/deserialization is not performed
    - Security is not enforced

Regular Invocation Example

```
URL url = new URL("http://localhost:8080/wsrf/services/MyService");

MyServiceAddressingLocator locator =
            new MyServiceAddressingLocator();
MyService port = locator.getMyServicePort(url);

port.hello();
```

Local Invocation Example

```
URL url = new URL("local:///wsrf/services/MyService");

MyServiceAddressingLocator locator =
            new MyServiceAddressingLocator();
MyService port = locator.getMyServicePort(url);

port.hello();
```

Same service just changed to 'local://' protocol

Call sequence is the same as with a regular invocation

Direct Invocation Example

```
InitialContext ctx = JNDIUtils.getInitialContext();

ResourceHome home = (ResourceHome)ctx.lookup(
                "java:comp/env/services/ContainerRegistryService/home");

// ContainerRegistryService is a singleton so lookup with a null key
RegistryService resource = (RegistryService)home.find(null);

EntryType[] entries = resource.getEntry();
for (int i=0;i<entries.length;i++) {
   System.out.println(entries[i].getMemberServiceEPR().getAddress());
}
```

Actual example that will list URLs of deployed services in the container

Background Tasks

- Instead of creating separate Threads use
  - WorkManager
    - Use for executing 'one-time' tasks
      - No while (true) { .. } type of things!
  - TimerManager
    - Used for executing periodic tasks
- Both use thread pools
  - Do not queue tasks that wait synchronously for results from other tasks
- If you have to create separate Threads
  - Limit the number of the threads
  - Have an explicit way to stop them

TimerManager Example

```
import commonj.timers.Timer;
import commonj.timers.TimerListener;
import commonj.timers.TimerManager;
…
InitialContext ctx = JNDIUtils.getInitialContext();
TimerManager timerManager =
        (TimerManager)initialContext.lookup(
                "java:comp/env/timer/ContainerTimer");

TimerListener timerTask = (new TimerListener () {
        public void timerExpired(Timer timer) {
                System.out.println("called");
        }
});

timerManager.schedule(timerTask, 1000 * 30);
```

WorkManager Example

```
import commonj.work.Work;
import commonj.work.WorkManager;
…
InitialContext ctx = JNDIUtils.getInitialContext();
WorkManager workManager =
        (WorkManager)initialContext.lookup(
                "java:comp/env/wm/ContainerWorkManager");

Work workTask = (new Work () {
        public void run() {
                System.out.println("called");
        }
        public void release() { }
        public boolean isDaemon() { return false; }
});

workManager.schedule(workTask);
```

Production Tuning

- Settings to watch for in production environment
  - JVM max/min heap size
  - File descriptors per process
  - Container service thread pool

JVM Heap Size

- Most JVM use 64MB max heap size by default
  - This might be too small for some applications
  - Indication of the problem
    - java.lang.OutOfMemoryError
      - Of course, could also indicate a memory leak in application
- To adjust, pass *–Xmx<size>m* option to JVM
  - In case of Java WS Core container set:
    - export GLOBUS_OPTION=-Xmx1024m

File Descriptors

- Most OS limit the number of opened file descriptors to 1024 per process
  - File descriptors = incoming connections + outgoing connections + opened files + pipes
  - This might be too small for some applications
  - Indication of the problem
    - java.io.IOException: Too many open files
      - Of course, could also indicate a problem in application
        - » Forgetting to close connections, files, etc.
- To adjust, see your OS documentation on how to increase this limit

Container Thread Pool

- Java WS Core container uses a thread pool for serving requests
  - Requests are also put into a queue
  - The maximum thread pool size is 20 by default
    - Used to be 8 in GT 4.0.2 and older
    - Might be too small for some applications
    - Can lead to "java.net.SocketTimeoutException: Read timed out" exceptions
      - When lots of requests queue up and there are no available threads to service them
- To adjust, edit $G_L/etc/globus_wsrf_core/server-config.wsdd file and add or modify the following parameter
  - <parameter name="containerThreadsMax" value="20"/>

General Debugging Tips

- Use a profiler tool!
- Read JVM troubleshooting documentation
  - Sun JVM
    - http://java.sun.com/j2se/1.5/pdf/jdk50_ts_guide.pdf
  - IBM JVM
    - http://publib.boulder.ibm.com/infocenter/javasdk/v5r0

Some Useful Debugging Tips

- JVM Thread Dump
  - Useful for detecting deadlocks or seeing the status of threads
  - On Unix
    - kill –QUIT <jvm process>
  - On Windows
    - Press Ctrl-Break in the window in which the JVM is running
- JVM Heap Dump
  - Useful for detecting memory problems
    - Sun JDK 1.4.2_12+ and 1.5.0_06+ only
      - Add -XX:+HeapDumpOnOutOfMemoryError option to JVM
        - » Will dump heap into a file in binary format on *OutOfMemoryError*
        - » Use a tool to examine the heap dump
    - IBM JDK 5.0
      - Will dump heap automatically on *OutOfMemoryError*

New Features in GT 4.2

- HTTP/S connection persistence
  - Improves performance especially for HTTPS connections
- WS-Enumeration support
  - Large XML datasets can be returned a chunk at a time
  - Service API for adding WS-Enumeration capabilities to any service
- TargetedXPath query dialect
  - Improved, more efficient XPath querying of resource properties
    - Use namespace prefixes reliably in the query expression
      - Explicit namespace mappings sent with the query
    - Query a particular resource property instead of the entire resource property document
    - Return query results as WS-Enumeration

New Features in GT 4.2

- Dynamic Deployment (standalone container only)
  - Deploy or undeploy (remotely) a service from the container without restarting it
  - Direct the container to reinitialize itself (after configuration change)
- SOAP with Attachments
  - Standalone container will now handle attachments
  - DIME, MIME, MTOM formats supported
- Other
  - Updated 3rd party libraries (including Axis)
  - Automatic validation of WSDD, JNDI, security descriptor files
  - Error codes in error messages

Questions?

- More information
  - GT 4.0.x
    - http://www.globus.org/toolkit/docs/4.0/common/javawscore/
  - Latest documentation (for GT 4.2)
    - http://www.globus.org/toolkit/docs/development/4.2-drafts/common/javawscore/
- Contribute to Java WS Core
  - http://dev.globus.org/wiki/Java_WS_Core

GT Java WS Security

Security Concepts Overview

- Authentication
  - Establish identity of an entity
- Message Protection
  - Integrity
  - Privacy
- Delegation
  - Empower an entity with rights of another
- Authorization
  - Ascertain and enforce rights of an identity

Outline

1. Authentication Framework
   - Message Protection
2. Delegation
3. Authorization Framework
   - Attribute Processing
4. Security Descriptor Framework
5. Writing secure service, resource and client

Authentication Framework

Authentication Schemes

- Secure Transport
  - Secure Sockets (https)
  - Anonymous access support
  - Container-level configuration
- Secure Message
  - Each individual message is secured
  - Replay Attack Prevention
- Secure Conversation
  - Handshake to establish secure context
  - Anonymous access support

Server-side features

- Message Protection options
  - Integrity and Privacy
- Configure required authentication as policy
  - At service or resource level
  - Programmatic or security descriptors
- Server response
  - Same authentication scheme as request

Client-side features

- Configurable client side authentication
  - Per invocation granularity
  - Properties on the Stub
  - Programmatically or Security Descriptors
- Message Protection options
  - Integrity and Privacy
  - Default: Integrity protection

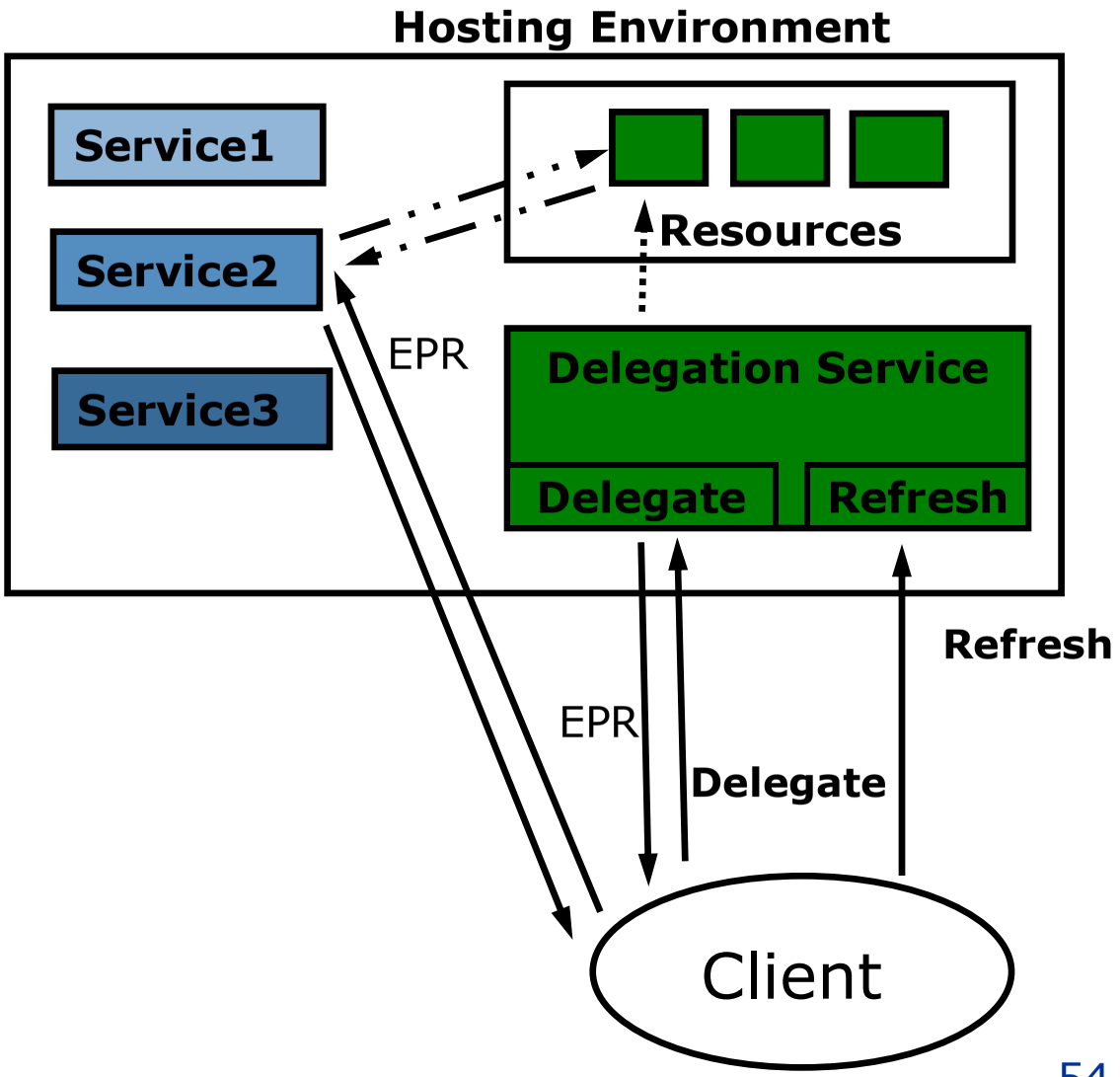
Related Utility API

- To get peer's subject:
  - SecurityManager.getManager().getPeerSubject()
- To get peer's identity
  - SecurityManager.getManager().getCaller()

Delegation

Delegation Service

- Higher level service
- Authentication protocol independent
- Refresh interface
- Delegate once, share across services and invocation



Delegation

- Secure Conversation
  - Can delegate as part of protocol
  - Extra round trip with delegation
  - Delegation Service is preferred way of delegating
- Secure Message and Secure Transport
  - Cannot delegate as part of protocol

Authorization Framework

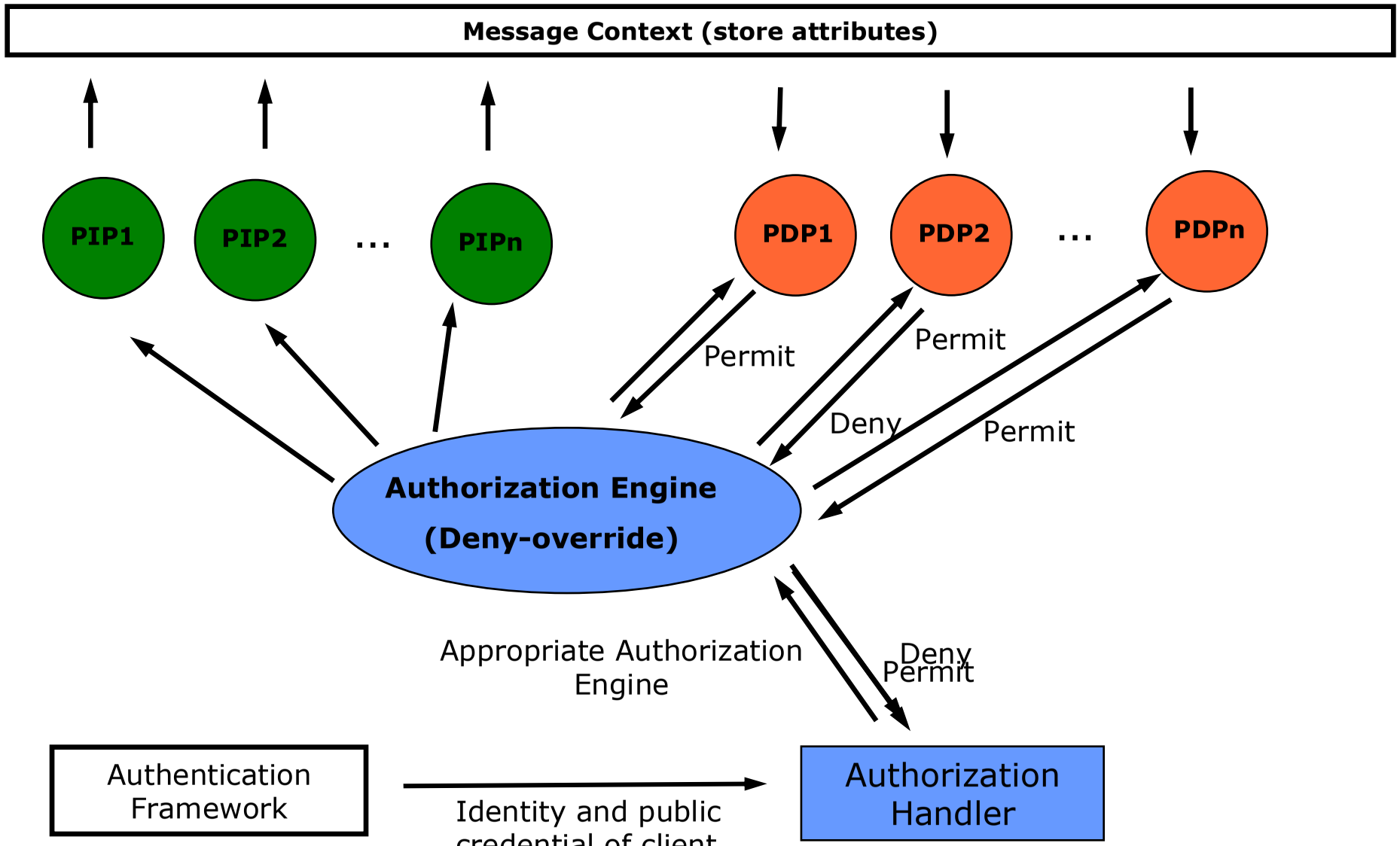
Server-side Authorization Framework

- Establishes if a client is allowed to invoke an operation on a resource
- Only authenticated calls are authorized
- Authorization policy configurable at resource, service or container level

Server-side Authorization Framework

- Policy Information Points (PIPs)
  - Collect attributes (subject, action, resource)
  - Ex: Parameter PIP
- Policy Decision Points (PDPs)
  - Evaluate authorization policy
  - Ex: GridMap Authorization, Self Authorization
- Authorization Engine
  - Orchestrates authorization process
  - Enforce authorization policy
  - Combining algorithm to renders a decision

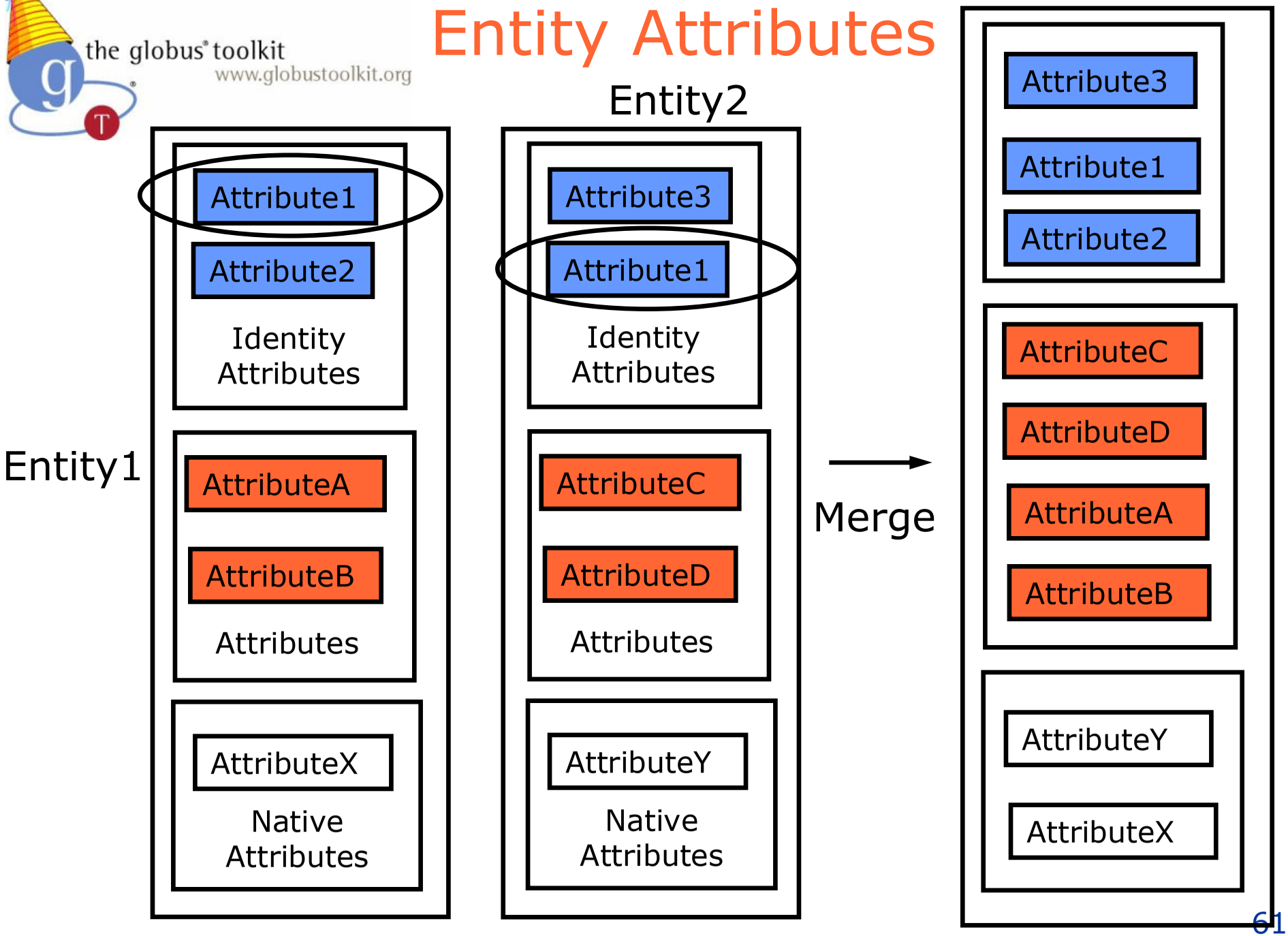
GT 4.0 Authorization Framework



GT 4.2 Attribute Framework

- Normalized Attribute representation
  - Attribute Identifier:
    - Unique Id (URI)
    - Data Type (URI)
    - Is Identity Attribute ? (boolean)
  - Set of values
  - Valid from
  - Valid to
  - Issuer
- Comparing attributes

Entity Attributes



GT 4.2 Attribute Framework

- Bootstrap PIP
  - Collects attributes about the request: subject, action and resource
  - Example: X509BootstrapPIP

GT 4.2 PDP Interface

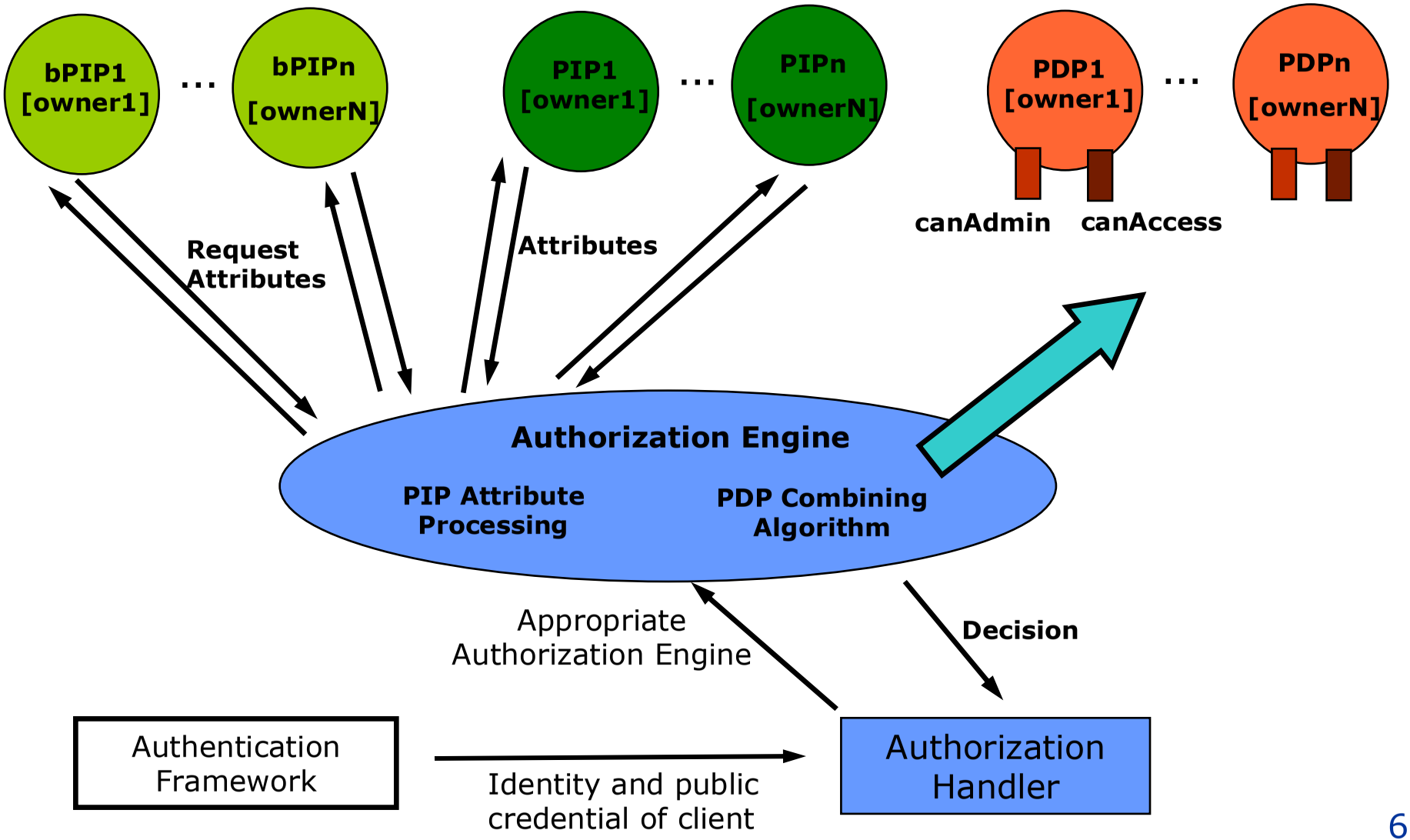
- Access rights
  - canAccess()
- Administrative rights
  - canAdmin()
- Return type: Decision
  - PERMIT/DENY/INDETERMINATE
  - Issuer of decision
  - Validity
  - Exception, if any

GT 4.2 Authorization Engine

- Pluggable combining algorithm
- AbstractEngine.java
  - Initializes PIPs and PDPs with configured parameters
  - Invokes collectAttributes() on all PIPs
  - Merges the entity attributes returned by PIPs
  - *Abstract method engineAuthorize process PDPs*
    - *Combines decisions from individual PDPs*
    - *Returns Decision*
- Default combining algorithm
  - Permit override with delegation of rights
  - At-least one decision chain from resource owner to requestor for a PERMIT

the globus toolkit
www.globustoolkit.org

GT 4.2 Authorization Framework



Authorization Engine Precedence

- Authorization engine used
  - Administrative authorization engine (container)

    <AND>

    1. Resource level authorization engine <OR>
    2. Service level authorization engine <OR>
    3. Container level authorization engine
- Default:
  - X509BootstrapPIP and Self authorization

Authorized User Information

- Getting information on authorized user
  - $GLOBUS_LOCATION/container-log4j.properties
  - # Comment out the line below if you want to log every authorization decision the container makes.

log4j.category.org.globus.wsrf.impl.security.authorization.AuthorizationHandler=WARN

Client-side Authorization

- Determines if said service/resource is allowed to cater to the client's request
- Pluggable authorization scheme
  - Defined interface, implement custom schemes
- Configured as property on stub or using security descriptors
- Examples: Self, Host, Identity, None
- Default: Host
- Required when secure conversation is used with delegation

GT 4.2 Enhancements

- HostOrSelf Authorization
  - Algorithm:
    - Do host authorization
    - If it fails, do self authorization
  - Set as default in 4.2 code base

Security Descriptor Framework

Security Descriptor Overview

- Used to configure security properties
- Declarative security
  - Configure properties in files
- Different types of descriptors for container, service, resource and client security properties
- GT 4.2 Enhancements
  - Defined schema for each descriptor

Server-side Security Descriptor

- Container descriptor in global section of deployment descriptor
  - $GLOBUS_LOCATION/etc/globus_wsrf_core/server-config.wsdd
  - Parameter: containerSecDesc
  - Can be done only in this file
- Service descriptor in service's deployment descriptor
  - Parameter: securityDescriptor
- Resource descriptor set programmatically
  - Load from file or use ResourceSecurityDescriptor object
- Loaded as file or resource stream

GT 4.2 Credentials Configure

- Proxy file name
  - <credential>
    <proxy-file value="proxy file"/>
    </credential>

- Certificate and key filename
  - <credential>
    <cert-key-files>
    <key-file value="key file"/>
    <cert-file value="certificate file"/>
    </cert-key-files>
    </credential>
- Absolute file name, as resource stream, relative to $GLOBUS_LOCATION

GT 4.2 Service Authentication Policy

- Default for all operation:

```
<auth-method>
  <GSISecureTransport/>
   <GSISecureMessage/>
</auth-method>
```

- Per operation configuration:

```
<methodAuthentication>
    <method name="createCounter">
            <auth-method>
                    <GSISecureConversation/>
            </auth-method>
    </method>
    <method name="destroy">
            <auth-method>
                    <GSISecureMessage>
                            <protection-level>
                                    <privacy/>
                            </protection-level>
                    </GSISecureMessage>
            </auth-method>
    </method>
</methodAuthentication>
```

GT 4.2 Run-as Configuration

- Determines the credential to associate with current thread
- Options: caller, system, service, resource
- All methods:
  - <run-as value="system"/>
- Per method:
  - <method name="subtract">
    <run-as value="caller"/>
    </method>

the globus toolkit
www.globustoolkit.org

GT 4.2 Authorization Configuration

```
<authzChain combiningAlg="org.globus.sample.SampleAlg" >
<authzChain>
  <bootstrapPips overwrite="true">
    <interceptor name="scope1:org.globus.sample.BootstrapPIP1"/>
  </bootstrapPips>

  <pips>
    <interceptor  name="scope2:org.globus.sample.PIP1"/>
  </pips>

  <pdps>
 <interceptor name="foo1:org.foo.authzMechanism/>
 <interceptor name="bar1:org.bar.barMechanism"/>
  </pdps>

 </authzChain>
```

Permit Override with delegation

X509BootstrapPIP is also invoked

Only X509BootstrapPIP is invoked

GT 4.2 Authorization Parameters

```
<containerSecurityConfig
    xmlns="http://www.globus.org/security/descriptor/container"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.globus.org/security/descriptor
    name_value_type.xsd"
    xmlns:param="http://www.globus.org/security/descriptor">


<authzChain> <pdps>
<interceptor
    name="prefix:org.globus.wsrf.impl.security.GridMapAuthorization">
<parameter>
     <param:nameValueParam>
       <param:parameter name="gridmap-file" value="C:/grid-mapfile"/>
     </param:nameValueParam>
</parameter>
</interceptor> </pdps> </authzChain>
</containerSecurityConfig>
```

Related Utility API

- To get resource credential
  - SecurityManager.getManager().getResourceSubject()
- To get service credential
  - SecurityManager.getManager().getServiceSubject()
- To get container credential
  - SecurityManager.getManager().getSystemSubject()
- To get effective credential
  - SecurityManager.getManager().getSubject()

Client side descriptor

- Security descriptor file
  - ((Stub)port).setProperty(Constants.CLIENT_DESCRIPTOR_FILE, fileName);
  - Absolute path or as resource stream or relative to $GLOBUS_LOCATION
- Security descriptor object
  - ((Stub)port).setProperty(Constants.CLIENT_DESCRIPTOR, instance of ClientSecurityDescriptor);

GT 4.2 Authentication Configuration

- GSI Secure Transport

```
<GSISecureTransport>
    <anonymous/>
</GSISecureTransport>
```

- GSI Secure Conversation

```
<GSISecureConversation>
    <integrity/>
</GSISecureConversation>
```

- GSI Secure Message

```
<GSISecureMessage>
    <privacy/>
    <peer-credentials value="path to peer's public key"/>
</GSISecureMessage>
```

GT 4.2 Authorization Configuration

- Authorization Element
  - <authz value="self"/>
- Values:
  - none
  - host
  - self
  - hostOrSelf
  - Expected DN as string
- Does not support custom authorization configuration

Writing secure service, resource and client

Writing Secure Service

- Create security descriptor file
  - Typically placed in service source/etc
    - Ensure your build process picks up etc directory into gar
  - Part of the source jar
  - Name file *security-config.xml
- Add parameter to deployment descriptor
  - <parameter name="securityDescriptor" value="etc/globus_sample_counter/security-config.xml"/>

Writing Secure Service

- Write security properties in descriptor file
- Deploy service
- GT 4.2, Run validate tool
  - globus-validate-descriptors
  - All files *security-config.xml are validated

Writing Secure Resource

```
public class TestResource implement SecureResource {

    ResourceSecurityDescriptor desc = null;
    public  TestResource() {
        this.desc = new ResourceSecurityDescriptor();
        // set properties programmatically
        this.desc.setDefaultRunAsType(RunAsValue._caller);
    }
     public ResourceSecurityDescriptor
          getSecurityDescriptor() {
          return this.desc;
     }
}
```

Writing Secure Client

- Construct ClientSecurityDescriptor
  - From file
  - Programmatically
- Extend from org.globus.wsrf.client.BaseClient
  - Parses standard security parameters
  - Use setOptions(stub) to set relevant security parameters
- If using GSI Secure Transport, Util.registerSecureTransport()
- If contacted service uses GSI Secure Transport, container's identity should be expected

Questions?

- Future Work:
  - http://www.globus.org/roadmap/Projects.cgi#security
- Documentation:
  - http://www.globus.org/toolkit/docs/development/4.2-drafts/security/index.html
- Code:
  - http://viewcvs.globus.org/viewcvs.cgi/wsrf/
- Contributions:
  - http://dev.globus.org/wiki/Java_WS_Core

Question: Do you see a Fun & Exciting Career in my future?

Magic 8 Ball: All Signs Point to **YES**

Say YES to Great Career Opportunities

Argonne NATIONAL LABORATORY

SOFTWARE ENGINEER/ARCHITECT

Mathematics and Computer Science Division, Argonne National Laboratory

The Grid is one of today's hottest technologies, and our team in the Distributed Systems Laboratory (**www.mcs.anl.gov/dsl**) is at the heart of it. Send us a resume through the Argonne site (**www.anl.gov/Careers/**), requisition number **MCS-310886.**

SOFTWARE DEVELOPERS

THE UNIVERSITY OF CHICAGO

Computation Institute, University of Chicago

Join a world-class team developing pioneering eScience technologies and applications. Apply using the University's online employment application (**http://jobs.uchicago.edu/**, click "Job Opportunities" and search for requisition numbers **072817 and 072442**).

See our Posting on the GlobusWorld Job Board or Talk to Any of our Globus Folks.