How to set up a GT4 web service with VOMS authorisation

A play in three acts

D. H. van Dok[*]

June 1, 2006, rev. 1.6

Introduction

The challenge to set up *any* web service in the Globus Toolkit 4 (GT4) Web Services Resource Framework (WSRF) can be daunting. In spite of the availability of excellent tutorials from IBM and Globus, the number of non-trivial steps and pitfalls is a guarantee to run into trouble at some point.

This documentation lists the steps and pitfalls I encountered in setting up an authorization scheme with VOMS (VO Management System) for a web service called DeploymentService (whose purpose is irrelevant right now). The characters, a Master and his pupil, represent my own perspectives during various stages of blundering agony and final enlightenment. The setting is a vanilla installation of Linux.

We bring together a number of technologies that all fall in the "grid computing" bin. The Globus Toolkit is a collection of grid middleware, with the recent version 4 being mostly reworked into a service-oriented architecture. The WSRF is a standard to extend web services—which are inherently stateless—to include state by means of 'stateful resources'; we're not going to do anything with that, but it is significant as the web service in the examples below is going to be deployed on the lighter-weight WSRF distribution rather than the full-blown toolkit.

VOMS is the way (a way?) to deal with VO membership information; a grid user can pass his VO membership attributes to a service by using an extended kind of proxy that's been signed by the VOMS server. The service can then make an informed decision based on the caller's organisation and role.

This is not just a random draw from the bin. There is a strong technology push for web services and the GT4 container, which will be found all over the grid infrastructure, is a natural choice for the deployment of grid services. Any serious application will need a proper way to deal with authorisation and VOMS is looking good.

The reader should have some familiarity with web services and grid authorisation with certificates and proxies. In particular, the GT4 Programmer's Tutorial [1] is recommended first reading.

Characters

Novice An aspiring apprentice.

Master A spiritual and worldly guide.

Root The groundskeeper.

Proxy A temporary VOMS proxy.

Props

- An unrevoked grid certificate [2].
- The Globus WSRF Java WS core [3].
- The SUN j2sdk, $\geq$ 1.4.2 [4].

[*]National Institute for Nuclear Physics and High Energy Physics, Amsterdam, the Netherlands (NIKHEF). Virtual Laboratory for e-Science, http://www.vl-e.nl/

Act 1 VOMS Credentials

Act 1, Scene 1 Setting Up

Uncharted territory, `/home/user`

[Enter Novice and Master; the novice is carrying a heavy pack.]

Novice: Ah, this looks as good a place as any to pitch up my tent and make camp! Look at the scenic view! Nothing but space for at least 10 Gigabytes around. And the silence! What a relief to get away from the madness of the city for a while, with its traffic jams and busy processes. Nothing to disturb the silence but the periodic call of a crond; why, I think I can even hear the distant murmur of `/dev/null`!

Master: This is the place you will call `$HOME`. We'll set up our training dojo over there by those trees, and here by the little stream will be the meditation garden.

Novice: I don't thing I have ever seen such wonderful trees! What kind are they?

Master: They are called 'source trees', for it is said they hold the secret to the source of life.

Novice: Master, may I know now what is in this heavy pack that I've been hauling all the way here? I think I'm entitled to know as I almost broke my back carrying it.

Master: Patience, my young padawan. You will learn everything in due time. Now rehearse what we have set out to do here.

Novice: Again? But. . .

Master: Yes, again.

Novice: We are here to show how a web service can be set up with authorisation via VOMS attributes, but I don't understand...

Master: Of course you don't! Otherwise I wouldn't have brought you here. What do you feel is necessary to accomplish our goal?

Novice: Errr. . . VOMS attributes?

Master: Very good! It is time for your rite of initiation. Now drop that large pack and off we go to the VOMS admin lair!

[exeunt]

Act 1, Scene 2 VO Membership

Outside the VOMS admin lair, https://kuiken.nikhef.nl:8443/voms/TEST/.

[Enter Master and Novice, panting.]

Novice: That was a long run! I'm exhausted.

Master: There is no time to rest yet. Here you must finish your rite of initiation.

Novice: What is this place? It looks ominous. What creature lurks here?

Master: Here dwells the VOMS admin; a powerful being that will hand you your VOMS attributes if you approach it carefully. You must go inside by this secured port; only with a valid certificate loaded in your browser will you be admitted entry (See [5].)

Inside, you will have to ask for membership of the `TEST` VO. The VO admin will ask you several questions, which you must answer as best you may. Only those who are pure of heart will be able to answer all questions.

Novice: Wish me luck!

[Enters lair, and returns after a short while]

Novice: It asked my name, and email address. Then there was some e-mail exchange with the server, asking me to confirm my email address; now I just have to await the response from the admin. Some rite!

Master: Well done! While we let the admin ponder your submission, we'll head back to base camp to attend to other matters.

[exeunt]

Act 1, Scene 3 VOMS Client Software

Base camp, `$HOME`.

[Enter Root.]

Root: What is this? An illegal camping site? This is intolerable. Where are the campers?

[Enter Master and Novice]

Master: Hello, who are you?

Root: I am Root, the keeper of these woods. You are camping here without permission!

Master: Oh. Can we have your permission? Me and my novice are here to study VOMS authorisation for web services.

Root: Err, I, well, I suppose it's alright. Just don't disturb the wildlife.

Master: Pleased to meet you. Apropos, could you help us with a slight problem we have?
Root: What's that?
Master: My novice needs to get to his VOMS attributes, but to do so we need the VOMS client tools installed. It seems to me that this rural CentOS 4 environment does not have them. Could you install this list of packages?

```
http://glite.web.cern.ch/glite/packages/externals/bin/rhel30/
        RPMS/gpt-VDT1.2.2rh9-1.i386.rpm
http://glite.web.cern.ch/glite/packages/externals/bin/rhel30/
        RPMS/vdt_globus_essentials-VDT1.2.2rh9-1.i386.rpm
http://glite.web.cern.ch/glite/packages/R1.5/R20051130/bin/rhel30/
        i386/RPMS/glite-security-voms-api-cpp-1.6.10-0.i386.rpm
http://glite.web.cern.ch/glite/packages/R1.5/R20051130/bin/rhel30/
        i386/RPMS/glite-security-voms-clients-1.6.10-0.i386.rpm
```

Root: What? I can't do that! Those aren't trusted, certified RPMS! They're not even for the right system! For all I know these RPMs violate every known policy, and undermine the security, and...
Master: [Waves hand] You *will* install these packages
Root: I will install these packages
Master: You are glad to be of service.
Root: I am glad to be of service
Master: Now move along.
Root: I must be moving along. Happy camping!
Novice: What was that? Did you pull some kind of trick on him or something? Can I learn that?
Master: Not now. While we wait for the packages to be installed, we shall unravel the mystery of this big pack of yours.
Novice: Ah!
Master: Pick it up, we need to find a large, open space...
[exeunt]

Act 1, Scene 4 Globus Toolkit WSRF

An open field, the size of a football pitch.
[Enter Master and Novice, with pack.]
Master: Time to inflate this pack. [Pulls cord]
Novice: Wow, this thing is bigger than I thought. No wonder it was heavy.
Master: This is the Globus Toolkit 4, Java WS core [3]. It is a smaller package than the full-blown toolkit, and you'll thank me for not having you drag *that* all the way here.

This is the minimal binary package, which suits us fine, but for educational purposes you may want to install the sources in the same place.

The WS core contains a web services container, a patched version of Apache Axis 1.2. It also contains some base services and useful scripts and tools. In this container we'll deploy our web service later. Let's call it `$GLOBUS_LOCATION`.
Novice: So how do we run the container?
Master: Put `$GLOBUS_LOCATION/bin` in your `$PATH`, and run `globus-start-container -nosec`. That means "no transport level security", as we may want to monitor our SOAP messages later with the TCP monitor.
Novice: But isn't security needed to deal with VOMS authorisation?
Master: Yes, of course it is; but that is handled by *message level security* as you'll see later.

There is one little detail about the container that we have to deal with now. In the file
`$GLOBUS_LOCATION/etc/globus_wsrf_core/server-config.wsdd`,
you may want to remove the Globus spyware option [6] that will send usage statistics every time the container is started. Remove the ⟨parameter⟩

```
<parameter name="usageStatisticsTargets"
                value="usage-stats.globus.org:4810"/>
```

in the ⟨globalConfiguration⟩.

In the meantime, I suspect that the VOMS client software has been installed; go to the meditation area and try a voms-proxy-init.
[exeunt]

Act 1, Scene 5 Certification Authorities

A tool shack; Root.

Root: [rummaging about] Now where did I leave that thing, it must be here somewhere...
[enter Master]
Master: Good day, old friend. This must be `/etc`.
Root: What, you again? Go away, this place is off-limits!
Master: No it's not. You wouldn't happen to have some grid-security around here would you?
Root: No, I don't believe I do; I don't see why I should.
Master: You should because all grid software looks there for authentication and certificate chain checking.
Root: I really don't have time for such things! I am a busy man, you know.
Master: It won't take long; just put an entry for the distribution of certificate authority (CA) public keys in yum.conf and run an install.

```
[eugridpma]
name=EUGridPMA
baseurl=http://www.eugridpma.org/distribution/igtf/current/
gpgcheck=1
```

Root: All right, all right, I'll put it in `/etc/yum.repos.d/ca.repo` Now, what packages should I install?
Master: Just the `ca_policy_eugridpma`; that will pull in the rest through the dependencies. Since you have to check the package's `gpg` signatures, you may want to import the public key first.

```
rpm --import http://www.eugridpma.org/distribution/igtf/current/GPG-KEY-EUGridPMA-RPM-3
```

Incidentally, you may also want to get the certificate revocation lists (CRLs); this is done automatically with `fetch-crl` which can be installed by

```
rpm -i http://www.eugridpma.org/distribution/util/fetch-crl/fetch-crl-2.5-1.noarch.rpm
```

and copying the example cron file to `/etc/cron.daily/`.
Root: I will do so in a moment. Now if you would please leave, I have something urgent to attend to.
Master: OK, see you later. [exit]
Root: If it were not for users, I'd be much happier. [exit]

Act 1, Scene 6 VOMS Proxy

A patch of moss by a chattering stream. Novice, meditating.

[enter Master]
Master: Any news yet from the VOMS admin?
Novice: Yes; a carrier pigeon just flew by and dropped off a letter. It said:

```
Accepted VO membership request


Dear Novice,

Your request (32) for the TEST VO has been accepted and
allowed by the VO Administrator.

>From this point you can use the voms-proxy-init command
to acquire the VO specific credentials, which will enable
you to use the resources of this VO.

Good Luck,
     VO Registration
```

But I couldn't find the `voms-proxy-init` command, so I went to look around for it. I found it in `/opt/glite/bin`,
but then it couldn't find some shared libraries, and I had to type

```
LD_LIBRARY_PATH=/opt/glite/lib:/opt/globus/lib /opt/glite/bin/voms-proxy-init
```

but that generated an ordinary proxy without VOMS attributes. It also complained:

```
Cannot find file or dir: /home/novice/.glite/vomses
Cannot find file or dir: /opt/glite/etc/vomses
```

So now I sit here in meditation, contemplating my problem.

Master: If you visit https://kuiken.nikhef.nl:8443/voms/TEST/webui/config you'll see a line that can be put into a "`vomses`" file. Just put

```
"TEST" "kuiken.nikhef.nl" "15000" "/O=dutchgrid/O=hosts/OU=nikhef.nl/CN=kuiken.nikhef.nl" "TEST"
```

in `$HOME/.glite/vomses` and then call `voms-proxy-init -voms TEST`. All should be well. Make double sure that the mode of the file is `644` (`-rw-r--r--`), or the software refuses to work.

Novice: It works! Where *do* you learn these things?

[enter Proxy]

Proxy: Your wish is my command (for the next twelve hours).

Novice: Now we shall finally see my VOMS attributes! Show me `voms-proxy-info`.

Proxy:

```
WARNING: Unable to verify signature!
Error: Cannot find certificate of AC issuer for vo TEST
```

Master: It is good to explain a few things now. First, `voms-proxy-init` is a variant of the usual `grid-proxy-init`, where upon request a connection is made with the VOMS server to collect the attributes. These are signed by the server and put inside your proxy. Second, `voms-proxy-info` verifies the attribute signature with the public key of the server, as would any form of authorisation against these attributes. Thirdly, the public key (a `pem` file) of the VOMS server must be placed under `/etc/grid-security/vomsdir`. Meditate some more, while I hassle Root into installing this file. [exit]

Novice: H'm. The mystery of it all.

[enter Master]

Master: Root was most cooperative. The VOMS admin sent us the public key file and Root installed it in

```
/etc/grid-security/vomsdir/dec-2005-kuiken.nikhef.nl.pem
```

so now it should work. By the way, call `voms-proxy-info` with the `-all` flag or you still won't see the attributes.

Novice: Hurray! It worked!

Proxy: Here are my attributes:

```
$ voms-proxy-info -all
=== VO TEST extension information ===
VO        : TEST
subject   : /O=dutchgrid/O=users/O=vlescience/CN=Novice
issuer    : /O=dutchgrid/O=hosts/OU=nikhef.nl/CN=kuiken.nikhef.nl
attribute : /TEST/Role=NULL/Capability=NULL
attribute : /TEST/tmp/Role=NULL/Capability=NULL
timeleft  : 10:51:08
```

Master: You have now completed the first stage of your training. In the next, we will look at the source tree of life, and web services.

[exeunt]

Act 2 Web Service

Act 2, Scene 1 Web Service Sources

Amidst a forest of source trees

Novice: What a wonderful forest this is; such variety of trees, and all teeming with life. But what is this? The climate here must be particularly mild, for if I'm not mistaken these trees only grow on Java.

Master: You are not mistaken; they have been specially imported. But take a closer look at the wildlife.

Novice: They are mostly bugs, but I also see some worker ants amongst them. They seem to busy themselves carrying snippets of bytecode up and down. They are compiling these enormous anthills.

Master: Those are called builds. The ants are wonderful creatures, that can be trained to perform many useful tasks. But look how the bugs get in their way, sometimes killing them, sometimes demolishing their builds. The task of a caretaker of source trees is to get rid of the nasty bugs, and they say it's never done.

Novice: Here I shall learn about web services. I will start by planting my own little source tree. Where do I begin?

Master: A web service is little more than a function that can be called remotely. So a simple Java class with a public method will serve as an example.

Novice: So how is this for a start?

```
package org.vlescience.webservices.deployment;
public class DeploymentService  {
    public java.lang.String deploy(java.net.URL gar) {
        // do nothing and assume success
        return "Success.\n";
    }
}
```

I call it the DeploymentService, whose purpose is to fetch a GAR file from a given URL and to deploy it on some Globus Toolkit container; the return value is a simple string reporting the success of this operation.

Master: Very well, that will do for now. Put it in

```
./org/vlescience/webservices/deployment/DeploymentService.java
```

and test that it can be compiled by hand.

Novice: Where shall I find a compiler?

Master: Over there between the trees you see a Sunlit patch [4]. Call it `$JAVA_HOME` and put its `/bin` in your path, then you shall find `javac`.

Novice: I thought we would rather let the ants do all our building?

Master: Certainly, but take a closer look at the kind of ant you find here.

Novice: That's funny! They all have the same peculiar marking on their head: "ant-1.6.2-3jpp".

Master: There are many kinds of ants in this world, and many versions, and not all of them work very well. But this kind is good; it's version 1.6 and was provided by the JPackage Project [7] ant farm.

The package we have in `$GLOBUS_LOCATION` contains several ant helper scripts for useful tasks, but for building and deploying our web service we shall fetch a useful addition: the globus-build-service [8]. This is a ready-to-use shell script plus an ant build file that will build our service, provided that it follows a certain directory structure.

Novice: Well, that's nice. Will this automagically turn my trivial class into a web service?

Master: You do not believe in magic, do you? To turn your class into a service, you need to do at least two more things.

Novice: What are they?

Master: One, provide a WSDL file which holds a description of your service in terms of the kind of messages that are involved and the data types that are being transmitted by these messages. Two, write a deployment descriptor containing all the housekeeping information for the sake of the container.

Novice: Can't the WSDL be generated directly from the Java source?

Master: Stumble in the dark, you will, if that path you follow, young padawan.

Novice: Funny you talk.

[exeunt]

Act 2, Scene 2 WSDL Wisdom

Meditation area, `schema/vlescience`.

[Enter Novice, and Master]

Novice: Why must I meditate more? Is it so hard to write a bit of WSDL?

Master: A fool can ask more than a wise man can answer.

Novice: What?!

Master: Let's hear your take on WSDL and how that works out with your service.

Novice: A web services definition language file describes a web service by it's port types; each port type specifies a number of operations; each operation has an input and an output message. Each message has a prescribed format as defined by its type.

Lastly, a binding defines the protocols through which a port type operates; traditionally, SOAP and HTTP.

Master: Well rehearsed; but given the many degrees of freedom in this specification, what would be the right choices for your service?

Novice: I think I need more meditation. . .

Master: There are some styles of WSDL to choose from, but the differences are subtle; see e.g. [9] for more advice. Let's avoid the philosophical discussion and go with "document/literal wrapped".

Novice: What does that look like?

Master: Well, each ⟨message⟩ has a single ⟨part⟩ named "parameters" corresponding to an element previously defined in the ⟨types⟩ section. As a convention it is named after the operation.

Novice: So the following snippet would define the call and return messages to and from my trivial service.

```
  <message name="DeployRequest">
    <part name="parameters" element="Deploy"/>
  </message>
  <message name="DeployResponse">
    <part name="parameters" element="DeployResponse"/>
  </message>
```

Master: Very good! Now to let the parameters correspond to the method's arguments and return value, define these types in a bit of schema.

Novice: How about this?

```
    <schema>
      <element name="Deploy" type="anyURI"/>
      <element name="DeployResponse" type="string"/>
    </schema>
```

Master: OK, you can get away with using simple types here, as your method has only one argument. Otherwise you'd have to use a ⟨complexType⟩.

This is really coming together nicely. The only thing left to do is to specify a ⟨portType⟩, which is almost trivial. Give the operation the name of the method in your service and tell it which ⟨message⟩s to use for input and output. There is only one gotcha: there is a sneaky conversion of the *first* capital in the name of the operation to lowercase, when Axis generates the mapping to Java calls. This is supposedly done to fall in with the standing convention in Java to use mixedCaseIdentifiers starting with a lowercase for methods, but this only seems to work well if you consistently name your *operations* with a starting capital.

Novice: This should do the job then.

```
  <portType name="deploymentPortType">
    <operation name="Deploy">
      <input message="DeployRequest"/>
      <output message="DeployResponse"/>
    </operation>
  </portType>
```

But I'm confused now, because I thought we would also be mentioning ⟨binding⟩ and ⟨service⟩ parts.

Master: I thought you wanted a bit of magic earlier on? Well, that is provided by the Globus tools. When we let ant build the stubs later on, the remaining WSDL elements are generated as well.

Let's put together what we've got in a complete and valid WSDL document, with all the namespace boilerplate in place.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="deployment"
  targetNamespace="http://webservices.vlescience.org/deployment"
  xmlns:tns="http://webservices.vlescience.org/deployment"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="Deploy" type="xsd:anyURI"/>
      <xsd:element name="DeployResponse" type="xsd:string"/>
    </xsd:schema>
  </types>

  <message name="DeployRequest">
    <part name="parameters" element="tns:Deploy"/>
  </message>

  <message name="DeployResponse">
    <part name="parameters" element="tns:DeployResponse"/>
  </message>

  <portType name="deploymentPortType">
    <operation name="Deploy">
      <input message="tns:DeployRequest"/>
      <output message="tns:DeployResponse"/>
    </operation>
  </portType>

</definitions>
```

Now tuck it away in `schema/vlescience/deployment.wsdl` and that's that.
[exeunt]

Act 2, Scene 3 WSDD (Why So Darn Difficult?)

Amongst the source trees.

[Enter Master and Novice]

Master: We are almost set to start building our service, but there is a bit of gardening still to do. First, download the globus build service [8]. It contains three files, of which we only need `build.xml` and `globus-build-service.sh`. Plant them at the root of our source tree.

Novice: Here we are; let's see what happens if I run it now.

```
$ ./globus-build-service.sh

Usage:
./globus-build-service.sh -d <service_dir> -s <schema_file> \
        [-fs <factory_schema_file>] [-t <target>] [--debug]
./globus-build-service.sh <service_id> [target] [--debug]
./globus-build-service.sh -h

<service_dir> is the directory that contains all the implementation and deployment files:
        <service_dir>/deploy-server.wsdd         Deployment file (mandatory)
        <service_dir>/deploy-jndi-config.wsdd    JNDI deployment file (mandatory)
        <service_dir>/impl/*.java                Java implementation files (mandatory)
        <service_dir>/etc/*.xml                  Configuration files (optional)

<schema_file> is the WSDL file with the service's interface description

<factory_schema_file> is an optional parameter. If your service is a
factory/instance service, you can use this parameter to specify the factory's
schema file.

<target> is an optional parameter to control what Ant builds. Valid values are
         all     Builds everything (default)
         stubs   Generates the stubs (but doesn't compile them).
         compileStubs   Generates and compiles the stubs.

--debug provides detailed information of what the build script is doing.

The script offers a shorthand way of building services through the <service_id>
parameter. It allows you to build services without having to type the
service directory and schema file every time. You must have a 'build.mappings'
file in the same directory as the build script, with one line for each service
using the following format:
        <service_id>,<service_dir>,<schema_file>,<factory_schema_file>

         (the <factory_schema_file> is optional)
```

H'm. It seems I may be missing some essentials. Also, my service is not in a directory named `impl`.

Master: The JNDI file is not *really* necessary, but you do need a WSDD file and you also need to work on your java source. Let's do that now.

Novice: So I create a directory named `org/vlescience/webservices/deployment/impl`?

Master: Yes. It is a convention that was introduced by Sotomayor, and we will adhere to it. Move the file `DeplomentService.java` over there and change the first line to

```
package org.vlescience.webservices.deployment.impl;
```

Novice: And now for the Web Service Deployment Descriptor, `deploy-server.wsdd`.

Master: This file is a bag of odds and ends that has the purpose of explaining to Apache Axis how exactly this service is supposed to work. I won't bother with the details here, as it distracts us too much from our goal of learning about VOMS authorisation. Here is the complete file.

```
<deployment
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
```

```
  <service name="DeploymentService"
           provider="Handler"
           style="document"
           use="literal">

    <parameter name="handlerClass"
               value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="className"
               value="org.vlescience.webservices.deployment.impl.DeploymentService"/>
    <parameter name="allowedMethods"
               value="*"/>
    <parameter name="scope"
               value="Application"/>

    <wsdlFile>
      share/schema/vlescience/deployment_service.wsdl
    </wsdlFile>

  </service>
</deployment>
```

Just put it in `org/vlescience/webservice/deployment/deploy-server.wsdd`.

Novice: Wait just a minute... Where does that ⟨wsdlFile⟩ come from?

Master: Remember that our WSDL was incomplete. The build script will fill in the blanks and generate a complete WSDL file.

Now that we have the sources in place, we can fire up our *build* by calling `globus-build-service`; if that works we can deploy the resulting `gar` file to the container.

Here is an overview of what is in our source tree so far:

```
build.xml
globus-build-service.sh
org/vlescience/webservices/deployment/deploy-server.wsdd
org/vlescience/webservices/deployment/impl/DeploymentService.java
schema/vlescience/deployment.wsdl
```

Novice: Here we go!

```
./globus-build-service.sh -d org/vlescience/webservices/deployment  \
        -s schema/vlescience/deployment.wsdl
```

This produces lots of output... what is all this about generating stubs?

Master: Stubs are for clients; they provide the translation from a straightforward call to your service's method to the appropriate SOAP messages.

Novice: Now it says: `BUILD SUCCESSFUL` and it's done. Hey, looky here, it created a GAR file! Let's see what is inside...

```
$ unzip -l org_vlescience_webservices_deployment.gar
Archive:  org_vlescience_webservices_deployment.gar
  Length     Date   Time    Name
 --------    ----   ----    ----
        0  05-16-06 11:25   META-INF/
      106  05-16-06 11:25   META-INF/MANIFEST.MF
        0  05-16-06 11:25   lib/
     1470  05-16-06 11:25   lib/org_vlescience_webservices_deployment.jar
     9307  05-16-06 11:25   lib/org_vlescience_webservices_deployment_stubs.jar
        0  05-16-06 11:25   schema/
        0  05-16-06 11:25   schema/vlescience/
      868  05-16-06 11:25   schema/vlescience/deployment.wsdl
      953  05-16-06 11:25   schema/vlescience/deployment_bindings.wsdl
      854  05-16-06 11:25   schema/vlescience/deployment_flattened.wsdl
      694  05-16-06 11:25   schema/vlescience/deployment_service.wsdl
      710  05-16-06 11:25   server-deploy.wsdd
 --------                   -------
    14962                   12 files
```

It sure produced a lot of WSDL.

Master: The WSDL files are just a breakdown of the parts that you would normally call a WSDL file. If you inspect the `deployment_service.wsdl` you see that it imports the others.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="deployment"
    targetNamespace="http://webservices.vlescience.org/deployment/service"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:binding="http://webservices.vlescience.org/deployment/bindings"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:import
      namespace="http://webservices.vlescience.org/deployment/bindings"
      location="deployment_bindings.wsdl"/>

  <wsdl:service name="deploymentService">
    ...
```

The GAR file can be deployed on the GT4 container by using `$GLOBUS_LOCATION/bin/globus-deploy-gar`.

Novice: OK.

Master: And if you now try `globus-start-container -nosec` you can see that your service is launched.

Novice: Hurray!

```
Starting SOAP server at: http://127.0.0.1:8080/wsrf/services/
With the following services:

[1]: http://127.0.0.1:8080/wsrf/services/Version
[2]: http://127.0.0.1:8080/wsrf/services/NotificationConsumerService
[3]: http://127.0.0.1:8080/wsrf/services/NotificationTestService
[4]: http://127.0.0.1:8080/wsrf/services/SecureCounterService
[5]: http://127.0.0.1:8080/wsrf/services/PersistenceTestSubscriptionManager
[6]: http://127.0.0.1:8080/wsrf/services/gsi/AuthenticationService
[7]: http://127.0.0.1:8080/wsrf/services/TestRPCService
[8]: http://127.0.0.1:8080/wsrf/services/SubscriptionManagerService
[9]: http://127.0.0.1:8080/wsrf/services/ManagementService
[10]: http://127.0.0.1:8080/wsrf/services/TestServiceWrongWSDL
[11]: http://127.0.0.1:8080/wsrf/services/WidgetService
[12]: http://127.0.0.1:8080/wsrf/services/SampleAuthzService
[13]: http://127.0.0.1:8080/wsrf/services/AuthzCalloutTestService
[14]: http://127.0.0.1:8080/wsrf/services/WidgetNotificationService
[15]: http://127.0.0.1:8080/wsrf/services/AdminService
[16]: http://127.0.0.1:8080/wsrf/services/ShutdownService
[17]: http://127.0.0.1:8080/wsrf/services/ContainerRegistryService
[18]: http://127.0.0.1:8080/wsrf/services/CounterService
[19]: http://127.0.0.1:8080/wsrf/services/TestService
[20]: http://127.0.0.1:8080/wsrf/services/TestAuthzService
[21]: http://127.0.0.1:8080/wsrf/services/SecurityTestService
[22]: http://127.0.0.1:8080/wsrf/services/DeploymentService
[23]: http://127.0.0.1:8080/wsrf/services/ContainerRegistryEntryService
[24]: http://127.0.0.1:8080/wsrf/services/NotificationConsumerFactoryService
[25]: http://127.0.0.1:8080/wsrf/services/TestServiceRequest
```

But. . . how do I know it works? How can I test if my service really does anything?

Master: The answer is: we need to write a client. Stop the container with `^C` and follow me.

[exeunt]

Act 2, Scene 4 Client Code

Up the source tree, `org/vlescience/webservice/deployment`.

[Enter Master and Novice]

Master: We already have an `impl` directory; now add a `client` directory and in it place the following file with the name `DeployUrl.java`:

```
package org.vlescience.webservices.deployment.client;
import org.vlescience.webservices.deployment.DeploymentPortType;
import org.vlescience.webservices.deployment.service.DeploymentServiceLocator;
import org.globus.wsrf.client.BaseClient;
import org.apache.axis.types.URI;
import org.apache.commons.cli.ParseException;
import org.apache.commons.cli.CommandLine;
import org.globus.wsrf.utils.FaultHelper;

public class DeployUrl extends BaseClient {

    final static DeploymentServiceLocator locator =
        new DeploymentServiceLocator();

    final static DeployUrl client = new DeployUrl();

    public static void main(String args[]) {
        String serviceurl;

        // first, parse the commandline
        try {
            CommandLine line = client.parse(args);
            serviceurl = line.getOptionValue('s');
            locator.setdeploymentPortTypePortEndpointAddress(serviceurl);
        } catch(ParseException e) {
            System.err.println("Parsing failed: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            System.exit(1);
        }

        try {

            // create the note
            DeploymentPortType port = locator.getdeploymentPortTypePort();
            String out = port.deploy(new URI("http://foo.bar/"));
            System.out.println("The webservice responds: " + out);

        } catch(Exception e) {
            if (client.isDebugMode()) {
                FaultHelper.printStackTrace(e);
            } else {
                System.err.println("Error: " + FaultHelper.getMessage(e));
            }
        }

    }

}
```

See how this code uses the stubs that were created earlier (`DeploymentPortType`, `DeploymentServiceLocator`); the URL of the service (e.g. http://localhost:8080/wsrf/services/DeploymentService) must be given on the commandline with the `-s` flag.

Novice: How do I compile the client?

Master: That can be done by hand, but contrary to the service this requires quite a few jar files on the classpath. It's much nicer to do that with an ant script. We're done here, so let's head back to the base of the source tree and see how we can use ant to our benefit.
[exeunt]

Act 2, Scene 5 Ant Tasks

The root of the source tree.

[enter Master, and Novice]

Novice: I suppose we have to extend the `build.xml` file to include a task for building our client. Unfortunately, Sotomayor didn't include such a task in his build service, so we must write one ourselves.

Master: But It would be unwise to actually change the original build file. There is a much cleaner approach by *importing* the build file in our own; this is supported since ant 1.6.

Novice: The Globus software comes with some handy ant tasks itself; for instance, I see that `globus-deploy-gar` that is nothing but a wrapper around the `deployGar` task in the

`$GLOBUS_LOCATION/share/globus_wsrf_common/build-packages.xml` build file. And the `globus-build-service.sh` does little beyond the setting of a number of key properties right before calling ant with the `all` task. So it makes sense to cut through all the wrapping and use a single central ant build task to control the others.

Master: You can keep your central build file more generic if you take out all the things that are specific for your service and put them in a separate properties file.

Novice: So this is the `buildservice.properties` file; these properties were normally set by `globus-build-service.sh`.

```
package=org.vlescience.webservices.deployment
interface.name=deployment
package.dir=org/vlescience/webservices/deployment
schema.path=vlescience
service.name=DeploymentService
gar.filename=org_vlescience_deployment_service
```

and the new `build.xml` becomes something like this (I've renamed the original file to `buildservice.xml`):

```
<project default="all">

  <property file="buildservice.properties" />
  <property name="clientjar.filename" value="${gar.filename}_client.jar" />

  <import file="buildservice.xml" />

      <target name="compileClient" depends="compileStubs">
      <javac debug="${java.debug}" destdir="${build.dest}" srcdir="${package.dir}/client">
        <include name="**/*.java"/>
        <classpath>
            <pathelement location="${stubs.dest}"/>
            <fileset dir="${build.lib.dir}">
                <include name="*.jar"/>
            </fileset>
            <fileset dir="${globus.location}/lib">
                <include name="*.jar"/>
            </fileset>
        </classpath>
      </javac>
    </target>

    <target name="clientJar" depends="compileClient" >
       <jar jarfile="${lib.dir}/${clientjar.filename}" basedir="${build.dest}">
         <include name="**/${package.dir}/client/**" />
       </jar>
    </target>

    <target name="deploy" depends="clientJar, dist" >
      <ant antfile="${globus.location}/share/globus_wsrf_common/build-packages.xml"
           target="deployGar" >
        <property name="overwrite" value="true" />
      </ant>
    </target>
    <target name="all" depends="clientJar, dist, deploy" />

</project>
```

Master: If you do it like this, the compiled client code will end up in a separate JAR file, that is shipped in the GAR file along with the rest. That is not a problem as long as you're aware that it's not necessary for the proper functioning of the service.

To *use* the client there is a convenient way to automatically create a *launcher* script: create a `post-deploy.xml` file to call the `generateLauncher` ant task; this file is inspected upon deployment of a GAR file. An executable wrapper script for starting your client is placed in `$GLOBUS_LOCATION/bin`. Put it in `org/vlescience/webservices/deployment/etc/post-deploy.xml`.

```
<project default="all" basedir=".">
    <property environment="env"/>
    <property file="build.properties"/>
    <property file="${user.home}/build.properties"/>
    <property name="env.GLOBUS_LOCATION" value="."/>
    <property name="deploy.dir" location="${env.GLOBUS_LOCATION}"/>
    <property name="abs.deploy.dir" location="${deploy.dir}"/>
    <property name="build.launcher"
        location="${abs.deploy.dir}/share/globus_wsrf_common/build-launcher.xml"/>

    <target name="setup">
        <ant antfile="${build.launcher}"
             target="generateLauncher">
            <property name="launcher-name" value="deploy-url"/>
            <property name="class.name"
                value="org.vlescience.webservices.deployment.client.DeployUrl"/>
        </ant>
   </target>
</project>
```

Novice: Let's see this run. . . Oops!

```
BUILD FAILED
/.../src/build.xml:24: Problem creating jar:
   /.../src/org/vlescience/webservices/deployment/lib/org_vlescience_deployment_service_client.jar
   (No such file or directory) (and the archive is probably corrupt but I could not delete it)
```

Master: You need to create the `org/vlescience/webservices/deployment/lib` directory first, silly! To make the build more robust, add a line to `build.xml`:

```
   ...
    <target name="clientJar" depends="compileClient" >
        <mkdir dir="${lib.dir}" />
        <jar jarfile="${lib.dir}/${clientjar.filename}" basedir="${build.dest}">
          <include name="**/${package.dir}/client/**" />
        </jar>
    </target>
   ...
```

Novice: OK, now it builds! Let's see if I can run the client. . .

```
$ deploy-url -s  http://127.0.0.1:8080/wsrf/services/DeploymentService
Error: No such operation 'Deploy'
```

Whoa!? What did I do wrong *this* time?

Master: This is unfortunately a very unhelpful error message, especially since it is very hard to get a little more context of the problem. Even cranking up the verbosity on the container doesn't help.

The meaning of this error message is that there was no method found in the service's implementing class that has the proper prototype. It could be a name mismatch, or a parameter type mismatch.

The true cause is a type mapping issue from the WSDL specified `xsd:anyURI` to a suitable Java type. The Axis documentation [10] mentions that the default mapping is as specified by Sun's JAX-RPC [11], and that would indeed turn out to be `java.net.URI`; however, in reality the mapping goes to
`org.apache.axis.types.URI`
although there is no documentation that would admit the fact.

I probably shouldn't mention this, but adding the flag `-T 1.1` to `WSDL2Java` will generate stubs with the JAX-RPC convention; these stubs don't work. The whole `-T` thing is a bit of a mess [12].

Novice: So if I change my service implementation to

```
package org.vlescience.webservices.deployment.impl;
import org.apache.axis.types.URI;
public class DeploymentService  {
    public java.lang.String deploy(URI gar) {
        // do nothing and assume success
        return "Success.\n";
    }
}
```

It should work? Let's run ant and restart the container. . . and run the client again.

```
$ deploy-url -s  http://127.0.0.1:8080/wsrf/services/DeploymentService
The webservice responds: Success.
```

Hurray!

Master: Congratulations on running your first web service! You are now ready for the real challenge of setting up VOMS authorisation. But look! The Sun is already setting, and the last light of this day will linger but a moment ere it withdraws behind the mountains. Let us retire to our camp, so that a good night's rest refreshes our spirit and in the morning sets us to our task with renewed vigour.
[exeunt]

Act 3 VOMS Authorisation

Act 3, Scene 1 VOMS PDP

A moonlit forest clearing

[Enter Novice]

Novice: I can't find sleep, or 't can't find me. For the foreign sounds of this place are amp'ed by night and scare off the slumber; the call of the `crond`, the crawl of nightly creatures that come above ground when all else is quiet; the squawk of the `awk` and the grinding rotation of logs. The excited anticipation of the discoveries of the morning will let no drowsiness settle on my brow; the dewdrops in the freshly scented air find me among the Perly strings of spiders, and the trails of the hunting Python.

Here by the moonlit trees shall I try to get a head start. Perhaps ere the day breaks shall I master the VOMS PDP without the help of my master.

Freeman and Ananthakrishnan [13] explain how authorisation is handled in the Globus Toolkit, by pluggable Policy Decision Points and Policy Information Points. I have found the VomsPDP as part of the Globus Virtual Workspaces [14] Tech Preview 1.1 ("VOMS plugin"). This I shall first untar and build.

```
$ cd $GLOBUS_LOCATION
$ tar xfz ~/globus_voms_interceptors.tar.gz
$ cd plugins/authz/voms
$ ant
```

Well, that went smoothly! It built and deployed a gar file right away! If everything else goes this smoothly I will be done before sunrise. My master will be so proud!

How to proceed? For the service, all I have to do is explain the authorisation scheme in the `deploy-server.wsdd`. The Globus programmer's tutorial [1] explains the basics, although it doesn't go beyond basic grid authentication.

So I create a security descriptor based on the VOMS PIP/PDP in `org/vlescience/webservices/deployment/etc/security-config.xml` like this:

```
<securityConfig xmlns="http://www.globus.org">
  <auth-method>
    <GSISecureConversation/>
  </auth-method>
  <authz value="ascope:org.globus.voms.VomsCredentialPIP
              bscope:org.globus.voms.VomsPDP"/>
</securityConfig>
```

Those `ascope` and `bscope` thingies are a bit obscure right now but I'm guessing I can't do without. I'll add this descriptor to `deploy-server.wsdd`:

```
<deployment
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service name="DeploymentService"
           provider="Handler"
           style="document"
           use="literal">

    <parameter name="handlerClass"
               value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="className"
               value="org.vlescience.webservices.deployment.impl.DeploymentService"/>
    <parameter name="allowedMethods"
               value="*"/>
    <parameter name="scope"
```

```
                value="Application"/>

    <parameter name="securityDescriptor"
                value="etc/org_vlescience_deployment_service/security-config.xml" />

    <wsdlFile>
      share/schema/vlescience/deployment_service.wsdl
    </wsdlFile>

  </service>
</deployment>
```

I'll redeploy the service, and start the container. Then I'll run the client again.

```
$ deploy-url -s  http://127.0.0.1:8080/wsrf/services/DeploymentService
Error: org.globus.wsrf.config.ConfigException: Failed to initialize security \
config for "DeploymentService" service [Caused by: [Caused by: Failed to load \
PIP/PDP chain; nested exception is:
        java.lang.InstantiationException]]
```

What on earth…I must attempt to figure this one out by myself. Let's see, an instantiation exception could be caused by trying to create an object of an abstract class. Aha! The VomsPDP *does* have an abstract method: `checkCallAndContent`. Apparently I have to write my own derived class and implement this method; The `VomsPDP` calls this method in addition to the normal policies, so the programmer can add his own verifications.

This shouldn't be too hard to cook up. Let's see…

```
package org.vlescience.webservices.deployment.impl.security;
import org.globus.voms.VomsPDP;
import java.util.Vector;
import java.util.Iterator;
import javax.xml.soap.SOAPMessage;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class MyPDP extends VomsPDP {
    static Log logger = LogFactory.getLog(MyPDP.class);
    public Boolean checkCallAndContent(String peerIdentity,
                                       Vector attributes,
                                       String operation,
                                       SOAPMessage content) throws Exception
    {
        Iterator attr = attributes.iterator();
        logger.info("PeerIdentity = " + peerIdentity);
        logger.info("operation = " + operation);
        while (attr.hasNext()) {
            logger.info("attribute: " + (String) attr.next());
        }
        return null;
    }
}
```

The bulk of the method is taken up by the logging code, but in the end `null` is returned which leaves the decisions to the policy defined elsewhere.

I guess I also need to change my security configuration:

```
<securityConfig xmlns="http://www.globus.org">

  <auth-method>
    <GSISecureConversation/>
  </auth-method>

  <authz value="ascope:org.globus.voms.VomsCredentialPIP
                bscope:org.vlescience.webservices.deployment.impl.security.MyPDP"/>

</securityConfig>
```

Now I can rebuild/restart/rerun my client.

```
Error: GSI Secure Conversation authentication required for \
"{http://webservices.vlescience.org/deployment}deploy" operation.
```

Shucks. I forgot to include the secure conversation part in the client. This is just a few lines of code.

```
package org.vlescience.webservices.deployment.client;
import org.vlescience.webservices.deployment.DeploymentPortType;
import org.vlescience.webservices.deployment.service.DeploymentServiceLocator;
import org.globus.wsrf.client.BaseClient;
import org.apache.axis.types.URI;
import org.apache.commons.cli.ParseException;
import org.apache.commons.cli.CommandLine;
import org.globus.wsrf.utils.FaultHelper;
import javax.xml.rpc.Stub;
import org.globus.wsrf.impl.security.authorization.NoAuthorization;
import org.globus.wsrf.security.Constants;

public class DeployUrl extends BaseClient {

    final static DeploymentServiceLocator locator =
        new DeploymentServiceLocator();

    final static DeployUrl client = new DeployUrl();

    public static void main(String args) {
        String serviceurl;

        // first, parse the commandline
        try {
            CommandLine line = client.parse(args);
            serviceurl = line.getOptionValue('s');
            locator.setdeploymentPortTypePortEndpointAddress(serviceurl);
        } catch(ParseException e) {
            System.err.println("Parsing failed: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            System.exit(1);
        }

        try {

            // create the note
            DeploymentPortType port = locator.getdeploymentPortTypePort();
            ((Stub)port)._setProperty(Constants.GSI_SEC_CONV,Constants.ENCRYPTION);
            ((Stub)port)._setProperty(Constants.AUTHORIZATION,NoAuthorization.getInstance());
            String out = port.deploy(new URI("http://foo.bar/"));
            System.out.println("The webservice responds: " + out);

        } catch(Exception e) {
            if (client.isDebugMode()) {
                FaultHelper.printStackTrace(e);
            } else {
                System.err.println("Error: " + FaultHelper.getMessage(e));
            }
        }

    }

}
```

Another rebuild, etc. Now it should work!

Proxy: No it won't. My time is up. [dies]
Novice: My proxy just died! I need it for the secure conversation. I'll do another `voms-proxy-init`.
Proxy: Your wish is my command (for the next twelve hours).
Novice: Run the client!

```
Error: org.globus.wsrf.impl.security.authorization.exceptions.AuthorizationException: \
Policy decision failed [Caused by:  [Caused by: java.lang.NullPointerException]]
```

I am getting *so* frustrated. But ah, I can see the glow of a new dawn coming from the east; my master will be awake soon; he will undoubtedly know what to do. I think that I will take some meditation in the meantime.
[exit]

Act 3, Scene 2 Gridmap Mishap

Meditation area; Novice, asleep.

[Enter Root]

Root: There he sits; weariness has finally overtaken him. I'll let him be for awhile, so that sweet sleep may sooth his worries. The student must have had trouble sleeping, and less so working the problem that was set for today. Had he known the webs that fate had weaved for him, perhaps he would not so restlessly have tangled himself into this mess.

I found his work when I rose at first light and I know the problem he was struggling with; yes, I've been there myself. An unfortunate bug [15] in the VomsPDP code will cause a crash if no ⟨gridmap⟩ was specified in the container's security descriptor. The grid-mapfile can be empty, as it is only checked to allow access to the DNs mentioned therein regardless of VOMS attributes.

I then went to the `$GLOBUS_LOCATION` to make some amends. As it is, the Java WSRF Core does not specify a ⟨parameter⟩ for the container security descriptor in `etc/globus_wsrf_core/server-config.wsdd`, so I added the lines

```
<parameter name="containerSecDesc"
           value="etc/globus_wsrf_core/global_security_descriptor.xml">
</parameter>
```

just below the comment marked `@CONTAINER_SECURITY_DESCRIPTOR@` (h'm, that looks like a leftover Automake variable). The file `global_security_descriptor.xml` is there, but the ⟨gridmap⟩ it contains pointed to `/etc/grid-security/grid-mapfile` and that is Root's turf, so I changed it to

```
<gridmap value="etc/globus_wsrf_core/grid-mapfile"/>
```

and I created an empty file by that name. That should keep the VomsPDP from crashing. Also, I commented out the ⟨credential⟩ element, which refer to the non-existent `containerkey.pem` and `containercert.pem`. Enabling the container security descriptor has the consequence that the container needs credentials to run with. Without the ⟨credential⟩ element, it will fall back on the proxy credentials of whoever runs the container.
[Novice awakens]

Novice: Oh, hello. Have you been standing there long? I must've dozed off.

Master: Good morrow, my young apprentice. I trust you are well rested?

Novice: Actually, I couldn't sleep last night so I tried to get a head start. But things soon got bungled; the fogs of the night must have clouded my mind, and the Sun has caught up with me. Let me show you what I've been trying to do.
[exeunt]

Act 3, Scene 3 The Final Strands

The source tree

[Enter Novice and Master]

Novice: See? The security descriptor is in place, the secure conversation is set up and the Voms PDP is installed; but I keep getting a Null Pointer Exception.

Master: You can't make the web service container behave without a master's eye watching over it. Now restart the container, and rerun the client.

Novice: But I tried, and it didn't work. [restarts the container]

Master: Nevertheless, try again.

Novice: OK.

```
$ deploy-url -s  http://127.0.0.1:8080/wsrf/services/DeploymentService
Error: org.globus.wsrf.impl.security.authorization.exceptions.AuthorizationException: \
"/O=dutchgrid/O=users/O=vlescience/CN=Novice" is not authorized to use operation: \
{http://webservices.vlescience.org/deployment}deploy on this service
```

I don't understand... It worked, sort of.

Master: You still have much to learn, my padawan. But what you see here is indeed the working VomsPDP. Now we shall try to add the right attribute checks to the configuration, so that the TEST VO membership will authorise the use of this service.

We're actually not that far off. In the security descriptor, you've already scoped the configuration of the PIP and the PDP with `ascope:` and `bscope:`. We can now pass settings to the PDP by setting the proper ⟨parameter⟩s in `deploy-server.wsdd`. Let's keep it simple by allowing access to anyone whose attributes match what's in the file `attr-authz`:

```
/TEST/Role=NULL/Capability=NULL
```

We'll put this file in the `etc` directory and refer to it from our WSDD:

```
<deployment
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service name="DeploymentService"
           provider="Handler"
           style="document"
           use="literal">

    <parameter name="handlerClass"
               value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="className"
               value="org.vlescience.webservices.deployment.impl.DeploymentService"/>
    <parameter name="allowedMethods"
               value="*"/>
    <parameter name="scope"
               value="Application"/>

    <parameter name="securityDescriptor"
               value="etc/org_vlescience_deployment_service/security-config.xml" />

    <parameter name="bscope-attrAuthzConfigFile"
               value="etc/org_vlescience_deployment_service/attr-authz" />

    <wsdlFile>
      share/schema/vlescience/deployment_service.wsdl
    </wsdlFile>

  </service>
</deployment>
```

The `bscope-` prefix will cause the new ⟨parameter⟩ to appear only to `MyPDP`.

Novice: I see. Are there any more ⟨parameter⟩s like those?

Master: There certainly are; you are welcome to inspect the `VomsPDP` source code if you like.

Some other ⟨parameter⟩s are applicable with the prefix `vomsPdp-` to override certain defaults, like `voms-hostport`, `vomsTrustStore`, `validate` and `vomsRefreshTime`, as can be found in
`VomsCredentialPIP.java`,
but the defaults serve us fine. Just run `ant`, restart the container and run the client once more.

Novice: Here we go...

```
$ deploy-url -s  http://127.0.0.1:8080/wsrf/services/DeploymentService
The webservice responds: Success.
```

That's great, but how do I know it *really* worked?

Master: We need to add a bit of code in the service to print some debugging information about our security context. If we're able to uncover the VO membership and attributes, let's call it a success.

Where to look for this data? For now, the answer is: in the MessageContext. That is where the VomsCredentialPIP has stored the VomsCredentialInformation as a public credential. The VomsPDP sources show how to retrieve that information.

Novice: I see... But how do we get the MessageContext?

Master: the MessageContext is accessible through the Axis API. The Axis User's Guide [10] mentions that
`org.apache.axis.MessageContext`
is the place to look for almost *anything*; use

```
MessageContext ctx = MessageContext.getCurrentContext();
```

to get the context and then cleverly copy code from VomsPDP.java to get what you want.

Novice: This form of software cloning has an unwholesome look to it; isn't there a cleaner API for this?

Master: Unfortunately, no. Or I should say: not yet. The VomsPDP was written after the interface for GT4.0 was frozen, but the next major release will indeed have a different interface for attribute storage [16].

Novice: So this is what I end up with:

```
package org.vlescience.webservices.deployment.impl;
import org.apache.axis.types.URI;
import org.apache.axis.MessageContext;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.globus.wsrf.security.authorization.attributes.AttributeInformation;
import org.globus.wsrf.security.SecurityManager;
import org.globus.wsrf.impl.security.authentication.Constants;
import org.globus.gsi.jaas.JaasSubject;
import org.globus.voms.VomsCredentialInformation;
import javax.security.auth.Subject;
import java.util.Set;
import java.util.Iterator;
import java.util.Vector;

public class DeploymentService  {
    static Log logger = LogFactory.getLog(DeploymentService.class);

    public java.lang.String deploy(URI gar) {
        // do nothing and assume success
        logCredentials();
        return "Success.\n";
    }

    private void logCredentials() {
        Subject peer = null;
        VomsCredentialInformation vomsinfo = null;
        AttributeInformation info = null;
        Vector rolesVector = null;

        MessageContext ctx = MessageContext.getCurrentContext();
        peer = (Subject) ctx.getProperty(Constants.PEER_SUBJECT);
        Set credSet = peer.getPublicCredentials();
        Iterator creds = credSet.iterator();

        while (creds.hasNext()) {
            Object o = creds.next();
            if (o instanceof AttributeInformation) {
                info = (AttributeInformation) o;
                break;
            }
        }

        if (info == null) {
            logger.info("cannot retrieve credential info from message context");
        } else {
            if (!(info instanceof VomsCredentialInformation)) {
                logger.warn("credential info from message context is not VOMS");
            } else {
                vomsinfo = (VomsCredentialInformation) info;
            }
        }

        if (vomsinfo != null) {
            String VO = vomsinfo.getVO();
            logger.info("VO " + VO);
            rolesVector = vomsinfo.getAttrs();

            for (int i=0; i<rolesVector.size(); i++) {
                logger.info("Roles " + rolesVector.get(i));
            }
        }
    }


}
```

Master: Very well; if you deploy this service and restart the container, you can see the results by setting the log level of `org.vlescience` to INFO in `$GLOBUS_LOCATION/container-log4j.properties`:

```
log4j.category.org.vlescience=INFO
```

[Redeploy, restart, wash, rinse, repeat]

```
 ... INFO  impl.DeploymentService [ ... ] VO TEST
 ... INFO  impl.DeploymentService [ ... ] Roles /TEST/Role=NULL/Capability=NULL
 ... INFO  impl.DeploymentService [ ... ] Roles /TEST/tmp/Role=NULL/Capability=NULL
```

Novice: Hurray! Does this mean my training is finally over?
Master: Well. . . you still have much to learn. But not today; the lesson is over.
Novice: Whew! This has been a truly exhausting experience, but all's well that ends well. I now know a little more about Globus security, `ant`, web services and Axis. I surely couldn't have done this without your help!
Master: Stand on the shoulders of giants, we do, h'm?
Novice: Now you talk funny again.
Master: Come. We will leave now in search of another challenge.
Novice: Where are you going? Shouldn't we be break up camp? Leave things neat and tidy, the way we found it? Root is going to be horribly cross if we don't.
Master: Nay, don't worry about it. The place will be torn up after we leave. That's one of the comforts of using a virtual machine.
[exeunt]

THE END

Epilogue

The examples found here can be downloaded as a tarball from http://www.nikhef.nl/~dennisvd/.

With gratitude to Oscar Koeroo, the VOMS administrator with whom I share a room. Without his help it would have been much harder to grasp the whole VOMS thing.

References

[1] Borja Sotomayor, The Globus Toolkit 4 Programmer's Tutorial, http://gdp.globus.org/gt4-tutorial/singlehtml/progtutorial_0.2.1.html

[2] The Dutchgrid Certification Authority, http://ca.dutchgrid.nl/

[3] The Globus Toolkit 4, http://www.globus.org/toolkit/downloads/4.0.2/#wscore_bin

[4] SUN Java 2 Platform, Standard Edition, v. 1.4.2 (J2SE) http://java.sun.com/j2se/1.4.2/download.html

[5] Loading a Certificate onto your Browser http://register.matrix.sara.nl/information/cert.html

[6] GT 4.0 Java WS Core System Administrator's Guide, Usage Statistics Configuration http://www.globus.org/toolkit/docs/4.0/common/javawscore/admin-index.html #s-javawscore-Interface_Config_Frag-usageStatisticsTargets

[7] The JPackage Project, http://www.jpackage.org

[8] The Globus Build Service, http://gsbt.sourceforge.net/

[9] Russell Butek (butek@us.ibm.com), Which style of WSDL should I use? http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/

[10] The Apache Axis User's Guide, http://ws.apache.org/axis/java/user-guide.html

[11] Java API for XML-Based RPC (JAX-RPC), http://java.sun.com/xml/jaxrpc/

[12] Inconsistent/garbled statements about WSDL2Java's `-T --typeMappingVersion` option in Axis Reference Guide, http://issues.apache.org/jira/browse/AXIS-2467.

[13] Tim Freeman and Rachana Ananthakrishnan, Authorization processing for Globus Toolkit Java Web services, http://www-128.ibm.com/developerworks/grid/library/gr-gt4auth/

[14] Globus Virtual Workspaces, http://workspace.globus.org/downloads/index.html

[15] Bugzilla Bug 4392. Missing gridmapfile will crash the VomsPDP
http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4392

[16] Tim Freeman (on the gt-user mailing list) Re: How to get to the VOMS attributes, properly
http://www.globus.org/mail_archive/gt-user/2006/05/msg00106.html