Lời Nói Đầu

Tài liệu này được thực hiện sau một thời gian làm việc về Tính toán lưới của nhóm nghiên cứu Grid Computing tại trung tâm Tính toán hiệu năng cao - trường Đại học Bách Khoa Hà Nội.

Trong tài liệu này, chúng tôi xin trình bày chi tiết quá trình cài đặt và lập trình phát triển dịch vụ trên hệ thống BKGrid2005. Nội dung của tài liệu được chia làm 2 chương:

Chương 1: Hướng dẫn cài đặt hệ thống BKGrid2005. Chương này trình bày chi tiết quá trình cài đặt, cấu hình một hệ thống tính toán lưới từ các phần mềm mã nguồn mở: hệ điều hành Linux, JDK, Ant, Globus, Tomcat, GridSphere, JADE, PBS.

Chương 2: Hướng dẫn lập trình dịch vụ lưới trên GT3. Chương này trình bày các khái niệm cơ sở, các bước của quá trình triển khai một dịch vụ trên lưới tính toán.

Mục lục

Chương 1: Hướng dẫn cài đặt hệ thống BKGrid2005

1.1. Cài đặt các thành phần hỗ trợ cho GT3.2.1

1.1.1. Cài đặt JDK

1.1.1.1. Giới thiệu

JDK (Java Development Kit) là môi trường cho các ứng dụng Java, để cài đặt máy ảo Java (Java Virtual Machine - JVM) và thông dịch cho các ứng dụng phát triển trên môi trường Java.

Có thể download bản mới nhất của J2SE (Java 2 Platform, Standard Edition) tại địa chỉ http://java.sun.com/j2se/index.jsp.

Chúng tôi dùng bản 1.4.2_04 (gói j2sdk-1_4_2_04-nb-3_6-bin-linux.bin) hỗ trợ cho Globus-Tookit, bao gồm j2sdk-1.4.2_04 và java-bin-3.6.

1.1.1.2. Các bước cài đặt

Bước 1: chọn 1 trong 2 cách sau

Cách 1:

- Download gói j2sdk-1_4_2_04-nb-3_6-bin-linux.bin vào /usr/local tại địa chỉ

http://java.sun.com/j2se/index.jsp

- Vào thư mục chứa bản vừa download (/usr/local/) để cài đặt, dùng các lệnh:

```
# cd /usr/local
# ./j2sdk-1_4_2_04-nb-3_6-bin-linux.bin
```

- Tiếp tục làm theo các chỉ dẫn trên màn hình để hoàn thành cài đặt.

Cách 2: Copy thư mục j2sdk1.4.2_04 trong /mnt/hue_data/LinuxSetup/GT3/ vào /usr/local/, dùng lệnh:

```
# cp /mnt/hue_data/LinuxSetup/GT3/j2sdk1.4.2_04 /usr/local
```

Bước 2: Thiết lập biến môi trường

- Thêm các dòng sau vào cuối tệp /etc/profile

```
JAVA_HOME=/usr/local/j2sdk1.4.2_04
PATH=$PATH:$JAVA_HOME/bin
export JAVA_HOME PATH
```

- Cập nhật biến môi trường, dùng lệnh:

```
# . /etc/profile
```

Kiểm tra xem cài đặt có thành công không, dùng lệnh:

```
# java -version
```

Nếu thấy xuất hiện thông báo kiểu như

```
java version "1.4.2_04"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_04-b05)
Java HotSpot(TM) Client VM (build 1.4.2_04-b05, mixed mode)
```

thì quá trình cài đặt đã thành công.

1.1.2. Ant

1.1.2.1. Giới thiệu

Ant là công cụ được xây dựng trên nền Java, cần cho việc cài đặt GT3. Nó bao gồm nhiều lớp java, dùng để biên dịch hay build các chương trình. Ant được sử dụng để chạy các Ant-based build script và qua đó tự động hóa tiến trình cài đặt (GT3 cần để re-build tại bất cứ thời điểm nào).

Chúng tôi sử dụng bản Apache-ant-1.6.5 để hỗ trợ cho GT3.

1.1.2.2. Các bước cài đặt

Bước 1: chọn 1 trong 2 cách sau

Cách 1:

- Download bản Apache-ant-1.6.5-bin.tar.gz (khoảng 7.7 MB) vào /usr/local tại địa chỉ:

http://ant.apache.org/srcdownload.cgi

- Vào thư mục chứa bản vừa download (/usr/local) để giải nén, dùng các lệnh sau:

```
# cd /usr/local
# tar -xzvf apache-ant-1.6.5-bin.tar.gz
```

Cách 2:

- Copy thư mục apache-ant-1.6.5 ở /mnt/hue_data/LinuxSetup/GT3/ vào /usr/local/, dùng lệnh:

```
# cp /mnt/hue_data/LinuxSetup/GT3/apache-ant-1.6.5 /usr/local
```

Bước 2: Thiết lập biến môi trường

- Thêm các dòng sau vào cuối tệp /etc/profile

```
ANT_HOME=/usr/local/apache-ant-1.6.5
PATH=$PATH:$ANT_HOME/bin
export ANT_HOME PATH
```

- Cập nhật biến môi trường, dùng lệnh:

```
# . /etc/profile
```

Kiểm tra xem cài đặt có thành công không, dùng lệnh:

```
# ant -version
```

Nếu thấy xuất hiện thông báo kiểu như

```
Apache Ant version 1.6.5 compiled on June 2 2005
```

thì quá trình cài đặt đã thành công.

1.2. Cài đặt và cấu hình GT3.2.1

1.2.1. Giới thiệu

Bộ công cụ Globus Toolkit là sản phẩm của Globus – dự án phát triển các giải pháp cho tính toán mạng lưới với sự hợp tác của nhiều công ty Mỹ. Globus Toolkit là tập hợp các dịch vụ lưới, phần mềm lưới, các thư viện xây dựng theo kiến trúc mở.

Chúng tôi dùng bản GT3.2.1 để làm nền tảng xây dựng ứng dụng BKGrid2005.

1.2.2. Các bước cài đặt GT3.2.1

- Chú ý:

+ Trước khi cài đặt, ta cần có các user sau:

UserId	GroupId	Mô tả
root	root	Quyền cao nhất
globus	globus	Chủ môi trường GT
user	user	Người sử dụng lưới

+ Qui trình cài đặt sau được dùng cho mayxx, chủ của lưới là globus_xx

- Các bước cài đặt:

Bước 1: Cài đặt GT3.2.1

- Tạo thư mục gt3.2.1:

```
# mkdir /usr/local/gt3.2.1
```

- Thiết lập biến môi trường cho globus: thêm các dòng sau vào cuối tệp /etc/profile

```
# for GT3.2.1
GLOBUS_LOCATION=/usr/local/gt3.2.1
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$GLOBUS_LOCATION/lib
```

```
PATH=$PATH:$GLOBUS_LOCATION/bin
export GLOBUS_LOCATION LD_LIBRARY_PATH PATH
```

- Cập nhật các biến môi trường, dùng lệnh:

```
# . /etc/profile
```

- Download gói gt3.2.1-all-linux-glibc2.3-installer.tar.gz (khoảng 93MB) vào /usr/local/gt3.2.1 tại địa chỉ http://www-unix.globus.org/toolkit/downloads/3.2.1/ hoặc có thể copy từ thư mục /mnt/hue_data/LinuxSetup/GT3, dùng lệnh:

```
# cp /mnt/hue_data/LinuxSetup/GT3/gt3.2.1-all-linux-glibc2.3-installer.tar.gz $GLOBUS_LOCATION
```

- Chuyển quyền sở hữu các tệp trong thư mục $GLOBUS_LOCATION cho globus_xx

```
# chown -R globus_xx $GLOBUS_LOCATION
```

- Chuyển sang user globus_xx

```
# su globus_xx
```

Ta bắt đầu thực hiện cài đặt GT3 trên quyền chủ GT globus_xx.

- Vào thư mục cài GT3:

```
[globus_xx] cd $GLOBUS_LOCATION
```

- Thực hiện giải nén:

```
[globus_xx] tar -xzvf gt3.2.1-all-linux-glibc2.3-installer.tar.gz
```

- Chuyển vào thư mục vừa giải nén

```
[globus_xx] cd gt3.2.1-all-linux-glibc2.3-installer
```

- Thực hiện lệnh sau và đợi khoảng 20 phút để kết thúc cài đặt GT3.2.1

```
[globus_xx] ./install-gt3-bin $GLOBUS_LOCATION
```

Bước 2: Cài đặt simpleCA

Giới thiệu

SimpleCA là một gói cung cấp cơ chế chứng thực uỷ quyền cho các người dùng và các dịch vụ trong lưới.

Trên một lưới, ta sẽ cài một nhà CA (tạm gọi là chủ CA) để chứng thực cho các thành viên tham gia vào lưới. Nhà CA này sẽ ký các giấy chứng nhận cho các thành phần tham gia vào lưới.

Cài đặt

- Chuyển sang quyền root, dùng lệnh:

```
[globus_xx] exit
```

- Download gói globus_simple_ca-latest-src_bundle.tar.gz (khoảng 548KB) vào thư mục $GLOBUS_LOCATION tại địa chỉ:

http://www-unix.globus.org/ftppub/gsi/simple_ca/globus_simple_ca-latest-src_bundle.tar.gz

Hoặc cũng có thể copy từ thư mục /mnt/hue_data/LinuxSetup/GT3, dùng lệnh:

```
# cp /mnt/hue_data/LinuxSetup/GT3/globus_simple_ca-latest-src_bundle.tar.gz $GLOBUS_LOCATION
```

- Tạo quyền cho globus_xx

```
# chown -R globus_xx $GLOBUS_LOCATION
```

- Chuyển sang user globus_xx

```
# su globus_xx
```

- Sang thư mục cài GT3

```
[globus_xx] cd $GLOBUS_LOCATION
```

- Thực hiện lệnh sau để build simpleCA:

```
[globus_xx] $GLOBUS_LOCATION/sbin/gpt-build globus_simple_ca-latest-src_bundle.tar.gz gcc32dbg
```

- Cài đặt simpleCA, dùng lenh:

```
[globus_xx] $GLOBUS_LOCATION/sbin/gpt-postinstall
```

Chú ý: Sau lệnh này, thư mục /home/globus_group/globus_xx/.globus/simpleCA sẽ được tạo ra với đầy đủ các tệp:

+ cacert.pem – đây là public CA certificate của CA

+ globus_simple_ca_XXXXXXXX_setup-0.13.tar.gz

+ grid-ca-ssl.conf

+ index.txt

+ serial

và các thư mục

+ certs: rỗng

+ crl: rỗng

+ newcerts: rỗng

+ private:

. cakey.pem – đây là private key của CA.

- Thực hiện các bước tiếp như bảng sau:

Hỏi	Đáp

Hỏi	Đáp
Do you want to keep this as the CA subject (y/n) [y]:	Y
Enter a unique subject name for this CA:	Enter
Enter the email of the CA (this is the email where certificate requests will be sent to be signed by the CA):	nhattanht@yahoo.com
[default: 5 years (1825 days)]	Enter
Enter PEM pass phrase:	parallelsc
Verifying password - Enter PEM pass phrase:	parallelsc

Chú ý: Nếu bạn muốn biết địa chỉ file của simpleCA ở đâu, chạy lệnh sau:

```
grid-cert-info -file ~/.globus/simpleCA/cacert.pem
```

- Có vài dòng thông báo để cài đặt GSI ở cuối quá trình cài đặt

```
Note: To complete setup of the GSI software you need to run the
following script as root to configure your security configuration
directory:

$GLOBUS_LOCATION/setup/globus_simple_ca_24d355a5_setup/setup-gsi
```

- Chuyển sang quyền root

```
[globus_xx] exit
```

- Copy dòng lệnh trên để cài đặt GSI.

```
# $GLOBUS_LOCATION/setup/globus_simple_ca_XXXXXXXX_setup/setup-gsi
```

Chú ý: sau lệnh này, thư mục /etc/grid-security sẽ được tạo ra với thư mục của trusted CA là certificate, trong đó có các tệp

+ XXXXXXXX.0

+ XXXXXXXX.signing_policy

+ globus-host-ssl.conf.XXXXXXXX

+ globus-user-ssl.conf.XXXXXXXX

+ grid-security.conf.XXXXXXXX

Làm tiếp như bảng hướng dẫn sau:

Hỏi	Đáp
Do you wish to continue (y/n) [y] :	y
(1) Base DN for user certificates [ou=, ou=GT3 Tutorial, o=Globus] (2) Base DN for host certificates [ou=GT3 Tutorial, o=Globus] ========== (q) save, configure the GSI and Quit (c) Cancel (exit without saving or configuring) (h) Help	q

- Thiết lập default-CA, dùng lệnh:

```
# $GLOBUS_LOCATION/bin/grid-default-ca
```

Gặp thông báo sau, nhấn phím 1 rồi Enter

```
Enter the index number of the CA to set as the default:
```

Chú ý : Trong trường hợp có nhiều nhà CA thì phải chọn đúng nhà CA tương ứng với số hiệu mảng băm xxxx ở gói globus_simple_ca-xxxx_setup-*.tar.gz mà quá trình cài đặt sinh ra. Gói này để ở /home/globus_group/globus_xx/.globus/simpleCA/globus_simple_ca-xxxx_setup-*.tar.gz

Có 3 tệp sẽ được tạo ra trong thư mục /etc/grid-security/, đó là:

/etc/grid-security/grid-security.conf

/etc/grid-security/globus-host-ssl.conf

/etc/grid-security/globus-user-ssl.conf

- Xin một giấy chứng nhận cho máy tham gia lưới (ở đây là mayxx), dùng lệnh:

```
# grid-cert-request -host mayxx
```

Có 3 tệp được tạo ra, đó là: hostcert.pem (empty), hostcert-request.pem và hostkey.pem trong thư mục /etc/grid-security

+ Copy tệp /etc/grid-security/hostcert_request.pem sang /tmp

```
# cp -f /etc/grid-security/hostcert_request.pem /tmp
```

Dưới quyền của globus_xx, ký lên file này, dùng các lệnh:

```
# su globus_xx
[globus_xx] cd /tmp
[globus_xx] grid-ca-sign -in hostcert_request.pem -out hostcert.pem
```

Chú ý:

+ CA lưu 1 bản trong /home/globus_group/globus_xx/.globus/newcerts

+ Nếu trong môi trường mạng thực sự phải chuyển tệp này đến cho CA ký.

+ Dưới quyền root, copy đè /tmp/hostcert.pem vào /etc/grid-security/hostcert.pem.

```
[globus_xx] exit
# cp -f /tmp/hostcert.pem /etc/grid-security/hostcert.pem
```

+ Tạo quyền cho root, dùng lệnh:

```
# chown root /etc/grid-security/hostcert.pem
```

- Đăng ký cho người dùng: ở đây ta thực hiện đăng ký cho root. Việc đăng ký cho các người dùng khác là hoàn toàn tương tự.

+ Thực hiện xin một giấy chứng nhận dưới quyền root

```
# grid-cert-request -cn rootxx
```

Thư mục /root/.globus sẽ được tạo ra với 3 tệp: usercert.pem (empty), usercert-request.pem và userkey.pem

Nếu là 1 user bình thường, 3 tệp trên sẽ được lưu trong thư mục ~$USER/.globus

+ Copy tệp /root/.globus/usercert_request.pem sang /tmp

```
# cp -f /root/.globus/usercert_request.pem  /tmp
```

+ Dưới quyền globus_xx, vào thư mục /tmp và ký nhận cho user

```
# su globus_xx
[globus_xx] cd /tmp
[globus_xx] grid-ca-sign -in usercert_request.pem -out usercert.pem
```

Chú ý: CA lưu 1 bản trong /home/globus_group/globus_xx/.globus/newcerts

+ Dưới quyền root, copy file /tmp/usercert.pem đè lên /root/.globus/usercert.pem

```
[globus_xx] exit
# cp -f /tmp/usercert.pem /root/.globus
```

- Tạo quyền cho root, dùng lệnh:

```
# chown root /root/.globus/usercert.pem
```

- Kiểm tra xem simpleCA có được cài đặt thành công không, dùng lệnh:

```
# grid-proxy-init -debug -verify
```

Chú ý: Nếu cài đặt cho user khác root thì chạy lệnh này dưới dấu nhắc của user đó.

Nếu thành công, bạn sẽ thấy trên màn hình như sau:

```
User Cert File: /root/.globus/usercert.pem
User Key File: /root/.globus/userkey.pem

Trusted CA Cert Dir: /etc/grid-security/certificates

Output File: /tmp/x509up_u0
Your identity: /O=Grid/OU=GlobusTest/OU=simpleCA-
may40.linux.hpc.hut.edu.vn/OU=linux.hpc.hut.edu.vn/CN=root
Enter GRID pass phrase for this identity:
Creating proxy ...++++++++++++
.....++++++++++++
 Done
Proxy Verify OK
Your proxy is valid until: Sat Dec 10 02:45:13 2005
```

Sau lệnh này, proxy của user (trong trường hợp này là root) sẽ được tạo ra, đó là tệp /tmp/x509up_u0

- Dưới quyền root, chạy script sau để cập nhật quyền cho 1 số tệp trong $GLOBUS_LOCATION/bin, nó cho phép công cụ quản lý tài nguyên của GT cso quyền như là root:

```
# $GLOBUS_LOCATION/bin/setperms.sh
```

- Chạy các lệnh:

```
# grid-cert-info -subject
```

Hiện ra thông báo, kiểu như:

```
/O=Grid/OU=GlobusTest/OU=simpleCA-
mayxx.linux.hpc.hut.edu.vn/OU=linux.hpc.hut.edu.vn/CN=rootxx
```

Chạy tiếp lệnh:

```
# whoami
```

Hiện ra kết quả, kiểu như:

```
root
```

- Dùng một trình soạn thảo văn bản thuần text (chẳng hạn gedit) tạo /etc/grid-security/grid-mapfile, có nội dung dựa vào kết quả hai lệnh trên:

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-
mayxx.linux.hpc.hut.edu.vn/OU=linux.hpc.hut.edu.vn/CN=rootxx" root
```

Chú ý: toàn bộ thông tin trên chỉ ở một dòng.

Đến đây, bạn đã hoàn thành cài đặt GT3.2.1.

Để chạy GT3, thực hiện như sau:

- Dưới quyền root chạy lệnh:

```
# xhost +
```

- Dưới quyền globus_xx chạy lệnh:

```
[globus_xx] globus-start-container
```

GT chạy trên cổng mặc định là 8080 (có thể thay đổi cổng bằng tham số -p)

- Dưới quyền user chạy

```
$ globus-service-browser
```

User của lưới là globus_xx; root hay 1 user khác tùy thuộc quá trình cài đặt CA ở trên.

Cài đặt simpleCA cho các máy còn lại trong lưới

- Các bước thực hiện hoàn toàn tương tự như cài đặt simpleCA ở bên trên, chỉ có điểm khác biệt là sử dụng gói: globus_simple_ca-xxxx_setup-*.tar.gz trong /home/globus_location/globus_xx/.globus/simpleCA. Trong đó xxxx là dãy số đại diện cho mỗi nhà CA vừa cài (trên máy xx). Cụ thể là:

Nếu muốn sử dụng certificate trên một máy khác, phải cài gói CA trên máy đó.

- Copy gói globus_simple_ca-xxxx_setup-*.tar.gz tới máy muốn cài.

- Dùng lệnh $GLOBUS_LOCATION/sbin/gpt-build globus_simple_ca_xxxxx_setup-0.17.tar.gz gcc32dbg và tiến hành tương tự như trên.

Chú ý: Do trong khi cài đặt có thể máy mà ta đang cài thiếu gói gì đó nên bạn phải chú ý làm theo các chỉ dẫn trong quá trình cài đặt.

1.3. Cài đặt kho lưu trữ giấy ủy nhiệm MyProxy.

Bước 1: Cài đặt gói myproxy

- Tải gói myproxy*.tar.gz vào thư mục $GLOBUS_LOCATION tại địa chỉ ftp://ftp.ncsa.uiuc.edu/aces/myproxy/, chẳng hạn myproxy-3.4.tar.gz (748KB)

- Chuyển vào thư mục $GLOBUS_LOCATION, cài đặt myproxy, dùng các lệnh sau:

```
# cd $GLOBUS_LOCATION
# $GLOBUS_LOCATION/sbin/gpt-build -verbose myproxy*.tar.gz gcc32dbg
```

- Thiết lập các biến môi trường người dùng, dùng lệnh sau:

```
# . $GLOBUS_LOCATION/etc/globus-user-env.sh
```

Bước 2: Cấu hình máy chủ MyProxy

- Copy file myproxy-server.config từ $GLOBUS_LOCATION/share vào $GLOBUS_LOCATION/etc/, dùng lệnh:

```
# cp $GLOBUS_LOCATION/share/myproxy/myproxy-server.config
$GLOBUS_LOCATION/etc/
```

- Thêm các dòng sau váo cuối tệp $GLOBUS_LOCATION/etc/myproxy-server.config

```
accepted_credentials "*"
authorized_retrievers "*"
default_retrievers "*"
authorized_renewers "*"
default_renewers "none"
authorized_key_retrievers "*"
default_key_retrievers "none"
trusted_retrievers "*"
default_trusted_retrievers "none"
```

- Thiết lập các biến môi trường:

+ Thêm các dòng sau vào cuối tệp /etc/profile

```
MYPROXY_SERVER=mayxx
MYPROXY_SERVER_DN="/O=Grid/OU=GlobusTest/OU=simpleCA-
mayxx.linux.hpc.hut.edu.vn/CN=host/mayxx"
export MYPROXY_SERVER MYPROXY_SERVER_DN
```

Chú ý: giá trị của biến MYPROXY_SERVER_DN chính là nội dung của Certificate Subject trong tệp /etc/grid-security/hostcert_request.pem

+ Cập nhật biến môi trường, dùng lệnh :

```
# . /etc/profile
```

- Kiểm tra quá trình cài đặt có thành công không:

+ Chạy lệnh sau để khởi động myproxy-server

```
# $GLOBUS_LOCATION/sbin/myproxy-server
```

Có kết nối TCP tại cổng 7512

- Thử xin một giấy chứng nhận

```
# myproxy-init
```

Chú ý: Nếu muốn chạy myproxy-server như một dịch vụ hệ thống (tự động khởi tạo khi bật máy) thì làm theo các bước sau:

+ Copy $GLOBUS_LOCATION/share/myproxy/etc.init.d.myproxy tới thư mục /etc/rc.d/init.d/myproxy

```
# cp $GLOBUS_LOCATION/share/myproxy/etc.init.d.myproxy
etc/rc.d/init.d/myproxy
```

+ Sau đó chạy lệnh

```
# chkconfig --add myproxy
```

1.4. Cài đặt máy chủ Tomcat

- Tải về phiên bản mới nhất của Tomcat vào /usr/local/ tại địa chỉ

http://tomcat.apache.org/download-41.cgi

Chúng tôi dùng phiên bản 4.1.31 (gói jakarta-tomcat-4.1.31.tar.gz, dung lượng 7.7MB)

- Vào /usr/local, giải nén tệp vừa tải về, dùng các lệnh:

```
# cd /usr/local
# tar -xzvf jakarta-tomcat-*.tar.gz
```

- Thiết lập biến môi trường CATALINA_HOME:

+ Thêm các dòng sau vào cuối tệp /etc/profile

```
# for TOMCAT web server
CATALINA_HOME=/usr/local/jakarta-tomcat-*
PATH=$PATH:$CATALINA_HOME/bin
export CATALINA_HOME PATH
```

Chú ý: Thay * trong dòng thứ hai bằng số hiệu phiên bản mà bạn dùng để cài đặt,chẳng hạn 4.1.31

+ Cập nhật biến môi trường, dùng lệnh:

```
# . /etc/profile
```

- Kiểm tra xem quá trình cài đặt thành công không:

+ Khởi động máy chủ Tomcat, dùng lệnh:

```
# startup.sh
```

Tomcat được khởi động ở cổng 8080.

+ Chờ khoảng 30 giây, mở một trình duyệt web, gõ vào địa chỉ: http://localhost:8080/

(Chú ý thiết lập noproxy cho localhost trong trình duyệt)

Nếu hiện ra giao diện Apache-Tomcat thì quá trình cài đặt đã thành công.

+ Sau khi đã khởi động Tomcat, có thể tắt nó bằng lệnh

```
# shutdown.sh
```

1.5. Cài đặt Portal Gridsphere

- Tải về phiên bản mới nhất của Gridsphere vào /usr/local từ địa chỉ

http://www.gridsphere.org/gridsphere/gridsphere

Chúng tôi dùng phiên bản 2.1.2 (19/12/2005 - gói gridsphere-2.1.2-src.tgz, 9.4MB)

- Vào thư mục /usr/local, giải nén gói vừa tải về, dùng các lệnh:

```
# cd /usr/local
# tar -xzvf gridsphere-2.0.2-src.tgz
```

Vào thư mục vừa giải nén, thực hiện cài đặt bằng các lệnh:

```
# cd gridsphere-2.0.2
# ant install
```

Chờ khoảng 5 phút để quá trình cài đặt kết thúc. Việc cài đặt sẽ tạo ra thư mục gridsphere trong $CATALINA_HOME/webapps

Kiểm tra cài đặt thành công hay không:

+ Khởi động máy chủ Tomcat, dùng lệnh:

```
# startup.sh
```

+ Mở một trình duyệt Web, gõ địa chỉ

http://localhost:8080/gridsphere/

(Chú ý thiết lập noproxy cho localhost trong trình duyệt web)

Nếu hiện ra giao diện của Gridsphere thì quá trình cài đặt đã thành công.

1.6. Cài đặt JADE

- Tạo thư mục /usr/local/jade

```
# mkdir /usr/local/jade
```

- Tải về phiên bản mới nhất của JADE (20/12/2005, phiên bản 3.3, gói JADE-bin-3.3.zip, 1.5MB) vào /usr/local/jade từ địa chỉ http://jade.tilab.com/

- Vào thư mục /usr/local/jade, giải nén tệp vừa tải về, dùng các lệnh:

```
# cd /usr/local/jade

# unzip  JADE-bin-3.3.zip
```

- Copy tất cả các file trong thư mục usr/local/jade/jade/lib vào $GLOBUS_LOCATION/lib để GT nhận JADE

```
# cp  /usr/local/jade/jade/lib/* $GLOBUS_LOCATION/lib
```

- Thiết lập biến môi trường CLASSPATH:

+ Thêm các dòng sau vào cuối tệp /etc/profile

```
# for JADE

CLASSPATH=$CLASSPATH:/usr/local/jade/jade/lib/Base64.jar:/usr/local/
jade/jade/lib/http.jar:/usr/local/jade/jade/lib/iiop.jar:/usr/local/
jade/jade/lib/jade.jar:/usr/local/jade/jade/lib/jadeTools.jar

export CLASSPATH
```

+ Cập nhật biến môi trườngg, dùng lệnh:

```
# . /etc/profile
```

- Kiểm tra xem quá trình cài đặt thành công không, dùng các lệnh:

```
# xhost +

# java jade.Boot -gui
```

JADE chạy mặc định ở cổng 1099. Nếu hiện ra giao diện của JADE thì cài đặt thành công.

Chú ý: JADE3.x chỉ chạy với Java1.2 .. 1.4; không chạy với Java các phiên bản 1.5 về sau.

1.7. Cấu hình kết nối với cluster-based PBS

1.7.1. Giới thiệu

Thực chất của việc kết nối một nút lưới với một Cluster chính là coi Cluster là một tài nguyên lưới. Tài nguyên này cũng như các tài nguyên khác, cung cấp một giao diện truy cập thống nhất cho các thành viên lưới khác sử dụng.

Để biến một cluster thành một tài nguyên lưới ta thực hiện cài Globus Toolkit lên nút chủ của cluster, thêm một mô-đun thực hiện giao tiếp với nút chủ của cluster và thực hiện các cấu

hình cần thiết để cho Globus có thể giao tiếp được với thành phần quản lý tài nguyên địa phương của cluster.

Globus Toolkit có cung cấp cho chúng ta một số dịch vụ để giao tiếp với PBS_Server. Những dịch vụ này nằm trong gói cài đặt scheduler-pbs-3.2-src_bundle.tar.gz

1.7.2. Các bước cài đặt

Bước 1: Cài đặt hệ thống cluster-based PBS

Tham khảo trong tài liệu kỹ thuật "Hướng dẫn cài đặt OpenPBS-2-3-16"

Bước 2: Cài đặt gói scheduler-pbs-3.2-src_bundle.tar.gz

Chú ý: Yêu cầu trước khi cài đặt

+ Cài đặt GT3.2.1 với service Fork MMJFS đã được cài đặt:

http://127.0.0.1:8080/ogsa/services/base/gram/ForkManagedJobFactoryService

http://127.0.0.1:8080/ogsa/services/base/gram/MasterForkManagedJobFactoryService

+ Submit được một công việc tới Fork service sử dụng lệnh managed-job-globusrun.

+ Cài đặt PBS và cấu hình đúng(sử dụng được qsub, qdel, ...).

+ Thiết lập biến môi trường PBS = "path/to/thư mục chứa qsub, qdel, ..."

+ Thiết lập biến môi trường MPICH = "path/to/thư mục gốc của MPICH"

- Cài đặt

+ Thực hiện lệnh:

```
# gpt-build scheduler-pbs-3.2-src_bundle.tar.gz gcc32dbg
```

Chờ khoảng 10 phút để hoàn thành quá trình cài đặt.

+ Cấu hình: chạy scrip sau để cấu hình tự động:

```
# auto-config-globus-pbs.sh
```

1.8. Cài đặt ứng dụng BKGrid2005

Bước 1: Triển khai các dịch vụ lưới

+ Vào thư mục $GLOBUS_LOCATION

```
# cd $GLOBUS_LOCATION
```

+ Triển khai các dịch vụ lưới bằng lệnh

```
# ant deploy -Dgar.name=....
```

Chú ý:

- Sau khi triển khai một dịch vụ lưới, có thể tìm thấy gói .jar (chứa các tệp class phía server và client - nếu có) của nó trong $GLOBUS_LOCATION/lib, các tệp mô tả dịch vụ dv .gwsdl,

wsdl, ...trong $GLOBUS_LOCATION/schema/gt3ide/; 2 tệp .wsdd và .xml tương ứng dịch vụ đó trong $GLOBUS_LOCATION/undeploy.

- Xóa dịch vụ bằng lệnh:

```
# ant undeploy -Dgar.id=tendv
```

Lệnh này sẽ xóa tất cả các tệp trên.

1. Dịch vụ lưới lai tác tử – khởi động môi trường sống cho các tác tử:

+ Dùng lệnh:

```
# ant deploy -Dgar.name=.../AutoAgent.gar
```

+ Cấu hình để dịch vụ AutoAgent được khởi tạo trước tất cả các dịch vụ khác.

Mở tệp $GLOBUS_LOCATION/server-config.wsdd; chuyển khối XML mô tả dịch vụ AutoAgent lên trên tất cả các dịch vụ khác.

Chú ý: Các dịch vụ sẽ được khởi tạo theo đúng thứ tự mô tả trong file này.

2. Các dịch vụ thông tin tài nguyên

- Dùng lệnh:

```
# ant deploy -Dgar.name=.../Service.gar
```

Các tác tử này sẽ đăng ký với một máy trong lưới làm nhiệm vụ thông tin tài nguyên.

Dịch vụ lập lịch:

- Dùng lệnh:

```
# ant deploy -Dgar.name=.../Scheduler6.gar
```

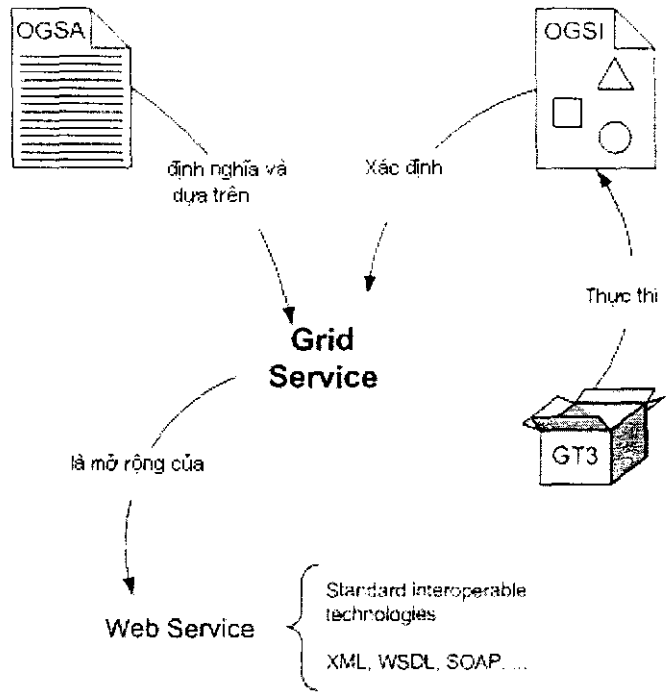
Chú ý: Trước khi bộ lập lịch hoạt động được phải khởi tạo hệ thống tác tử để bộ lập lịch có thể nhận được các thông tin về tài nguyên.

Bước 2: Triển khai portal:

Chương 2: Hướng dẫn lập trình dịch vụ lưới trên GT3

2.1. Các khái niệm cơ sở

2.1.1. OGSA, OGSI và GT3



OGSA

OGSA (Open Grid Service Architecture) - Kiến trúc dịch vụ lưới mở được phát triển bởi Global Grid Forum [http://www.ggf.org], mục đích để đưa ra một chuẩn chung và một kiến trúc mở cho các ứng dụng dựa trên lưới, đồng thời chuẩn hóa các dịch vụ trong một ứng dụng lưới (dịch vụ quản lý công việc, dịch vụ quản lý tài nguyên, dịch vụ bảo mật, ...).

Khi đưa ra một kiến trúc phân tán mới, ta cần lựa chọn một số phần mềm trung gian phân tán (distributed middleware) làm nền tảng trên kiến trúc. Hay nói cách khác, để triệu gọi các phương thức từ xa trong môi trường phân tán, cần có một chuẩn đã công nghiệp hóa, phổ biến rộng rãi hỗ trợ cho kiến trúc này. Ví dụ nếu OGSA định nghĩa giao diện đệ trình công việc JobSubmisionInterface có phương thức submitJob thì sẽ phải có một cách thức tiêu chuẩn để thực thi phương thức đó nếu muốn kiến trúc trở thành một chuẩn công nghiệp rộng rãi. Trên lý thuyết, có thể lựa chọn bất cứ phần mềm trung gian phân tán nào, chẳng hạn RMI, CORBA hay RPC. Tuy nhiên, công nghệ dịch vụ web (Web Service) đã được lựa chọn. Lý do cho sự lựa chọn này sẽ được trình bày ngay sau đây.

Mặc dù kiến trúc dịch vụ web là sự lựa chọn tốt nhất, nó vẫn có một vài điểm không tương thích với các yêu cầu của OGSA. OGSA đã khắc phục những cản trở này bằng cách đưa ra một dạng mở rộng dịch vụ web gọi là dịch vụ lưới (*Grid Service*). Một dịch vụ lưới đơn giản

chỉ là dịch vụ web với rất nhiều mở rộng để tương thích với các ứng dụng dựa trên lưới (grid-based application). Cuối cùng, dịch vụ lưới trở thành công nghệ phân tán của OGSA, hay nói một cách chính xác, OGSA là nền tảng của dịch vụ lưới.

OGSI

OGSA không đi vào chi tiết khi mô tả các dịch vụ lưới. Nó chỉ phác thảo những gì một dịch vụ lưới có mà dịch vụ web không có. Chính vì thế mà OGSA đưa ra một chuẩn khác được gọi là OGSI (Open Grid Services Infrastructure), cũng được phát triển bởi Global Grid Forum, để đưa ra đặc tả chi tiết về kĩ thuật và khuôn dạng của dịch vụ lưới. Nói cách khác, nó là tầm nhìn kiến trúc cao hơn cho dịch vụ lưới, để phù hợp với những thế hệ tiếp theo của ứng dụng lưới.

Globus Toolkit 3

Globus Toolkit là một bộ công cụ, được phát triển bởi Globus Alliance, để ta có thể phát triển các ứng dụng lưới. Phiên bản thứ 3 của bộ công cụ này (GT3) chứa đặc tả đầy đủ của OGSI (trong lược đồ trên, GT3 cài đặt kiến trúc OGSI). Tuy nhiên, một điều quan trọng là GT3 không chỉ là một thực thi của OGSI. Nó còn chứa rất nhiều các dịch vụ, chương trình và tiện ích khác. Một số xây dựng trên kiến trúc OGSI gọi là *WS components (các thành phần dịch vụ web)*, trong khi một số thành phần khác không xây dựng trên kiến trúc của OGSI được gọi là *pre-WS components (các thành phần trước dịch vụ web)*.

Sự khác nhau giữa OGSA, OGSI và GT3

Nếu bạn vẫn còn băn khoăn khi phân biệt OGSA, OGSI và GT3, hãy xét một ví dụ đơn giản sau. Giả sử rằng bạn muốn xây một ngôi nhà mới. Điều đầu tiên bạn làm là thuê một kiến trúc sư để vẽ phác thảo một bản kiến trúc cho ngôi nhà của bạn. Khi bạn đã hài lòng với bản kiến trúc, bạn sẽ thuê một kĩ sư để lên kế hoạch chi tiết cho việc xây dựng. Sau đó, người kĩ sư sẽ thông qua kế hoạch của anh ta cho các kĩ sư xây dựng lành nghề (công nhân xây dựng, kĩ sư điện, nước,...), những người thực sự xây dựng ngôi nhà của bạn.

Chúng ta có thể nói rằng OGSA (định nghĩa) là bản phác thảo của kiến trúc, OGSI (đặc tả) là thiết kế cấu trúc mà người kĩ sư tạo ra để hỗ trợ cho bản kiến trúc xây dựng. Và cuối cùng, GT3 sẽ là những viên gạch, ximăng,... để xây dựng theo đặc tả của người kĩ sư.

2.1.2. Dịch vụ web (Web Services)

Bởi vì dịch vụ web là cơ sở của dịch vụ lưới, việc hiểu kiến trúc dịch vụ web là thiết yếu để sử dụng GT3 và lập trình dịch vụ lưới.

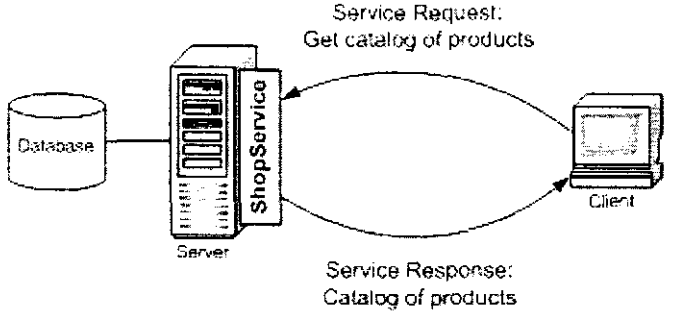
Gần đây, người ta nói rất nhiều về "dịch vụ web", và có nhiều công ty bắt đầu phát triển các ứng dụng doanh nghiệp dựa trên công nghệ này.. Vậy, dịch vụ web chính xác là gì? Đơn giản, nó chỉ là một công nghệ tính toán phân tán(tương tự CORBA, RMI, EJB, ...), cho phép ta phát triển các ứng dụng client/server.

Ví dụ, giả sử bạn phải phát triển một ứng dụng cho một dãy các kho chứa. Các kho lưu trữ này nằm ở khắp mọi nơi trên đất nước, nhưng bản danh mục (catolog) chính của các sản phẩm chỉ có sẵn tại cơ sở dữ liệu tại trung tâm cơ quan của bạn, và phần mềm tại các kho

chứa phải truy nhập vào bản danh mục này. Bạn có thể xuất bản một bản danh mục thông qua một dịch vụ web gọi là *ShopService (dịch vụ bán hàng)*.

Chú ý: Không được nhầm lẫn giữa dịch vụ web và website. Các thông tin của một website được dành cho con người, còn các thông tin có sẵn qua một dịch vụ web luôn luôn được truy nhập bởi một phần mềm, không phải trực tiếp bởi một người. Mặc dù dịch vụ web dựa nhiều trên công nghệ Web hiện tại (HTTP, ...), nhưng chúng không có có mối liên hệ nào giữa trình duyệt và HTML. Bạn hãy ghi nhớ, website cho con người, dịch vụ web cho phần mềm!

Các máy khách (các PCs tại các kho chứa) có thể truy nhập dịch vụ web (nằm trên server), và gửi các yêu cầu dịch vụ (service request) để xem nội dung bảng danh mục sản phẩm. Server có thể trả về nội dung danh mục thông qua các đáp ứng dịch vụ (*service response*). Tất nhiên, đây chỉ là một ví dụ rất đơn giản để mô tả hoạt động của dịch vụ web. Hình vẽ dưới đây mô tả hoạt động trên:



Với ví dụ trên, hẳn bạn sẽ nghĩ rằng *RMI, CORBA, EJBs,* và các công nghệ tính toán phân tán khác cũng có thể thực hiện được công việc trên, vậy dịch vụ web có gì là đặc biệt đâu. Tuy nhiên, hãy xem xét một số ưu điểm của dịch vụ web so với các công nghệ khác:

- Dịch vụ web là độc lập về nền hệ điều hành và ngôn ngữ, bởi vì ta sử dụng ngôn ngữ XML chuẩn. Điều này có nghĩa là chương trình khách có thể được lập trình bằng C++ và chạy trên nền hệ điều hành Windows, trong khi dịch vụ web được lập trình bằng Java và chạy trên nền hệ điều hành Linux.
- Hầu hết các dịch vụ web sử dụng HTTP cho truyền thông điệp, điều này là một thuận lợi lớn khi ta phát triển các ứng dụng trên Internet, bởi vì các tường lửa và proxy trên Internet không làm đảo lộn các truyền thông của HTTP (không giống như CORBA, gặp phiền toái với vấn đề tường lửa)

Tuy nhiên, dịch vụ web có một số bất lợi sau:

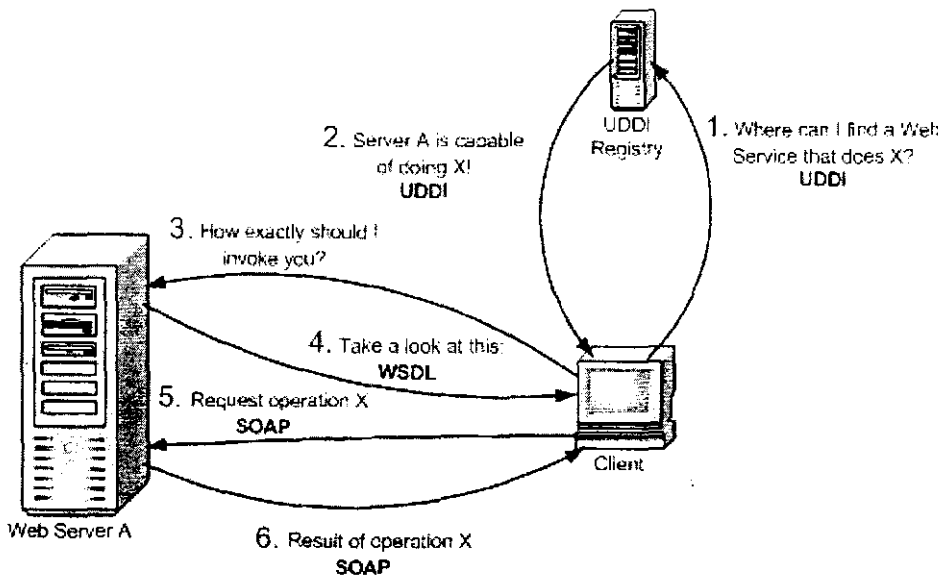
- Trước hết, việc truyền dữ liệu qua XML rõ ràng là không hiệu quả, bởi vì đã sử dụng mã nhị phân. Để có được tính linh động, bạn đã đánh đổi bằng tính hiệu quả. Mặc dù vậy, bất lợi này vẫn được chấp nhận trong hầu hết các ứng dụng, nhưng bạn sẽ không bao giờ thấy một ứng dụng thời gian thực sử dụng dịch vụ web.
- Thiếu tính linh động: Hiện tại dịch vụ web là không linh động, khi chúng chỉ cho phép một số dạng triệu gọi dịch vụ cơ bản. CORBA, có thể cho phép lập trình

viên nhiều cách cung cấp dịch vụ hơn như là dịch vụ vĩnh viễn (persistency), thông báo (notifications), quản lý vòng đời (lifecycle management), ... Phần tiếp theo, ta sẽ được biết dịch vụ lưới bổ sung những thiết sót này.

Tuy nhiên, có một đặc điểm quan trọng để phân biệt dịch vụ web với các công nghệ tính toán phân tán khác. Trong khi các công nghệ như CORBA, EJBs hướng vào các hệ thống tính toán phân tán phụ thuộc (*highly-coupled* distributed systems), khi mà client và server phải phụ thuộc vào nhau, dịch vụ web lại thích hợp cho các hệ thống phân tán không phụ thuộc (*losely- coupled* systems), khi mà client có thể không biết gì về dịch vụ web cho tới khi nó thực sự triệu gọi các dịch vụ Web. Các hệ thống phân tán phụ thuộc là ý tưởng cho các ứng dụng Intranet, nhưng lại có hiệu suất thấp trên môi trường Internet. Dịch vụ web, là thích hợp hơn đối với các ứng dụng trên Internet, ví dụ như là các ứng dụng lưới.

Một triệu gọi dịch vụ web điển hình

Để hiểu rõ hơn về hoạt động của dịch vụ web, hãy theo dõi các bước để triệu gọi một dịch vụ web. Các khái niệm về SOAP, WSDL sẽ được giải thích cụ thể ở ngay phần sau.



- Như đã đề cập trước đó, một client có thể không biết gì về dịch vụ web mà nó định triệu gọi. Bởi vậy, bước đầu tiên là tìm một dịch vụ web đáp ứng các đòi hỏi của nó. Ví dụ, nếu quan tâm tới một dịch vụ web cung cấp thông tin về thời tiết của các thành phố của Mỹ, ta có thể tìm nó qua bộ đăng ký và khai phá dịch vụ UDDI.
- Bộ đăng ký UDDI sẽ trả lời cho chúng ta biết server nào cung cấp cho ta dịch vụ mà ta yêu cầu (chẳng hạn thời tiết các thành phố của Việt Nam).
- Bây giờ, khi ta biết về vị trí của dịch vụ web, nhưng ta vẫn chưa biết triệu gọi dịch vụ đó như thế nào. Chúng ta phải yêu cầu dịch vụ web mô tả thông tin về chính nó, để ta biết chính xác phương thức nào ta cần triệu gọi
- Dịch vụ web sẽ trả lời bằng một ngôn ngữ gọi là WSDL.

- Cuối cùng, chúng ta đã biết dịch vụ web nằm ở đâu và triệu gọi nó như thế nào. Quá trình triệu gọi được thực hiện bởi một ngôn ngữ gọi là SOAP. Do đó, trước tiên ta phải gửi một yêu cầu SOAP (*SOAP request*) để lấy thông tin về nhiệt độ.
- Dịch vụ Web sẽ trả lời với một đáp ứng SOAP (*SOAP response)* chứa thông tin về nhiệt độ mà ta yêu cầu. Nó có thể là một thông điệp lỗi nếu yêu cầu SOAP của chúng ta là lỗi.

Địa chỉ của dịch vụ web

Ta đã triệu gọi một dịch vụ web đơn giản, và bộ đăng kí UDDI đã cho client biết nơi mà dịch vụ web được định vị. Nhưng địa chỉ dịch vụ web chính xác là như thế nào? Câu trả lời rất đơn giản: như là một trang Web. Chúng được đánh địa chỉ qua các định danh tài nguyên thống nhất URIs (Uniform Resource Identifiers). Nếu bạn quen thuộc hơn với thuật ngữ URL (Uniform Resource Locator), đừng lo lắng vì URI và URL trên thực tế là giống nhau.

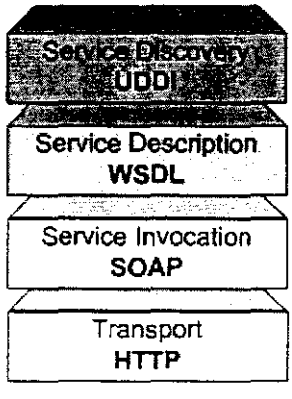
Ví dụ, bộ đăng ký UDDI có thể trả lời với địa chỉ URI sau:

http://webservices.mysite.com/weather/us/WeatherService

Điều này cũng dễ hiểu như địa chỉ của một trang web. Tuy nhiên, nhớ rằng dịch vụ web luôn được sử dụng bởi các phần mềm (không bao giờ trực tiếp bởi con người). Nếu bạn gõ một địa chỉ dịch vụ web URI vào trong trình duyệt của bạn, bạn có thể nhận một thông báo lỗi hay một số mã lỗi (một số máy chủ web sẽ hiện một giao diện đồ họa tới dịch vụ web, nhưng điều này không phổ biến). Khi bạn có một dịch vụ web URI, bạn cần đưa URI đó tới một chương trình. Thực tế, hầu hết các chương trình ta viết sẽ nhận về Grid Service URI như một tham số dòng lệnh.

Kiến trúc dịch vụ web

Kiến trúc phân lớp của dịch vụ web được biểu diễn như hình vẽ:



- **Tìm kiếm dịch vụ (Service Discovery):** Thành phần này cho phép ta tìm kiếm dịch vụ web đáp ứng yêu cầu. Nó được thực hiện bởi UDDI - bộ tích hợp, tìm kiếm và mô tả phổ biến (Universal Description, Discovery, and Integration). GT3 hiện tại không hỗ trợ cho UDDI.
- **Mô tả dịch vụ (Service Description)** : Một trong những tính năng thú vị nhất của dịch vụ web là chúng có thể mô tả chính mình. Điều này có nghĩa là ngay khi bạn xác

định được vị trí của dịch vụ web, bạn có thể yêu cầu dịch vụ mô tả những toán tử mà nó cung cấp và triệu gọi nó như thế nào. Điều này được thực hiện bởi ngôn ngữ mô tả dịch vụ Web - Web Service Description Language (WSDL).

- **Triệu gọi dịch vụ (Service Invocation)**: Triệu gọi một dịch vụ web (hay bất kì một đối tượng phân tán nào như CORBA, Enterprise Java Bean) liên quan tới truyền thông điệp giữa client và server. Giao thức truy nhập đối tượng đơn giản - SOAP (Simple Object Access Protocol) sẽ xác định xem chúng ta định dạng yêu cầu tới server như thế nào. Trên lý thuyết, ta có thể sử dụng các ngôn ngữ triệu gọi dịch vụ khác (như XML-RPC, hay thậm chí một số ngôn ngữ *ad hoc* XML). Tuy nhiên, SOAP là sự lựa chọn phổ biến nhất cho dịch vụ web.
- **Truyền tải (Transport)**: Cuối cùng, tất cả những thông điệp này phải được truyền đi giữa client và server. Giao thức được lựa chọn cho phần này của kiến trúc là HTTP (HyperText Transfer Protocol), cũng là giao thức để truy nhập các trang web trên Internet. Tuy nhiên, trên lý thuyết, ta có thể sử dụng các giao thức khác, nhưng HTTP hiện tại là phổ biến nhất.

Một ứng dụng dịch vụ web sẽ như thế nào

Bạn đã biết dịch vụ web là gì, ắt hẳn là bạn sẽ muốn bắt đầu lập trình dịch vụ web. Trước khi bạn làm điều đó, bạn cần biết cấu trúc của các ứng dụng dựa trên dịch vụ web như thế nào. Nếu bạn đã từng lập trình CORBA hay RMI, cấu trúc này cũng tương tự như thế.

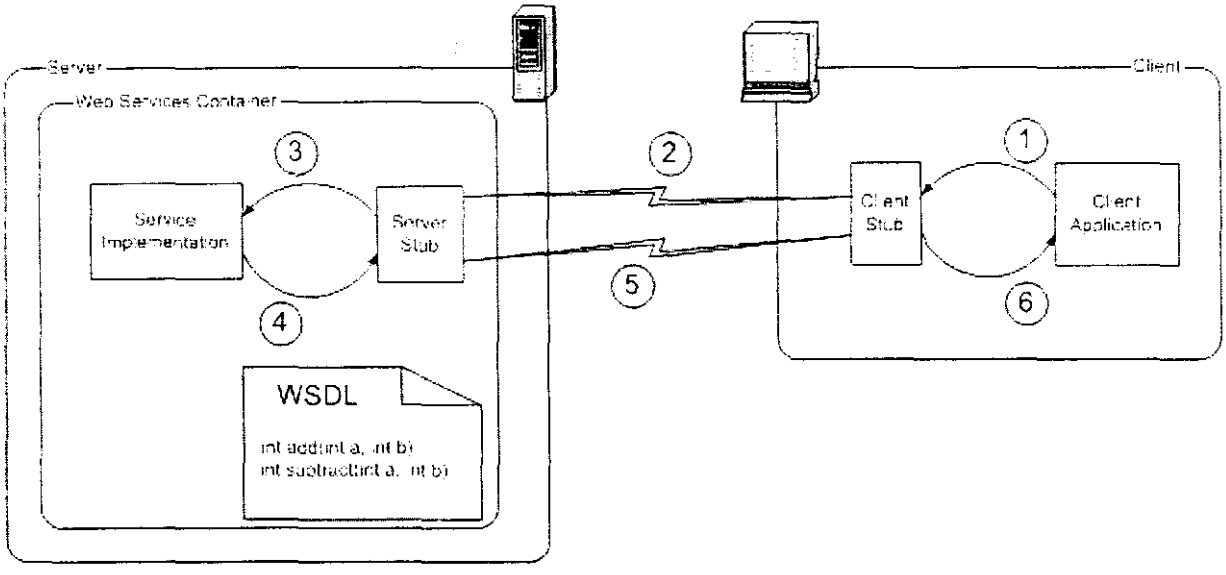
Điều trước tiên, mặc dù có rất nhiều các giao thức và ngôn ngữ trong kiến trúc của dịch vụ web, nhưng các lập trình viên dịch vụ web thường không bao giờ viết một dòng SOAP hay WSDL nào. Ngay khi ta tới được điểm mà các ứng dụng client cần triệu gọi một dịch vụ web, chúng ta sẽ uỷ nhiệm công việc này cho một phần của phần mềm gọi là *client stub*. Một thông tin tuyệt vời nữa là có rất nhiều công cụ sinh client stubs tự động cho chúng ta, dựa trên mô tả WSDL của dịch vụ web.

Do đó, trong lược đồ triệu gọi phần trước, client thường không phải thực hiện tất cả các bước trong một triệu gọi đơn. Một thứ tự các sự kiện chính xác hơn có thể là:

- Chúng ta xác định một dịch vụ web đáp ứng các yêu cầu của ta thông qua UDDI.
- Thiết lập mô tả WSDL của dịch vụ web.
- Ta phát sinh stubs ngay sau đó, cho chúng vào ứng dụng của chúng ta.
- Ứng dụng sẽ sử dụng stubs tại mỗi thời điểm nó triệu gọi dịch vụ web.

Mô hình lập trình phía server rất đơn giản, ta không phải viết một chương trình server phức tạp, tự động thông dịch các yêu cầu SOAP và đưa ra các đáp ứng SOAP. Chúng ta đơn giản chỉ cài đặt tất cả các chức năng cho dịch vụ web của chúng ta, sau đó sinh ra một *server stub* (hay còn được gọi là *skeleton*). Server stub sẽ chịu trách nhiệm thông dịch các yêu cầu, đưa chúng tới cho dịch vụ thực thi. Nó cũng có thể tự động sinh ra mô tả WSDL, hay các ngôn ngữ mô tả khác (IDL trên CORBA). Hơn nữa, cả thực thi phía server và server stubs đều được quản lý bởi một thành phần gọi là trình chứa dịch vụ web (*Web Service container)*, bảo đảm các yêu cầu HTTP cho dịch vụ web được trực tiếp tới server stub.

Các bước liên quan tới triệu gọi một dịch vụ web được mô tả bằng lược đồ sau:



Giả sử rằng chúng ta đã định vị được dịch vụ web, và phát sinh client stubs từ mô tả WSDL. Ngoài ra, các lập trình viên phía server cũng phải phát sinh server stubs.

- Bất cứ khi nào ứng dụng client cần triệu gọi dịch vụ web, nó sẽ gọi client stub. Client stub sẽ chuyển triệu gọi địa phương ('local invocation') vào trong một yêu cầu SOAP hợp lý. Điều này thường được gọi là tiến trình *marshaling* hay *serializing*.
- Yêu cầu SOAP được gửi qua mạng thông qua giao thức HTTP. Trình chứa dịch vụ nhận được các yêu cầu SOAP, và chuyển nó tới server stub. Server stub sẽ biến đổi các yêu cầu SOAP thành các cái gì đó mà thực thi phía server có thể hiểu được. (điều này thường được gọi là *unmarshaling* hay *deserializing*). Bạn có thể tham khảo thêm cuốn "Java lập trình mạng" để hiểu thêm về các thuật ngữ này.
- Thực thi dịch vụ sẽ nhận các yêu cầu từ service stub, và tiến hành công việc mà nó được yêu cầu. Ví dụ, bạn triệu gọi phương thức int add(int a, int b), thực thi dịch vụ sẽ thực hiện một phép cộng.
- Kết quả của các toán tử được yêu cầu được chuyển tới server stub, và stub sẽ chuyển nó thành các đáp ứng SOAP.
- Đáp ứng SOAP được gửi qua mạng thông qua giao thức HTTP. Client stub nhận được các đáp ứng SOAP và chuyển nó thành cái gì đó mà các ứng dụng client có thể hiểu được.
- Cuối cùng, ứng dụng nhận được kết quả của triệu gọi dịch vụ web và sử dụng nó.

Cuối cùng, bạn cũng không phải băn khoăn gì, kiến trúc dịch vụ web được chuẩn hoá và đặc tả bởi tổ chức World Wide Web Consortium, tổ chức đã đưa ra các chuẩn XML, HTML, CSS, ..

2.1.3. Dịch vụ lưới (Grid Services)

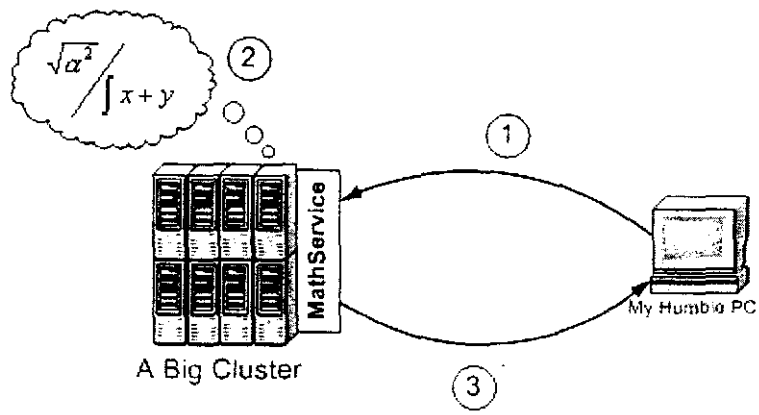
Như đã đề cập, dịch vụ web là lựa chọn công nghệ cho các ứng dụng dựa trên Internet, với các client và server là độc lập. Điều này khiến cho chúng được lựa chọn cho việc xây dựng thế hệ tiếp theo của các ứng dụng lưới. Tuy nhiên, nhớ rằng dịch vụ web còn một số hạn chế. Thực tế một dịch vụ web đơn giản (hiện được định nghĩa bởi W3C) là rất không hữu ích cho xây dựng ứng dụng lưới. **Dịch vụ lưới**, dựa trên nền tảng dịch vụ web với một số đặc tính và dịch vụ được cải tiến, là sự lựa chọn cho việc xây dựng.

Chúng ta hãy cũng xem qua những cải tiến được giới thiệu trong OGSI:

- Các dịch vụ có trạng thái và ngắn hạn tiềm năng (Stateful and potentially transient services)
- Dữ liệu dịch vụ (Service Data)
- Thông báo (Notifications)
- Các nhóm dịch vụ (Service Groups)
- Mở rộng portType (portType extension)
- Quản lý vòng đời (Lifecycle management)
- GSH & GSR

Các dịch vụ có trạng thái và ngắn hạn tiềm năng (Stateful and potentially transient services)

Tính năng đầu tiên có thể là cải tiến quan trọng nhất đối với dịch vụ web. Hãy thử xem tính năng này là gì thông qua ví dụ đơn giản sau đây. Tưởng tượng rằng tổ chức của bạn có một cluster rất lớn, có khả năng thực hiện hầu hết các tính toán khổng lồ. Tuy nhiên, cluster này được đặt ở trụ sở chính tại Chicago, và bạn lại cần sử dụng một cách thuận tiện năng lực tính toán của cluster từ cơ quan của bạn tại New York, Los Angeles, và Seattle. Điều này dường như là một kịch bản hoàn hảo cho một dịch vụ web!



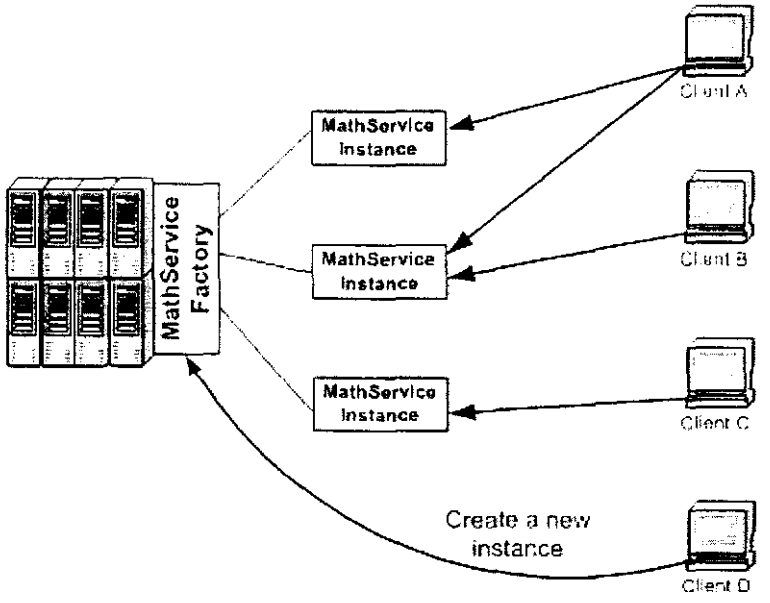
Chúng ta có thể cài đặt một dịch vụ web tên là *MathService*, đưa ra các toán tử như là SolveReallyBigSystem(), SolveFermatsLastTheorem(), ... Trước tiên, ta có thể thực hiện các triệu gọi dịch vụ web điển hình:

- Triệu gọi MathService, yêu cầu nó thực hiện một toán tử xác định.
- MathService sẽ chỉ thị cho cluster thực hiện toán tử đó.
- MathService sẽ trả về kết quả của toán tử.

Tuy nhiên, hãy thử thực tế hơn. Nếu bạn dự định truy nhập một cluster từ xa để thực hiện các tính toán phức tạp, bạn có thể không thực hiện một đơn toán tử, mà là một chuỗi các toán tử, mà tất cả chúng đều liên quan đến nhau. Tuy nhiên, dịch vụ web là phi trạng thái (*stateless).* Nghĩa là dịch vụ web không thể nhớ được những gì bạn đã làm từ một triệu gọi tới một triệu gọi khác. Nếu bạn muốn thực hiện một chuỗi các toán tử, bạn có thể phải lấy kết quả của một toán tử và gửi nó đến như là tham số cho toán tử tiếp theo.

Hơn nữa, thậm chí ta đã giải quyết được vấn đề phi trạng thái (một số trình chứa dịch vụ webs đã giải quyết vấn đề này), dịch vụ webs luôn luôn tồn tại lâu dài (*non-transient)*, nghĩa là dịch vụ web tồn tại với tất cả các client của chúng. Dịch vụ web còn được gọi kiên định - *persistent* (đối lập với *transient*; nhưng không có nghĩa là 'persistent' trong ngữ cảnh của các kho lưu trữ dữ liệu thứ cấp, ổ cứng,...) bởi vì nó tồn tại song song với trình chứa dịch vụ web (một dịch vụ web luôn luôn có sẵn từ khi server được khởi động, nó sẽ không bị huỷ cho tới khi ta tắt server). Trong bất cứ trường hợp nào, một client đang sử dụng một dịch vụ web, tất cả các thông tin của dịch vụ web đang ghi nhớ có thể được truy nhập bởi client thứ hai. Thực tế, một client đang sử dụng dịch vụ web, một client khác truy nhập vào dịch vụ web sẽ có thể có những lộn xộn tiềm tàng đối với toán tử của client đầu tiên. Điều này quả là một vấn đề!

Dịch vụ lưới giải quyết cả hai vấn đề trên bằng cách cho người lập trình sử dụng cách tiếp cận xưởng chế tác/thực thể (*factory/instance)* đối với dịch vụ web. Thay vì có một dịch vụ MathService không trạng thái lớn, chia sẻ cho tất cả người dùng, chúng ta có một xưởng chế tác MathService trung tâm, chịu trách nhiệm duy trì các thực thể MathService. Khi client muốn triệu gọi toán tử MathService, nó sẽ nói chuyện với thực thể, không phải xưởng chế tác. Khi client cần một thực thể mới được tạo hay huỷ bỏ, nó sẽ nói chuyện với xưởng chế tác. Có thể nói ngắn gọn xưởng chế tác như một ông quản đốc :)



Lược đồ trên cho thấy tại sao không cần thiết có mỗi thực thể cho mỗi client. Mỗi thực thể có thể được chia sẻ bởi hai client, và một client có thể truy nhập tới hai dịch vụ. Những thực thể này là ngắn hạn (*transient)*, bởi vì chúng có thời gian sống hạn chế, không phụ thuộc vào vòng đời của trình chứa dịch vụ lưới. Nói cách khác, chúng ta có thể tạo và huỷ các thực thể dịch vụ bất cứ khi nào ta muốn chúng (thay vì có một dịch vụ có sẵn, tồn tại lâu dài). Vòng đời thực tế của một thực thể có thể thay đổi từ ứng dụng tới ứng dụng khác. Thông thường, chúng ta muốn thực thể sống, miễn là có một client sử dụng một tính năng nào của chúng. Tuy nhiên, có các kịch bản khác nhau, khi mà ta muốn một thực thể được chia sẻ bởi một vài người dùng, và có thể tự huỷ khi không có client nào sử dụng nó trong một khoảng thời gian nhất định.

Cuối cùng, chú ý thế nào là một dịch vụ lưới ngắn hạn tiềm tàng (*potentially* transient). Điều này nghĩa là không phải tất cả dịch vụ lưới phải sử dụng tiếp cận xưởng chế tác/thực thể. Một dịch vụ lưới có thể dài hạn (persistent), như là một dịch vụ web thông thường. Lựa chọn giữa dịch vụ lưới dài hạn hay xưởng chế tác/thực thể phụ thuộc hoàn toàn vào các yêu cầu của ứng dụng của bạn.

Quản lý vòng đời (Lifecycle management)

Bởi vì chúng ta đang giải quyết vấn đề vòng đời của dịch vụ (nếu ta sử dụng tiếp cận xưởng chế tác/thực thể, các thực thể có thể được khởi tạo và hủy bỏ tại bất cứ thời điểm nào), các cơ chế quản lý vòng đời cũng được cung cấp trong dịch vụ lưới. OGSI hỗ trợ một số cơ chế rất đơn giản, mà sẽ được cài đặt trong các ví dụ ta thấy ở phần sau.

Dữ liệu dịch vụ (Service Data)

Dữ liệu dịch vụ, cùng với có trạng thái và ngắn hạn, được đánh giá rất cao trong những mở rộng của dịch vụ lưới đối với dịch vụ web. Dữ liệu dịch vụ cho phép ta thêm vào một tập dữ liệu có cấu trúc tới bất kì dịch vụ nào, mà có thể truy nhập trực tiếp thông qua giao diện của nó. Bởi vì các dịch vụ web đơn giản chỉ cho phép hoạt động cùng với giao diện WSDL, bạn có thể thấy dữ liệu dịch vụ không chỉ hoạt động trong WSDL, mà còn trong các thuộc tính.

Tuy nhiên, dữ liệu dịch vụ là không chỉ là những thuộc tính đơn giản, ta có thể thêm vào bất cứ kiểu dữ liệu nào một cách dễ dàng (lớp, mảng, ...)

Nói chung, dữ liệu dịch vụ mà ta thêm vào trong dịch vụ được chia thành hai loại sau đây:

- **Thông tin trạng thái (State information)**: cung cấp thông tin về trạng thái hiện tại của dịch vụ, như là kết quả của toán tử, thông tin thời gian thực, ...
- **Thông tin dịch vụ (Service metadata)**: Thông tin của chính dịch vụ, như là dữ liệu hệ thống, các giao diện được hỗ trợ, giá của các dịch vụ,....

Phần dữ liệu dịch vụ sẽ được giải thích chi tiết khi ta bắt đầu lập trình dịch vụ lưới với dữ liệu dịch vụ.

Thông báo (Notifications)

Một dịch vụ lưới có thể được cấu hình là nhà cung cấp thông báo (*notification source)*, và tất nhiên client là người yêu cầu các thông báo (*notification sinks).* Điều này nghĩa là mọi thay đổi trong dịch vụ lưới, được thông báo tới tất cả các client (tất nhiên là không phải tất cả các thay đổi đều được thông báo, chỉ những gì mà người lập trình dịch vụ lưới muốn). Trong ví dụ MathService, giả sử tất cả các khách hàng thực hiện các tính toán, sử dụng một biến, gọi là *InterestingCoefficient*, được lưu trữ trong dịch vụ lưới. Bất cứ khách hàng nào muốn cũng có thể thay đổi giá trị đó để thực hiện các tính toán. Tuy nhiên, các khách hàng phải được thông báo về thay đổi đó khi nó xảy ra. Chúng ta có thể dễ dàng thực hiện yêu cầu này với cơ chế thông báo của dịch vụ lưới. Các ví dụ lập trình ở phần sau sẽ thể hiện rõ ràng hơn điều này.

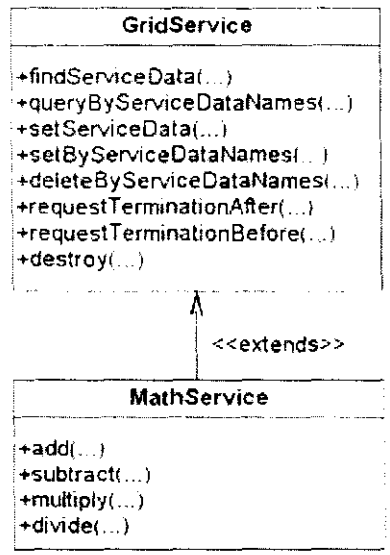
Nhóm dịch vụ (Service Groups)

Bất cứ dịch vụ nào cũng có thể được cấu hình như một nhóm dịch vụ (*service group)* để kết tập các dịch vụ khác. Bạn có thể dễ dàng thực hiện các toán tử như là 'thêm dịch vụ mới vào nhóm', 'bỏ một dịch vụ trong nhóm', và quan trọng hơn là 'tìm một dịch vụ trong nhóm'. Mặc dù nhóm dịch vụ trong OGSI là khá đơn giản, nó là nền tảng cho các dịch vụ thư mục (như là dịch vụ chỉ mục của GT3), cho phép chúng ta nhóm các dịch vụ khác nhau lại và truy cập chúng thông qua một *lối vào của nhóm dịch vụ.*

Thừa kế portType (portType extension)

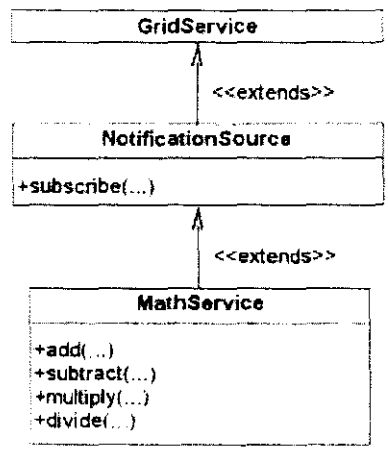
Trong phần trước ta đã thấy rằng, một dịch vụ web thể hiện giao diện của nó thông qua tài liệu đặc tả WSDL. Giao diện thường được gọi là *portType* (bởi vì có một thẻ WSDL có cùng tên). Một dịch vụ web thông thường có thể chỉ có một portType. Dịch vụ lưới, hỗ trợ thừa kế portType (*portType extension)*, nghĩa là chúng ta có thể định nghĩa một portType như là một mở rộng của một portType trước đó đang tồn tại.

Ví dụ, đặc tả OGSI uỷ nhiệm rằng tất cả các dịch vụ dịch vụ lưới phải được thừa kế từ một portType chuẩn tên là GridService:

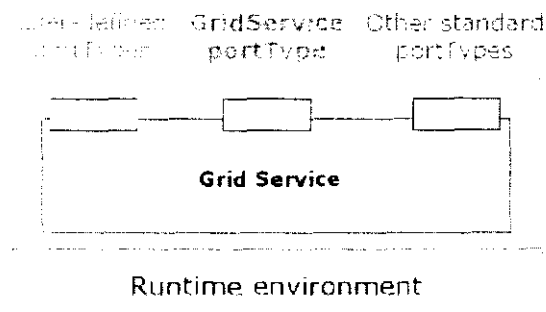


Nhờ có thừa kế portType, ta có thể định nghĩa đơn giản portType của chính ta như là một thừa kế của GridService. Với một dịch vụ web, ta cần thêm khai báo của tất cả các toán tử (bao gồm toán tử GridService) như là một portType đơn.

Bên cạnh portType GridService chuẩn, OGSI định nghĩa một một số portType chuẩn khác, chúng ta có thể mở rộng để thêm các chức năng vào trong dịch vụ lưới của ta. Ví dụ có một NotificationSource portType, ta có thể mở rộng nếu ta muốn dịch vụ của ta là một nhà cung cấp thông tin. Chú ý là NotificationSource thừa kế từ GridService:



Nói chung, ta có thể thấy 3 loại portType sau của dịch vụ lưới:



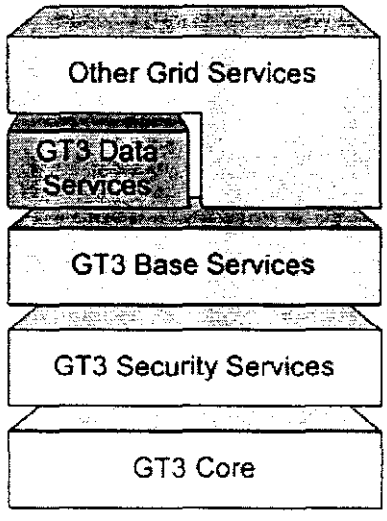
GSH & GSR

Trong phần trước, ta biết là dịch vụ web được đánh địa chỉ với URIs. Bởi vì dịch vụ lưới là dịch vụ web, chúng cũng có thể đánh địa chỉ với URIs. Tuy nhiên, OGSI đưa ra một lược đồ đánh địa chỉ hiệu quả hơn.

Mỗi "Grid Service URI" còn gọi là *Grid Service Handle*, hay đơn giản là GSH. Mỗi GSH phải là duy nhất. Không thể có hai dịch vụ lưới có cùng GSH. Chỉ có một vấn đề với GSH là nó cho chúng ta biết dịch vụ lưới ở đâu, nhưng không cho ta bất cứ thông tin nào về cách giao tiếp với dịch vụ lưới (nó có phương thức gì, loại thông điệp nào,...). Để thực hiện điều này, ta cần tới tham chiếu dịch vụ lưới *Grid Service Reference*, hay GSR. Trên lý thuyết, GSR có thể có nhiều dạng khác nhau, nhưng bởi vì chúng ta thường sử dụng SOAP để giao tiếp với dịch vụ lưới, GSR sẽ là một file WSDL file (WSDL mô tả một dịch vụ web: những phương thức nó có, ...). Trong tài liệu này, ta thực hiện WSDL như là một định dạng GSR.

2.1.4. GT3

Nếu bạn đã lập trình các ứng dụng lưới, bạn có thể biết là dịch vụ lưới là rất tốt đẹp, nhưng vẫn chưa đủ cho lưới. Dịch vụ lưới chỉ là một phần nhỏ (nhưng quan trọng!) trong toàn bộ kiến trúc của GT3, đòi hỏi rất nhiều nhà lập trình dịch vụ làm việc với lập trình lưới. Dưới đây là kiến trúc phân lớp của GT3:



OGSI (ví dụ "dịch vụ lưới") là lớp 'GT3 Core' . Ta thử xem các lớp theo hướng từ dưới lên:

Các dịch vụ bảo mật (GT3 Security Services): Bảo mật là thành phần quan trọng nhất của các ứng dụng dựa trên lưới. Các dịch vụ bảo mật của GT3 cho phép chúng ta hạn chế truy nhập các dịch vụ lưới, cho nên chỉ các client đã chứng thực mới được sử dụng chúng. Bên cạnh các phương pháp bảo mật truyền thống (đặt máy chủ web đằng sau tường lửa,...), GT3 sử dụng hơn một lớp bảo mật với các công nghệ như SSL và giấy chứng nhận X.509.

Các dịch vụ cơ bản (GT3 Base Services): Lớp này bao gồm toàn bộ các dịch vụ quan tâm:

- Dịch vụ quản lý công việc (Managed Job Service): Dịch vụ này được sử dụng để kiểm tra tiến trình thực hiện theo định kì của các công việc, mà ta đệ trình lên lưới để yêu cầu thực hiện.
- Dịch vụ chỉ mục (Index Service): Ở phần giới thiệu về dịch vụ web, ta đã biết về các loại dịch vụ web mà ta cần, nhưng không biết được là chúng ở đâu. Điều này cũng xảy ra với dịch vụ lưới, ta có thể biết được là dịch vụ lưới nào đáp ứng được các yêu cầu của ta, nhưng lại không biết nó ở đâu. Trong khi điều này đã được giải quyết trong dịch vụ web với UDDI, GT3 cũng có một Index Service của chính nó. Ví dụ, chúng ta có hàng tá các MathServices ở quanh đất nước, với các đặc tính khác nhau (một số phù hợp cho các phân tích thống kê, một số lại tốt hơn cho hoạt động mô phỏng). Index Service cho phép chúng ta truy vấn dịch vụ MathService nào là đáp ứng được với yêu cầu riêng biệt của ta.
- Dịch vụ truyền file tin cậy - Reliable File Transfer (RFT) Service: Dịch vụ này cho phép ta thực hiện truyền một lượng lớn file giữa client và dịch vụ lưới. Ví dụ, chúng ta có một toán tử trong MathService, phải thao tác hàng GB dữ liệu thô (cho các phân tích thống kê), tất nhiên, chúng ta không thể gửi tất cả các thông tin như một tham số đầu vào. Chúng ta có thể gửi chúng như một file. Dịch vụ truyền file tin cậy RFT bảo đảm việc di chuyển là tin cậy. Ví dụ, nếu có một file truyền bị gián đoạn (do mạng máy tính bị lỗi), RFT cho phép ta khởi động lại quá trình truyền file tại thời điểm nó bị gián đoán, thay vì khởi động tất cả lại từ đầu.

Dịch vụ dữ liệu (GT3 Data Services): Lớp này bao gồm quản lý bản sao, là rất hữu ích cho các ứng dụng phải giải quyết với tập lớn các dữ liệu. Khi làm việc với lượng lớn dữ liệu, ta không phải quan tâm tới việc phải tải toàn bộ mọi thứ, ta chỉ phải làm việc với một phần nhỏ của khối dữ liệu. Quản lý bản sao (Replica Management) lưu giữ các tập con dữ liệu mà ta cần làm việc.

Các dịch vụ lưới khác (Other Grid Services): Một số dịch vụ non-GT3 Grid Service có thể chạy ở phía trên của kiến trúc GT3.

WSRF & GT4

Đây là những bước phát triển tiếp theo của công nghệ lưới. Nếu bạn quan tâm, có thể tham khảo thêm trên mạng Internet qua địa chỉ: http://www.globus.org

2.2. Cách viết một dịch vụ lưới trên GT3

2.2.1. Định nghĩa giao diện dịch vụ

Đây là bước đầu tiên trong quá trình viết một dịch vụ lưới. Chúng ta cần phải xác định dịch vụ của chúng ta sẽ cung cấp những gì. Tại thời điểm này, ta chưa phải quan tâm tới nội dung bên trong của dịch vụ (sử dụng giải thuật gì, truy nhập cơ sở dữ liệu nào), mà chỉ cần biết các "thao tác" nào được cung cấp cho người dùng. Nhớ rằng ở phần trước, khi giới thiệu về dịch vụ web, giao diện của dịch vụ còn gọi là port Type (còn được viết là portType).

Chúng ta sử dụng một ngôn ngữ XML đặc biệt để viết các mô tả giao diện này: WSDL (dịch vụ Web Service Description Language).

Với GT3 chúng ta có hai sự lựa chọn khi viết định nghĩa giao diện dịch vụ

- Viết trực tiếp WSDL: đây là cách linh hoạt nhất nhưng cũng tốn nhiều công sức nhất. Với cách viết này chúng ta có toàn quyền định nghĩa portType.

- Sinh WSDL từ giao diện Java: đây là cách dễ dàng, ít tốn kém công sức nhưng không linh hoạt.

Giao diện của trong dịch vụ đầu tiên này là mô tả GWSDL, để tương ứng với giao diện Java sau:

```
public interface HuycvFactory
{
      public void add(int a);
      public void subtract(int a);
      public int getValue();
}
```

Chúng ta sẽ đi sâu vào chi tiết của mô tả giao diện GWSDL.

Như ta đã biết, file mô tả dịch vụ được viết theo chuẩn XML. Do đó, mở đầu file này luôn là thẻ khai báo

```
<?xml version="1.0" encoding="UTF-8"?>
```

Tiếp theo, ta khai báo phần tử gốc của file, tức là phần tử "definitions".

```
<definitions                                    name="MathServiceService"
targetNamespace="http://www.MathService/MathServiceService"
xmlns:tns="http://www.MathService/MathServiceService"
xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExt
ensions"
xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">
...
</definication>
```

Phần tử này có các thuộc tính như sau:

Name: tên của phần tử gốc, tức là tên của dịch vụ lưới. Ở ví dụ này là 'MathServiceService'

targetNamespace: đây là không gian tên tại đó có chứa các giao diện và các thao tác của dịch vụ. Không gian tên này có tính chất logic. Bởi vì, nơi lưu trữ thực chất của các file cấu thành nên dịch vụ lưới sẽ được tìm trong file namespace2package.mappings. Ở ví dụ này là "http://www.MathService/MathServiceService"

xmlns:tn: thuộc tính này có giá trị trùng với thuộc tính targetNamespace. Ở ví dụ này là "http://www.MathService/MathServiceService"

Các thuộc tính còn lại có các giá trị mặc định.

Tiếp theo, chúng ta cần khai báo để có thể sử dụng file mở rộng theo chuẩn OGSI. Giá trị cần nhập cũng là mặc định:

```
<import location="../../ogsi/ogsi.gwsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
```

Đến đây, chúng ta bắt đầu định nghĩa giao diện cho dịch vụ (còn gọi là portType) sử dụng thẻ <gwsdl:portType>. Do dịch vụ của ta sử dụng cơ chế Factory nên giao diện này phải được kế thừa từ lớp cơ sở ogsi:GridService. Khai báo như sau:

```
<gwsdl:portType name="MathServicePortType"
extends="ogsi:GridService">
```

Tiếp theo, mỗi toán tử được khai báo dưới dạng một phần tử có tên là operation có các phần tử con là name, input message, output message và fault. Với mỗi phần tử con này, có các thuộc tính là message xác định các thông điệp vào, ra cho mỗi toán tử. Như vậy, ta cần phải định nghĩa các thông điệp này.

Mỗi phần tử thông điệp sẽ trỏ tới một phần tử XML cụ thể có không gian tên xác định. Ví dụ:

```
<message name="addOutputMessage">
    <part name="parameters" element="tns:addResponse"/>
</message>
```

Các phần tử XML cụ thể này được định nghĩa bằng XML Schema dưới dạng thẻ <types>. Ví dụ:

```
<types>
<xsd:schema
targetNamespace="http://www.MathService/MathServiceService"
attributeFormDefault="qualified" elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema">
    <!--BEGIN ELEMENT DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->
    <xsd:element name="getValue">
        <xsd:complexType/>
    </xsd:element>
```

```
...
<!--END ELEMENT DEFINITIONS -->
</xsd:schema>
</types>
```

Như ta thấy, các thuộc tính của thẻ con <xsd :schema> có giá trị là mặc định. Ta chú ý tới thẻ <xsd:element>. Các tham số của phần tử add được định nghĩa trong thẻ con <xsd:sequence> của thẻ <xsd:complexType>. Trong ví dụ trên, hàm add của dịch vụ lưới có một tham số đầu vào kiểu số nguyên.

Toàn bộ mã nguồn của file này như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions                                    name="MathServiceService"
targetNamespace="http://www.MathService/MathServiceService"
xmlns:tns="http://www.MathService/MathServiceService"
xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExt
ensions"
xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">

<import                                  location="../../ogsi/ogsi.gwsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>

<types>
<xsd:schema
targetNamespace="http://www.MathService/MathServiceService"
attributeFormDefault="qualified"       elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema">

    <!--BEGIN ELEMENT DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->
    <xsd:element name="getValue">
        <xsd:complexType/>
    </xsd:element>
    <xsd:element name="getValueResponse">
        <xsd:complexType>
```

```
                <xsd:sequence>
                    <xsd:element name="value" type="xsd:int"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="subtract">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="arg1" type="xsd:int"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="subtractResponse">
            <xsd:complexType/>
        </xsd:element>
        <xsd:element name="add">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="arg1" type="xsd:int"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="addResponse">
            <xsd:complexType/>
        </xsd:element>
        <!--END ELEMENT DEFINITIONS -->
    </xsd:schema>
    </types>

    <!--BEGIN MESSAGE DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->
```

```
<message name="getValueInputMessage">
      <part name="parameters" element="tns:getValue"/>
</message>

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

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

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

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

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

<!--END MESSAGE DEFINITIONS -->

<gwsdl:portType                              name="MathServicePortType"
extends="ogsi:GridService">
      <!--BEGIN OPERATION DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -
->
      <operation name="getValue">
            <input message="tns:getValueInputMessage"/>
```

```
            <output message="tns:getValueOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
        </operation>
        <operation name="subtract">
            <input message="tns:subtractInputMessage"/>
            <output message="tns:subtractOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
        </operation>
        <operation name="add">
            <input message="tns:addInputMessage"/>
            <output message="tns:addOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
        </operation>
        <!--END OPERATION DEFINITIONS -->
</gwsdl:portType>
</definitions>
```

Tóm lại, để viết một file mô tả dịch vụ lưới, ta cần làm theo các bước như sau:

+ Viết phần tử gốc <definitions>

+ Viết định nghĩa giao diện <gwsdl:PortType>

+ Viết các thẻ <message> cho mỗi hàm tính toán trong giao diện đã viết ở bước trên.

+ Viết các thẻ <types>.

Tuy nhiên, đây là những bước rất cơ bản để viết một file mô tả dịch vụ lưới. Ngoài ra, còn có thể có các cách khác ngắn gọn hơn và các kỹ thuật để ta khai báo các hàm mà tham số truyền vào dưới dạng phức tạp.

Phân biệt giữa file mô tả dịch vụ lưới và file mô tả dịch vụ Web.

Như đã trình bày ở phần trước, file mô tả dịch vụ lưới là sự mở rộng của file mô tả dịch vụ Web. Tuy nhiên, điều này không hoàn toàn đúng. File mô tả dịch vụ lưới có các đặc tính mới mà chuẩn WSDL 1.1 không hỗ trợ. Hiện nay phiên bản WSDL 1.2 có hỗ trợ cho việc mô tả dịch lưới và vẫn chưa thành chuẩn thay thế cho WSDL 1.1. Chính vì vậy, các nhà phát triển đã đưa ra phiên bản khác là GWSDL được coi là giải pháp tạm thời cho dịch vụ lưới. Trong tương lai, dịch vụ lưới sẽ được mô tả bằng ngôn ngữ WSDL 1.2.

Có hai đặc tính đáng chú ý nhất của file mô tả dịch vụ lưới so với chuẩn WSDL 1.1. Đầu tiên, file mô tả dịch vụ lưới hỗ trợ thuộc tính cho phép khai báo sự kế thừa giao diện. Ví dụ:

```
<gwsdl:portType                                    name="MathServicePortType"
extends="ogsi:GridService">
```

Như vậy, giao diện của dịch vụ MathService được kế thừa từ giao diện là GridSerive.

Đặc tính thứ hai là trong file mô tả dịch vụ lưới cho phép ta khai báo các dữ liệu dịch vụ (Service Data).

2.2.2. Viết file mã nguồn cài đặt các giao diện đã định nghĩa

Bước này thực chất là ta phải viết một file thực thi dịch vụ, thực thi các nhiệm vụ mà dịch vụ của bạn cung cấp. Trong ví dụ đầu tiên này (và trong các ví dụ tiếp theo). Chúng ta cung cấp cho người sử dụng dịch vụ cộng, trừ hai số nguyên (có thể đơn giản quá! - không sao, mục đích của chúng ta là hiểu cách triển khai một dịch vụ lưới).

Toàn bộ mã nguồn của file này như sau:

```
package MathService.impl;


import org.globus.ogsa.impl.ogsi.GridServiceImpl;
import MathService.stubs.MathServicePortType;
import java.rmi.RemoteException;


public class MathServiceImpl extends GridServiceImpl implements
MathServicePortType
{
      private int value = 0;
      public MathServiceImpl()
      {
            super("MathService");
      }


      // Insert implementation here
      public void add(int a) throws RemoteException
      {
      value = value + a;
      }
      public void subtract(int a) throws RemoteException
      {
```

```
        value = value - a;
        }
        public int getValue() throws RemoteException
        {
        return value;
        }
}
```

2.2.3. Viết file cấu hình triển khai dịch vụ

Cho tới thời điểm này chúng ta đã viết xong hai phần quan trọng nhất của dịch vụ lưới : giao diện dịch vụ (GWSDL) và cài đặt dịch vụ (Java). Công việc tiếp theo của chúng ta là kết hợp tất cả các thành phần lại và cài đặt chúng trên một dịch vụ lưới server - bước "triển khai" dịch vụ lưới.

Trong bước triển khai này chúng ta phải có một thành phần rất quan trọng là file "mô tả triển khai". File này sẽ cho server biết cách thức triển khai dịch vụ lưới của chúng ta (ví dụ như cho biết GSH của dịch vụ lưới cần triển khai). File mô tả này được viết theo định dạng WSDD (Web Service Deployment Descriptor).

Cũng giống như file mô tả dịch vụ lưới, file mô tả triển khai dịch vụ được viết theo chuẩn XML. Ngoài ra, file cấu hình này được sử dụng khi triển khai trên Web Server Apache Axis.

Thẻ gốc của file này là thẻ <deployment>, trong đó bao gồm các thẻ con chứa các tham số về dịch vụ lưới. Thẻ gốc có các thuộc tính name, xmlns và xmlns:java nhận các giá trị mặc định.

```
<deployment                                    name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
        <service name="MathService" provider="Handler" style="wrapped">
...
        </service>
</deployment>
```

Thẻ này xác định tên của dịch vụ. Khi ta triển khai dịch vụ lưới trên nền GT3 thì địa chỉ hợp lệ của dịch vụ sẽ là: http://localhost:8080/ogsa/services/MathService.

```
<parameter name="name" value="MathService (Factory)"/>
```

Thẻ này dùng để định tên của dịch vụ. Trong trường hợp này, dịch vụ của ta có tên là MathService.

```
<parameter name="instance-name" value="MathService (Instance)"/>
```

Viết file sử dụng thẻ này xác định tên của thể hiện của dịch vụ lưới mỗi khi người sử dụng yêu cầu máy chủ tạo một thể hiện của dịch vụ lưới bằng lệnh ogsi-create-service.

```
<parameter                                    name="instance-className"
value="MathService.stubs.MathServicePortType"/>
```

Thẻ này xác định lớp đóng vai trò giao diện của dịch vụ đối với người sử dụng.

```
<parameter                                    name="instance-schemaPath"
value="schema/gt3ide/MathServiceService/MathService_service.wsdl"/>
```

Đường dẫn tới file mô tả dịch vụ lưới được chỉ ra trong thẻ này.

```
<parameter                                    name="instance-baseClassName"
value="MathService.impl.MathServiceImpl"/>
```

Thẻ này xác định lớp thực thi dịch vụ lưới.

```
<parameter name="allowedMethods" value="*"/>
```

Thẻ này dùng để thông báo cho Web Server biết tất cả phương thức của dịch vụ lưới đều khả dụng đối với người sử dụng.

```
<parameter name="persistent" value="true"/>
```

Thẻ này xác định tính tồn tại lâu dài của dịch vụ (cùng với Web Server).

```
<parameter name="className" value="org.gridforum.ogsi.Factory"/>
```

Trong ví dụ của ta, dịch vụ lưới được viết dựa trên kỹ thuật Factory. Do đó, lớp cơ sở hỗ trợ kỹ thuật này phải được xác định thông qua tham số này.

```
<parameter                                    name="baseClassName"
value="org.globus.ogsa.impl.ogsi.GridServiceImpl"/>
```

Như đã trình bày ở phần trước, tất cả các dịch vụ do nhà phát triển tạo ra để cung cấp cho người sử dụng đều phải kế thừa từ lớp cơ sở là GridServiceImpl. Tham số này xác định lớp cơ sở cho dịch vụ lưới của ta.

```
<parameter
 name="schemaPath" value="schema/ogsi/ogsi_factory_service.wsdl"/>
```

Tham số này chỉ ra đường dẫn tới file mô tả dịch vụ cơ sở cho các dịch vụ lưới. Giá trị của tham số này là mặc định.

```
<parameter                                    name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>
```

Thẻ này có giá trị mặc định.

```
<parameter                                    name="factoryCallback"
value="org.globus.ogsa.impl.ogsi.DynamicFactoryCallbackImpl"/>
```

Thẻ này xác định cơ chế gọi ngược (callback) khi dịch vụ được triển khai.

```
<parameter                                    name="operationProviders"
value="org.globus.ogsa.impl.ogsi.FactoryProvider"/>
```

Thẻ này nhận giá trị mặc định, có tác dụng thông báo cho Web Server biết là ta có sử dụng cơ chế nhà cung cấp chức năng (operation provider) hay không. Dịch vụ của ta dùng cơ chế Factory nên giá trị ở đây là mặc định.

Các tham số chung: mọi dịch vụ lưới đều có 3 tham số chung như sau

```
<!-- Start common parameters -->
<parameter name="allowedMethods" value="*"/>
<parameter name="persistent" value="true"/>
<parameter                                    name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>
```

Toàn bộ mã nguồn của file này như sau:

```
<?xml version="1.0"?>
<deployment                                  name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
     xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
     <service name="MathService" provider="Handler" style="wrapped">
          <parameter name="name" value="MathService (Factory)"/>
          <parameter     name="instance-name"     value="MathService
(Instance)"/>
          <parameter                          name="instance-className"
value="MathService.stubs.MathServicePortType"/>
          <parameter                        name="instance-schemaPath"
value="schema/gt3ide/MathServiceService/MathService_service.wsdl"/>
          <parameter                     name="instance-baseClassName"
value="MathService.impl.MathServiceImpl"/>

          <parameter name="allowedMethods" value="*"/>
          <parameter name="persistent" value="true"/>
          <parameter                                  name="className"
value="org.gridforum.ogsi.Factory"/>
```

```
            <parameter                                    name="baseClassName"
value="org.globus.ogsa.impl.ogsi.GridServiceImpl"/>
            <parameter                                      name="schemaPath"
value="schema/ogsi/ogsi_factory_service.wsdl"/>
            <parameter                                    name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>
            <parameter                                 name="factoryCallback"
value="org.globus.ogsa.impl.ogsi.DynamicFactoryCallbackImpl"/>
            <parameter                              name="operationProviders"
value="org.globus.ogsa.impl.ogsi.FactoryProvider"/>
      </service>
</deployment>
```

2.2.4. Đóng gói dịch vụ

Hiện tại chúng ta đã có các tệp:

(1) Giao diện dịch vụ - GWSDL

(2) Cài đặt dịch vụ - Java

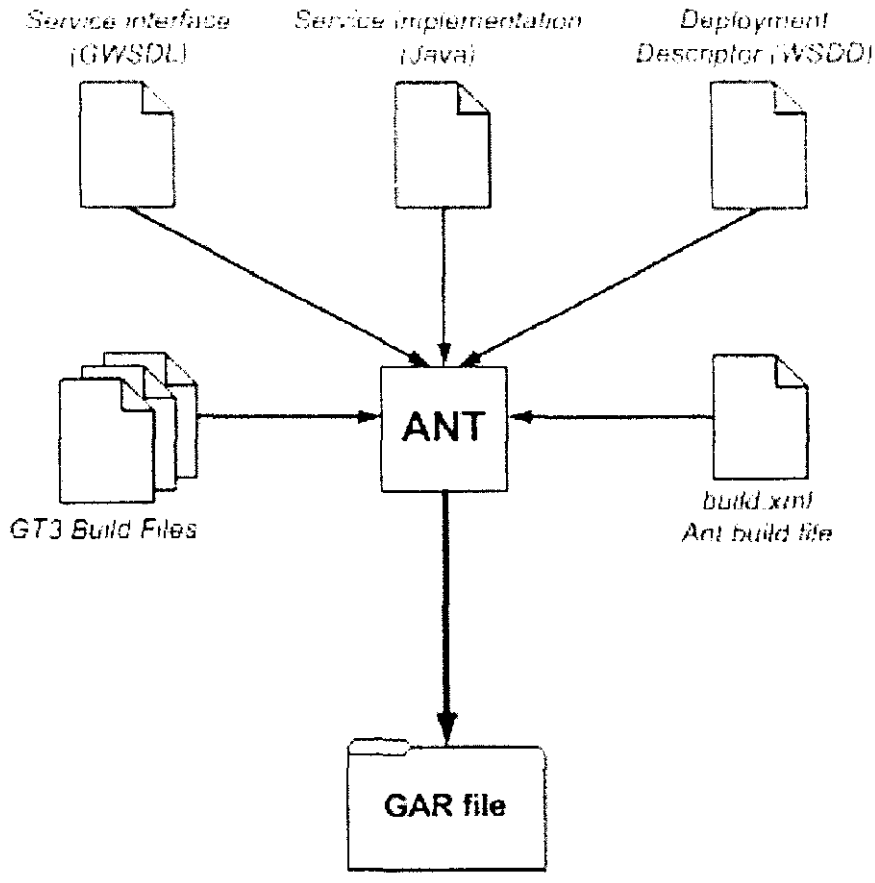
(3) Mô tả triển khai dịch vụ

Chúng ta cần biết cách thức lưu trữ các tệp này trên trình chứa. Chúng ta sẽ sử dụng các file trên để sinh ra một file lưu trữ Grid Archive (GAR). File này là một file đơn và chứa tất cả các file và các thông tin mà trình chứa dịch vụ lưới cần để triển khai dịch vụ của chúng ta.

Công việc tạo file GAR khá phức tạp, bao gồm các bước sau :

- Chuyển đổi file GWSDL sang WSDL
- Tạo các lớp stub từ WSDL
- Biên dịch các lớp stub
- Biên dịch các file cài đặt dịch vụ
- Tổ chức tất cả các file vào trong một cấu trúc thư mục riêng biệt.

Để thực hiện các công việc trên chúng ta cần tới công cụ ANT - công cụ để build các file Java (phát triển bởi Apache Software Foundation [http://www.apache.org/])



2.2.5. Triển khai dịch vụ lên lưới

-Triển khai dịch vụ lưới bằng lệnh.

```
# cd $GLOBUS_LOCATION
# ant deploy -Dgar.name=.../MathService/build/lib/MathService.gar
```

-Nếu thành công, chúc mừng bạn. Mọi việc đến đây là hoàn thành. Bạn đã sẵn sàng cung cấp dịch vụ cho người sử dụng.

2.2.6. Viết file client triệu gọi dịch vụ

File thực thi phía máy khách có thể tách biệt với các file phía cung cấp dịch vụ. Tuy nhiên, nếu bạn muốn cung cấp một dịch vụ hoàn chỉnh cho người sử dụng thì bạn nên đóng gói luôn file này vào file GAR. Bạn nhấn chuột phải vào gói MathService.impl, chọn New → Class. Đặt tên file này là MathServiceClient1. Trong ví dụ này, ta minh họa hai cơ chế sử dụng dịch vụ là dài hạn và ngắn hạn. Theo cơ chế dài hạn là một thể hiện của dịch vụ sẽ tồn tại cùng với trình chứa mỗi khi được khởi tạo. Đối với cơ chế ngắn hạn, thể hiện của dịch vụ sẽ bị huỷ mỗi khi thực hiện xong dịch vụ. Vì vậy, bạn tạo thêm một lớp nữa có tên là MathServiceClient2 theo cách trên.

- Cơ chế dịch vụ dài hạn (MathServiceClient1.java)

Trong file thực thi phía máy khách dạng này, ta chỉ lấy về tham chiếu của dịch vụ lưới thông qua đoạn mã:

```
            //     Get command-line arguments
            URL GSH = new java.net.URL(args[0]);
            //     Get a reference to the MathService instance
            MathServiceServiceGridLocator mathServiceLocator = new
MathServiceServiceGridLocator();
            MathServicePortType                    math                =
mathServiceLocator.getMathServiceServicePort(GSH);
```

Sau khi đã có tham chiếu tới dịch vụ lưới, người sử dụng có thể gọi các phương thức của dịch vụ lưới như là gọi một đối tượng trên máy khách:

```
            math.add(a);
            System.out.println("Added " + a);
            //     Get current value through remote method 'getValue'
            int value = math.getValue();
            System.out.println("Current value: " + value);
```

Toàn bộ mã nguồn của file này như sau:

```
package MathService.impl;

import MathService.stubs.service.MathServiceServiceGridLocator;
import MathService.stubs.MathServicePortType;
import java.net.URL;

public class MathServiceClient1 {
      public static void main(String[] args)
      {
      try
      {
            //     Get command-line arguments
            URL GSH = new java.net.URL(args[0]);
            int a = Integer.parseInt(args[1]);
```

```
            //     Get a reference to the MathService instance
            MathServiceServiceGridLocator mathServiceLocator = new
MathServiceServiceGridLocator();
            MathServicePortType                    math                    =
mathServiceLocator.getMathServiceServicePort(GSH);
            //     Call remote method 'add'
            math.add(a);
            System.out.println("Added " + a);
            //     Get current value through remote method 'getValue'
            int value = math.getValue();
            System.out.println("Current value: " + value);
      }
      catch(Exception e)
      {
            System.out.println("ERROR!");
            e.printStackTrace();
      }
}
}
```

- Cơ chế dịch vụ ngắn hạn (HuycvMathClient2.java)

Đối với người sử dụng ở dạng thứ hai, ta không lấy về tham chiếu tới dịch vụ. Thay vào đó, người sử dụng yêu cầu dịch vụ tạo ra một thể hiện để phục vụ riêng và lấy tham chiếu tới thể hiện này. Các thao tác trên được thực hiện bằng đoạn mã:

```
//Get command-line arguments
            URL GSH = new java.net.URL(args[0]);
            int a = Integer.parseInt(args[1]);

            // Create a new instance
            OGSIServiceGridLocator                    gridLocator=new
OGSIServiceGridLocator();
            Factory fac=gridLocator.getFactoryPort(GSH);
            GridServiceFactory mathFac=new GridServiceFactory(fac);
```

```
            LocatorType loc=mathFac.createService();


            // Get a reference to the MathService instance
            MathServiceServiceGridLocator mathServiceLocator = new
MathServiceServiceGridLocator();
            MathServicePortType                    math                    =
mathServiceLocator.getMathServiceServicePort(loc);
```

Tiếp theo, việc gọi các phương thức của dịch vụ lưới là không đổi so với người sử dụng ở phương pháp thứ nhất.

```
      //     Call remote method 'add'
            math.add(a);
            System.out.println("Added " + a);
            //     Get current value through remote method 'getValue'
            int value = math.getValue();
            System.out.println("Current value: " + value);
```

Tuy nhiên, sau khi sử dụng xong dịch vụ và lấy được kết quả, người sử dụng phải huỷ thể hiện dịch vụ này:

```
// Destroy the instance
            GridService gs=gridLocator.getGridServicePort(loc);
            gs.destroy();
```

Toàn bộ mã nguồn của file này như sau:

```
package MathService.impl;


import org.gridforum.ogsi.OGSIServiceGridLocator;
import org.gridforum.ogsi.Factory;
import org.gridforum.ogsi.LocatorType;
import org.globus.ogsa.utils.GridServiceFactory;
import org.gridforum.ogsi.GridService;


import MathService.stubs.service.MathServiceServiceGridLocator;
import MathService.stubs.MathServicePortType;
import java.net.URL;
```

```
public class MathServiceClient2 {
    public static void main(String[] args)
    {
    try
    {
        //    Get command-line arguments
        URL GSH = new java.net.URL(args[0]);
        int a = Integer.parseInt(args[1]);

        // Create a new instance
        OGSIServiceGridLocator gridLocator=new
OGSIServiceGridLocator();
        Factory fac=gridLocator.getFactoryPort(GSH);
        GridServiceFactory mathFac=new GridServiceFactory(fac);
        LocatorType loc=mathFac.createService();

        // Get a reference to the MathService instance
        MathServiceServiceGridLocator mathServiceLocator = new
MathServiceServiceGridLocator();
        MathServicePortType math =
mathServiceLocator.getMathServiceServicePort(loc);

        //    Call remote method 'add'
        math.add(a);
        System.out.println("Added " + a);
        //    Get current value through remote method 'getValue'
        int value = math.getValue();
        System.out.println("Current value: " + value);

        // Destroy the instance
        GridService gs=gridLocator.getGridServicePort(loc);
```

```
            gs.destroy();


        }
        catch(Exception e)
        {
            System.out.println("ERROR!");
            e.printStackTrace();
        }
}
}
```

Triệu gọi dịch vụ MathService

Sau khi khởi động trình chứa GT3. Trong số các dịch vụ liệt kê sẽ có dịch vụ:

http://localhost:8080/ogsa/services/MathService

Bạn mởi một terminal khác, tạo proxy cho người sử dụng và thực thi các lệnh. Các lệnh dưới đây được thực hiện với giả sử người sử dụng lưới là root. Các người sử dụng khác bạn tiến hành hoàn toàn tương tự.

```
# grid-proxy-init
```

Chạy lệnh sau để thiết lập biến môi trường người sử dụng:

```
# . $GLOBUS_LOCATION/etc/globus-devel-env.sh
```

Khi sử dụng cơ chế dịch vụ dài hạn, bạn phải yêu cầu trình chứa tạo ra một thể hiện của dịch vụ trước khi muốn sử dụng, dùng lệnh:

```
#                                                    ogsi-create-service
http://localhost:8080/ogsa/services/MathService
```

Sau khi gõ lệnh này, trình chứa sẽ tạo ra một thể hiện của MathService với GSH riêng biệt (dưới dạng một hash_value), chẳng hạn

http://localhost:8080/ogsa/services/MathService/hash-1265354-1139387699328

```
# cd .../MathService/build/classes

#              java              MathService/impl/MathServiceClient1
http://localhost:8080/ogsa/services/MathService/hash-1265354-
1139387699328 10
```

Nếu thành công, kết quả sẽ hiển thị ra màn hình là tổng của số nguyên đầu vào và giá trị cũ của dịch vụ cùng với giá trị hiện thời của dịch vụ (thể hiện tính trạng thái của dịch vụ lưới). Bạn có thể chạy lệnh này nhiều lần để kiểm tra kết quả.

Đối với cơ chế ngắn hạn, mỗi khi sử dụng dịch vụ xong, thể hiện của dịch vụ sẽ bị huỷ. Vì vậy, bạn không cần khởi tạo một thể hiện của dịch vụ. Bạn thực hiện lệnh:

```
# java MathService/impl/MathServiceClient2
http://localhost:8080/ogsa/services/MathService 10
```

Do mỗi lần sử dụng xong, thể hiện của dịch vụ bị huỷ nên giá trị trả về luôn không đổi với các lần gọi khác nhau và với cùng giá trị đầu vào.

2.2.7. Cách xây dựng một dịch vụ lưới trên GT3 bằng công cụ Eclipse

Sau khi hoàn thành cài đặt Eclipse, bước tiếp theo là bạn sẽ tạo một dịch vụ lưới mới trên bộ công cụ Eclipse 3.0. Bạn khởi động chương trình:

```
# cd …/eclipse

# ./eclipse
```

- Chương trình có thể hỏi bạn không gian tên cho dịch vụ mới của bạn. Bạn chọn mặc định là $Eclipse_Directory/workspace.

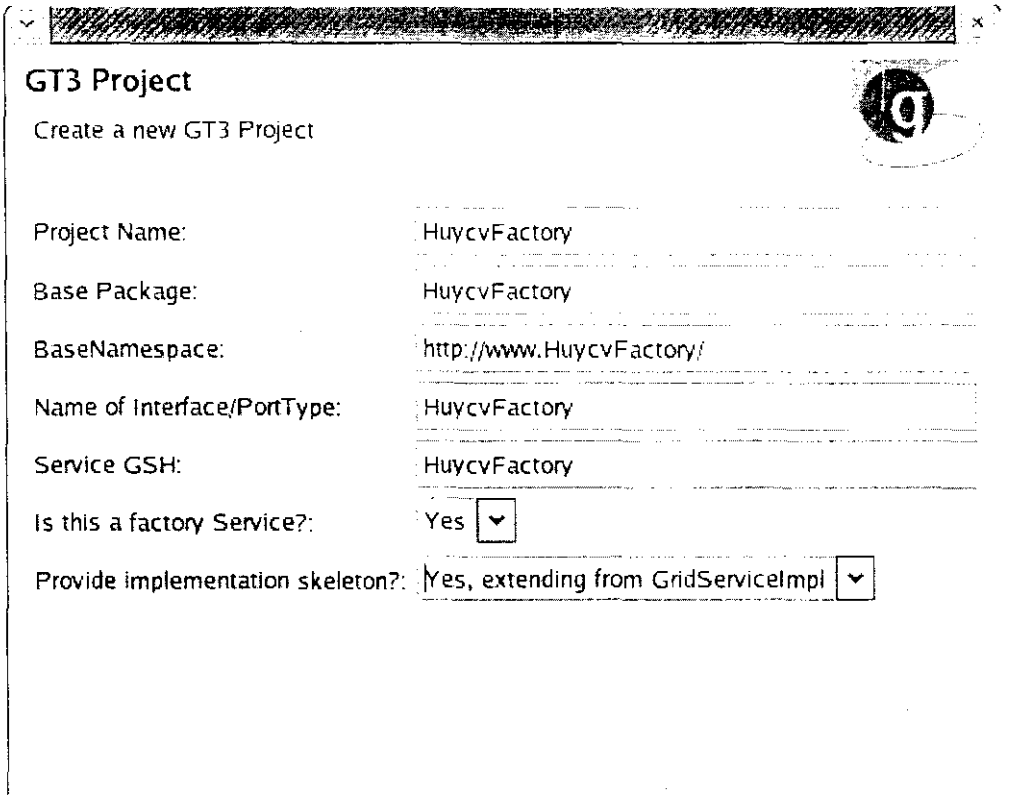
- Sau đó, bạn tạo một dịch vụ lưới:

```
File -> New -> Project -> GT3 Project
```

- Nếu bạn sử dụng Eclipse lần đầu tiên, chương trình sẽ hỏi đường dẫn tới thư mục chứa GT3. Bạn gõ đường dẫn:

```
/usr/local/gt3.2.1
```

-Tiếp theo, Eclipse hiện lên cửa sổ giao diện của GT3 project, bạn phải điền các tham số như sau:



Project Name: Đây là tên của dịch vụ bạn muốn tạo. Trong ví dụ này, đó là: HuycvFactory

Base Package: Tên của gói chứa các file thực thi dịch vụ và file mô tả triển khai dịch vụ (WSDD). Ví dụ: HuycvFactory.

Base Namespace: Không gian tên thể hiện trong file mô tả dịch vụ (GWSDL). Thực ra, bạn không cần quá bận tâm vào không gian tên này vì nó sẽ được ánh xạ thông qua file namespace2package.mappings (file này trong thư mục $HuycvFactory). Tuy nhiên, tên này phải là một địa chỉ URI hợp lệ. Trong ví dụ này, ta lấy địa chỉ là http://www.HuycvFactory.

Name of Interface/PortType: Đây là tên cơ sở cho nhiều giao diện khác như stubs, portType. Ví dụ này, ta dùng giá trị HuycvFactory.

Service GSH: Địa chỉ URL cho người sử dụng. Nếu như bạn triển khai dịch vụ trên GT3 thì các GSH luôn bắt đầu bằng http://localhost:8080/ogsa/services. Muốn cho dịch vụ của bạn có địa chỉ là http://localhost:8080/ogsa/services/HuycvFactory. Bạn gõ giá trị HuycvFactory.

Is this a factory service?: Eclipse hỗ trợ bạn viết dịch vụ lưới theo hai dạng. Dạng dịch vụ luôn tồn tại (cùng với container) và một loại có kiểm soát thông qua cơ chế Factory. Trong ví dụ này, chúng ta đang dùng cơ chế Factory. Vì vậy, bạn trả lời "Yes"

Provide implementation skeleton?: Đây là tuỳ chọn hỗ trợ bạn các đoạn mã trong các file thực thi dịch vụ. Nếu bạn sử dụng kỹ thuật Factory thì dịch vụ của bạn phải kế thừa lớp cơ bản (theo đặc tả OGSI) là lớp GridServiceImpl. Ngoài ra, nếu ta dùng kỹ thuật operation provider, thì bạn chọn Java class that acts as an operation provider. Trong ví của ta đang triển khai, bạn chọn Java class that extends from GridServiceImpl. Tất nhiên, nếu không sử dụng hỗ trợ này, bạn chọn no Java class at all.

Sau khi đã điền hết các tham số như trên, chương trình có thể hỏi bạn chuyển sang giao diện thiết kế của Globus Toolkit 3. Bạn nhấp chuột để đồng ý. Nếu bạn chú ý thì ở góc phải phía trên của chương trình có biểu tượng GT3.

Chú ý: Sau khi chọn xong, chương trình sẽ chuyển bạn đến file thực thi dịch vụ. Màn hình sẽ báo là có lỗi (gạch đỏ dưới đoạn mã có lỗi). Tuy nhiên, bạn không cần lo lắng quá vì do bạn chưa tạo stub nên ban đầu sẽ có lỗi này.

Đến đây, tất cả đã sẵn sàng cho việc lập trình.

2.2.7.1. Phía server

- Lựa chọn chức năng AutoUpdate

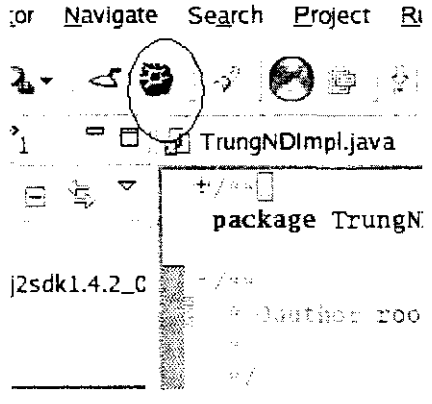
Vào menu sau để chọn : Window/ Preference/ GT3 Prefences/ Automatic GWSDL Update.

- Viết file .java

Như đã nói ở trên đây là các file cài đặt các chức năng mà dịch vụ của chúng ta cung cấp.

- Build stub

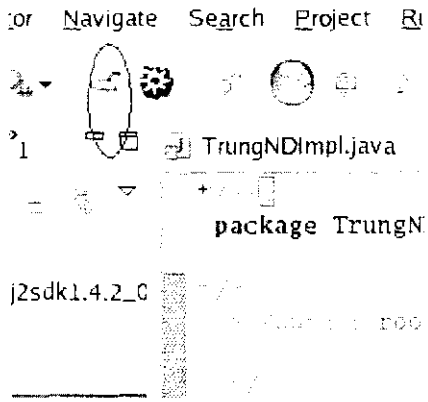
Lựa chọn chức năng build stub trên toolbar



2.2.7.2. Phía client

- Viết các file client

- Build gar



2.2.7.3. Dùng Eclipse để chạy dịch vụ lưới

- Deploy dịch vụ (console)

```
# cd $GLOBUS_LOCATION
# ant deploy -Dgar.name=.../MathService/build/lib/MathService.gar
```

- Chạy dịch vụ

+ Phía server, dùng các lệnh:

```
# su globus_xx
# cd $GLOBUS_LOCATION
# globus-start-container
```

+ Phía client (dùng Eclipse)

Vào menu: Run/ Run.../Configurations, chọn chạy Java Application. Sau đó chọn file Client cần chạy.

Thêm tham số cho lệnh chạy : Chọn trong phần Arguments, thêm các tham số cần thiết cần cung cấp như khi chạy client trên console trong ô Program Arguments. Chẳng hạn: http://localhost:8080/ogsa/services/MathService 10

Thêm các gói .jar vào CLASSPATH : Import các gói .jar từ thư mục $GLOBUS_LOCATION/lib vào Classpath/Bootstrap Entries, dùng chức năng Add External JARs. Chú ý loại bỏ các gói ko phải là .jar

Chọn nút Run.

2.3. Các cơ chế nâng cao

2.3.1. Cơ chế nhà cung cấp

2.3.1.1. Mô hình kế thừa và nhà cung cấp

- Bản chất cơ chế Operation Provider ?

+ Cách tiếp cận Factory: Khi triển khai một dịch vụ lưới, ta phải tạo một lớp Java kế thừa từ lớp GridServiceImpl

Với cách tiếp cận này, ta có thể có được tất cả các chức năng cơ bản của một dịch vụ lưới được cung cấp bởi lớp cơ sở. Cách tiếp cận này được gọi là triển khai bằng phương pháp kế thừa.

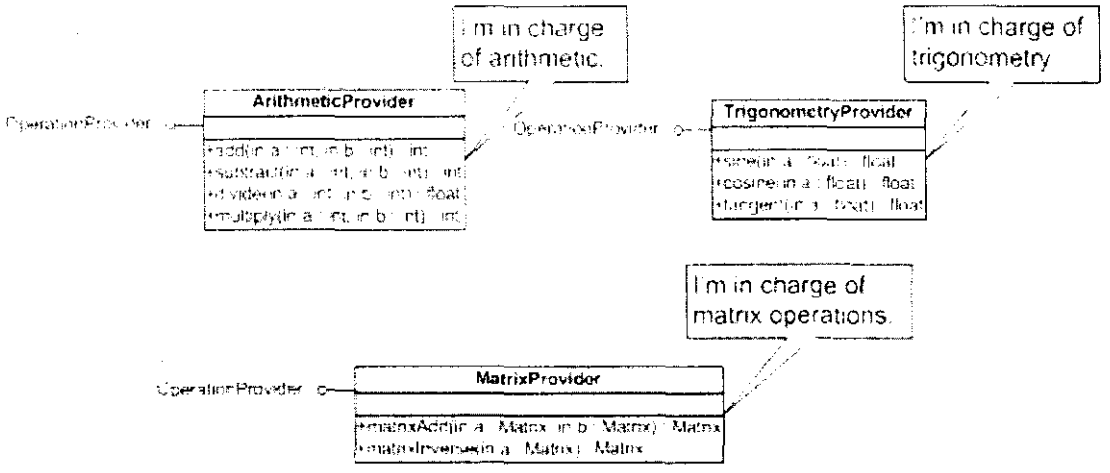
Vì lớp cơ sở GridServiceImpl đã có tất cả các chức năng cơ bản của một dịch vụ lưới, nên lớp MathImpl chỉ đơn giản cung cấp các chức năng được định nghĩa trong portType:

[MathService.gwsdl]

```
<gwsdl:portType                                    name="MathServicePortType"
extends="ogsi:GridService">
      <!--BEGIN OPERATION DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -
->
      <operation name="getValue">
            <input message="tns:getValueInputMessage"/>
            <output message="tns:getValueOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <operation name="subtract">
            <input message="tns:subtractInputMessage"/>
            <output message="tns:subtractOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <operation name="add">
            <input message="tns:addInputMessage"/>
            <output message="tns:addOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <!--END OPERATION DEFINITIONS -->
</gwsdl:portType>
```

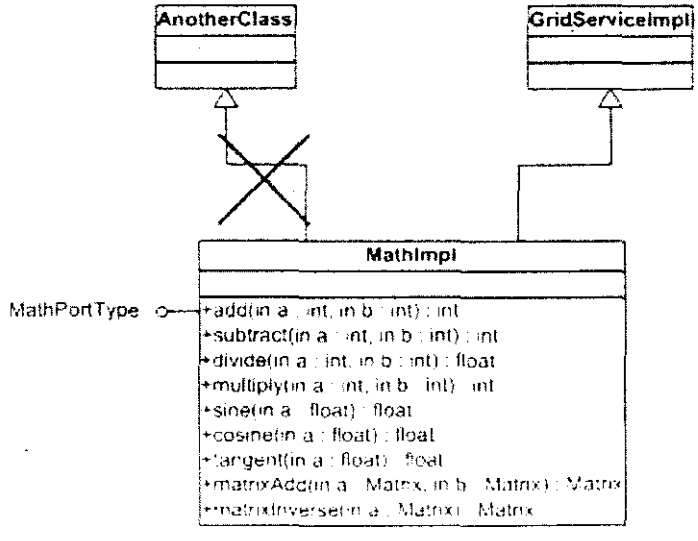
+ GT3 cung cấp một cách tiếp cận khác: triển khai bằng phương pháp chuyển giao (frng ngôn ngữ của GT3 gọi là cơ chế nhà cung cấp chức năng). Đây là một cách tiếp cận quen thuộc thiết kế hướng đối tượng.

Với cách tiếp cận này, ta có thể chia các chức năng trong portType thành một số lớp. Chẳng hạn chia MathImpl thành 3 lớp: các phép toán số học, các phép toán hình học và các phép toán ma trận. Mỗi lớp này được gọi là một nhà cung cấp.

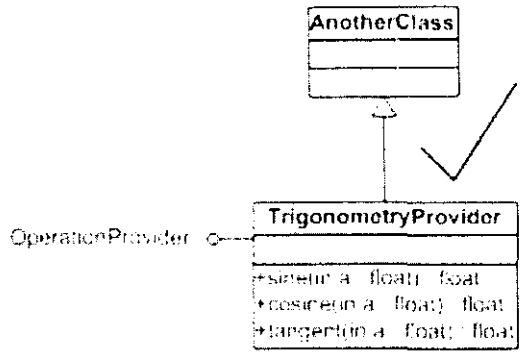


Chú ý cách các lớp này không kế thừa từ bất kỳ lớp nào. Chúng chỉ thực thi một giao diện gọi là Operation Provider. Tất nhiên, bạn sẽ băn khoăn rằng, nếu chúng không kế thừa từ các lớp cơ sở thì làm thế nào chúng có thể có các chức năng cơ bản của một dịch vụ lưới được. Trên thực tế, GridServiceImpl vẫn tồn tại trong cách tiếp cận này. Sự khác nhau chỉ là ta không kế thừa trực tiếp từ lớp này. Ta chỉ cần báo cho trình chứa dịch vụ lưới biết rằng GridServiceImpl sẽ cung cấp các chức năng cơ bản. Điều này được thực hiện thông qua file mô tả triển khai mà ta sẽ trình bày ở phần sau.

Với cách tiếp cận theo kế thừa, lớp MathImpl rõ ràng không thể kế thừa từ một lớp khác nữa, do nó đã kế thừa lớp GridServiceImpl (Java không hỗ trợ đa kế thừa).



Sử dụng cách tiếp cận theo mô hình chuyển giao, các lớp cung cấp có thể kế thừa từ bất kì lớp nào.



Cách tiếp cận theo mô hình chuyển giao còn có một số ưu điểm khác, đó là tăng tính modun, tính tái sử dụng (ta có thể phân phối các chức năng thành một số nhà cung cấp, sau đó tái sử dụng các nhà cung cấp này trong các dịch vụ lưới khác nhau).

Nhược điểm duy nhất của cách tiếp cận theo mô hình chuyển giao là cần viết mã chương trình nhiều hơn một ít. File mô tả triển khai cần thêm một vài dòng, và bạn cần viết mã thực thi cho các phương thức trong giao diện OperationProvider.

Hai cách tiếp cận trên không loại trừ lẫn nhau. Ta có thể thực thi một số phần trong portType của một lớp bằng cách kế thừa và thực thi các phần còn lại bằng cơ chế nhà cung cấp. Thực tế, đây là kỹ thuật thường dùng trong GT3, vì bộ công cụ chứa rất nhiều các nhà cung cấp mà ta có thể sử dụng để thêm các chức năng cho dịch vụ lưới của mình. Chẳng hạn, phần sau ta sẽ thấy, để sử dụng cơ chế thông báo cho một dịch vụ, ta không phải tự viết thêm một dòng mã nào cả. Chúng ta đơn giản chỉ sử dụng nhà cung cấp cơ chế thông báo để thêm chức năng này vào dịch vụ của chúng ta.

2.3.1.2. Xây dựng dịch vụ dùng cơ chế nhà cung cấp

- Định nghĩa giao diện dịch vụ GWSDL

Trong ví dụ này, ta sử dụng lại file định nghĩa giao diện dịch vụ GWSDL trong ví dụ trước. Sở dĩ có thể làm được điều này là do chừng nào mà các giao diện (interface) chưa thay đổi (nghĩa là ta không thêm, xóa đi, hay thêm các tham số vào các phương thức, ...), ta có thể sử dụng lại file GWSDL và các lớp stub. Sử dụng cơ chế nhà cung cấp thay cho cách kế thừa chỉ là vấn đề thuộc khía cạnh triển khai, do đó không ảnh hưởng đến giao diện.

Toàn bộ mã nguồn của file này như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions                                    name="MathProviderService"
targetNamespace="http://MathProvider/MathProviderService"
xmlns:tns="http://MathProvider/MathProviderService"
xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExt
ensions"
xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">


<import                                    location="../../ogsi/ogsi.gwsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>


<types>
<xsd:schema
targetNamespace="http://MathProvider/MathProviderService"
attributeFormDefault="qualified"        elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema">


      <!--BEGIN ELEMENT DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->
      <xsd:element name="getValue">
            <xsd:complexType/>
      </xsd:element>
      <xsd:element name="getValueResponse">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element name="value" type="xsd:int"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="subtract">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element name="arg1" type="xsd:int"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="subtractResponse">
            <xsd:complexType/>
      </xsd:element>
```

```
      <xsd:element name="add">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element name="arg1" type="xsd:int"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="addResponse">
            <xsd:complexType/>
      </xsd:element>
      <!--END ELEMENT DEFINITIONS -->
</xsd:schema>
</types>

<!--BEGIN MESSAGE DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->

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

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

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

<message name="subtractOutputMessage">
      <part name="parameters" element="tns:subtractResponse"/>
</message>
```

```
<message name="addInputMessage">
      <part name="parameters" element="tns:add"/>
</message>

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

<!--END MESSAGE DEFINITIONS -->

<gwsdl:portType                              name="MathProviderPortType"
extends="ogsi:GridService">
      <!--BEGIN OPERATION DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -
->
      <operation name="getValue">
            <input message="tns:getValueInputMessage"/>
            <output message="tns:getValueOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <operation name="subtract">
            <input message="tns:subtractInputMessage"/>
            <output message="tns:subtractOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <operation name="add">
            <input message="tns:addInputMessage"/>
            <output message="tns:addOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <!--END OPERATION DEFINITIONS -->
</gwsdl:portType>
```

```
</definitions>
```

- Viết file thực thi cho dịch vụ .JAVA:

Lớp cung cấp chức năng (MathProvider) rất giống lớp MathImpl. Điểm khác nhau lớn nhất là trong lớp này ta cần viết mã thực thi giao diện OperationProvider.

Toàn bộ mã nguồn của file này như sau:

```
package MathProvider.impl;

import org.globus.ogsa.GridServiceBase;
import org.globus.ogsa.GridServiceException;
import org.globus.ogsa.OperationProvider;
import org.globus.ogsa.GridContext;

import java.rmi.RemoteException;
import javax.xml.namespace.QName;

public class MathProviderProvider implements OperationProvider
{

    // Operation provider properties
    private static final String
namespace="http://MathProvider/MathProviderService";
    private static final QName[] operations = new QName[]{
        new QName(namespace, "add"),
        new QName(namespace, "subtract"),
        new QName(namespace, "getValue")
        };
    private GridServiceBase base;

    // Operation provider methods
    public void initialize(GridServiceBase base) throws
GridServiceException
    {
```

```
            this.base = base;
        }

        public QName[] getOperations()
        {
            return operations;
        }

        // Insert implementation here
        private int value = 0;
        public void add(int a) throws RemoteException
        {
        value = value + a;
        }
        public void subtract(int a) throws RemoteException
        {
        value = value - a;
        }
        public int getValue() throws RemoteException
        {
        return value;
        }
}
```

Ta sẽ phân tích đoạn mã này.

Như đã đề cập ở trên, lớp này thực thi giao diện OperationProvider, không kế thừa từ bất kì lớp cơ sở nào:

```
public class MathProvider implements OperationProvider
```

Tiếp theo, ta cần viết mã cho hai phương thức trong giao diện OperationPovider: initialize() và getOperations(). Hai phương thức này cần ba thuộc tính riêng: namespace, operations và base.

```
// Operation provider properties
```

```
    private static final String namespace =
"http://www.globus.org/namespaces/2004/02/progtutorial/MathService";

    private static final QName[] operations =
        new QName[]
            {
            new QName(namespace, "add"),
            new QName(namespace, "subtract"),
            new QName(namespace, "getValue")
            };

    private GridServiceBase base;

    // Operation Provider methods
    public void initialize(GridServiceBase base) throws
GridServiceException
    {
        this.base = base;
    }

    public QName[] getOperations()
    {
        return operations;
    }
```

Hai phương thức trên và ba thuộc tính để làm cho nhà cung cấp thích hợp với dịch vụ lưới. Thuộc tính operations xác định các chức năng mà lớp này cung cấp. Trình chứa dịch vụ lưới dùng phương thức getOperations() để "hỏi" lớp này xem nó cung cấp những chức năng nào.

Danh sách các chức năng này là một mảng các tên đầy đủ. Điều này nghĩa là ta phải xác định cả namespace của chức năng và tên của nó. Thuộc tính namespace phải trùng với thuộc tính target namespace đã được xác định trong file GWSDL mà lớp cung cấp sẽ thực thi.

```
private static final String
namespace="http://MathProvider/MathProviderService";
    private static final QName[] operations =
```

```
            new QName[]
                  {
                  new QName(namespace, "add"),
                  new QName(namespace, "subtract"),
                  new QName(namespace, "getValue")
                  };
```

Ngoài các phương thức initilize() và getOperations(), các thuộc tính namespace, operations và base, thì phần mã còn lại là hoàn toàn giống như trường hợp trước (Factory).

```
private int value = 0;

      public void add(int a) throws RemoteException
      {
            value = value + a;
      }

      public void subtract(int a) throws RemoteException
      {
            value = value - a;
      }

      public int getValue() throws RemoteException
      {
            return value;
      }
```

- Viết file mô tả triển khai dịch vụ lưới

```
<?xml version="1.0"?>
<deployment                                    name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

      <service          name="MathProvider"          provider="Handler"
style="wrapped">
```

```
            <parameter name="name" value="MathProvider"/>
            <parameter                                    name="className"
value="MathProvider.stubs.MathProviderPortType"/>
            <parameter                                   name="schemaPath"
value="schema/gt3ide/MathProviderService/MathProvider_service.wsdl"/
>


            <parameter                                name="baseClassName"
value="org.globus.ogsa.impl.ogsi.GridServiceImpl"/>
            <parameter                           name="operationProviders"
value="MathProvider.impl.MathProviderProvider"/>


            <parameter name="allowedMethods" value="*"/>
            <parameter name="persistent" value="true"/>
            <parameter                                 name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>
      </service>
</deployment>
```

Chú ý các dòng bổ sung:

```
            <parameter                                    name="className"
value="MathProvider.stubs.MathProviderPortType"/>
            <parameter                                   name="schemaPath"
value="schema/gt3ide/MathProviderService/MathProvider_service.wsdl"/
>


            <parameter                                name="baseClassName"
value="org.globus.ogsa.impl.ogsi.GridServiceImpl"/>
```

Tên của lớp cơ sở không còn là MathImpl, hay là bất kì lớp nào mà ta xây dựng nữa.

```
            <parameter                                name="baseClassName"
value="org.globus.ogsa.impl.ogsi.GridServiceImpl"/>
```

Chúng ta sử dụng tham số này để "báo" cho trình chứa dịch vụ lưới biết GridServiceImpl sẽ cung cấp các chức năng cơ bản cho dịch vụ lưới của chúng ta. Phần thực thi cho các phương thức trong Math portType sẽ được tìm thấy trong một nhà cung cấp, được xác định trong tham số operationProvider:

```
<parameter                                    name="operationProviders"
value="MathProvider.impl.MathProviderProvider"/>
```

Cuối cùng, ta cần chỉ ra portType cho dịch vụ lưới. Nhớ rằng, nhà cung cấp chức năng không có portType "có thể thực thi", cho nên trình chứa dịch vụ lưới không thể sử dụng các nhà cung cấp chức năng để định nghĩa rõ PortType như thế nào.

```
<parameter                                              name="className"
value="MathProvider.stubs.MathProviderPortType"/>
```

- Viết file client

Ta sử dụng lại file client của ví dụ trước do muốn dùng chung một giao diện dịch vụ. Client gắn với một giao diện cụ thể, không liên quan đến việc thực thi giao diện đó. Cho nên, khi mà ta chưa thay đổi giao diện dịch vụ, ta có thể sử dụng lại cả file GWSDL và client. Tất nhiên, ta phải cung cấp một GSH để client truy nhập.

Mã nguồn file client như sau:

```
package MathProvider.impl;

import MathProvider.stubs.service.MathProviderServiceGridLocator;
import MathProvider.stubs.MathProviderPortType;
import java.net.URL;

public class MathProviderClient {
    public static void main(String[] args)
    {
    try
    {
        //    Get command-line arguments
        URL GSH = new java.net.URL(args[0]);
        int a = Integer.parseInt(args[1]);
        //    Get a reference to the MathService instance
        MathProviderServiceGridLocator mathServiceLocator = new
MathProviderServiceGridLocator();
        MathProviderPortType                    math                    =
mathServiceLocator.getMathProviderServicePort(GSH);
        //    Call remote method 'add'
        math.add(a);
```

```
            System.out.println("Added " + a);
            //     Get current value through remote method 'getValue'
            int value = math.getValue();
            System.out.println("Current value: " + value);
        }
        catch(Exception e)
        {
            System.out.println("ERROR!");
            e.printStackTrace();
        }
}
}
```

2.3.2. Cơ chế dữ liệu dịch vụ

2.3.2.1. Dữ liệu dịch vụ

Dữ liệu dịch vụ là một trong những bước phát triển chính của dịch vụ lưới xét trên khía cạnh so với dịch vụ web.

- Tại sao cần có dữ liệu dịch vụ?

Đối với kiến trúc dịch vụ web, một trong những thành phần quan trọng nhất là khả năng tìm kiếm dịch vụ, cho phép thu được URI của dịch vụ có thể đáp ứng các yêu cầu của ta thông qua thành phần quản lý đăng ký UDDI. Các dịch vụ web quảng bá khả năng của nó cho bộ đăng kí thông qua ngôn ngữ mô tả WSDL.

Tuy nhiên, nhược điểm của WSDL là quá đi sau vào chi tiết mô tả cách triệu gọi từng phương thức, các giao thức, ... Ta cần một cái gì đó đơn giản hơn để phân lớp và lập chỉ mục. Dữ liệu dịch vụ là nhằm đáp ứng các yêu cầu này.

- Mặc dù có thể triển khai dữ liệu dịch vụ theo nhiều cách khác nhau như các dịch vụ web tiêu chuẩn, tuy nhiên trong khuôn khổ tài liệu ta chỉ nghiên cứu giải pháp của kiến trúc OGSI (dùng trong dịch vụ lưới).

- Dữ liệu dịch vụ là gì?

Dữ liệu dịch vụ là một tập có cấu trúc các thông tin liên quan đến một dịch vụ lưới. Các thông tin này phải đáp ứng yêu cầu dễ dàng trong việc truy vấn, để có thể dễ dàng phân lớp và lập chỉ mục các dịch vụ lưới dựa trên khả năng mà từng dịch vụ có.

Việc thực hiện các thuật toán phức tạp để truy vấn dữ liệu dịch vụ và chọn ra dịch vụ phù hợp không phải là nhiệm vụ của client, đó là công việc của dịch vụ chỉ mục. Dịch vụ chỉ mục sẽ chọn ra dịch vụ phù hợp dựa trên các thông số như giá thành, tốc độ, tính sẵn sàng, ... Tập các thông số để lựa chọn tùy thuộc vào ứng dụng cụ thể.

Dữ liệu dịch vụ là thành phần thiết yếu của quá trình tìm kiếm dịch vụ. Ngoài ra, nó còn có quan hệ chặt chẽ với cơ chế thông báo.

- Dữ liệu dịch vụ được triển khai như thế nào?

Mỗi dịch vụ lưới có một vài phần tử dữ liệu dịch vụ (SDE) gắn với nó. Mỗi SDE có một tên khác nhau, tên này là duy nhất đối với mỗi loại dịch vụ lưới cụ thể. Nó chứa dữ liệu có cấu trúc (được trình bày dưới khuôn dạng XML), cấu trúc này đã được định nghĩa trong file mô tả dịch vụ lưới GWSDL.

- Ví dụ 1 về dữ liệu dịch vụ:

Dịch vụ MathService có 2 SDE là SystemInfo và LastResults, trong đó SDE SystemInfo chứa các thông tin về hệ thống nơi mà dịch vụ MathService chạy (số CPU, tải của hệ thống, tải của CPU, ...); còn SDE LastResult lưu thông tin về kết quả của phép toán.

Mỗi SDE có thể có nhiều giá trị, chẳng hạn LastResult có thể chứa 10 kết quả cuối cùng của phép toán.

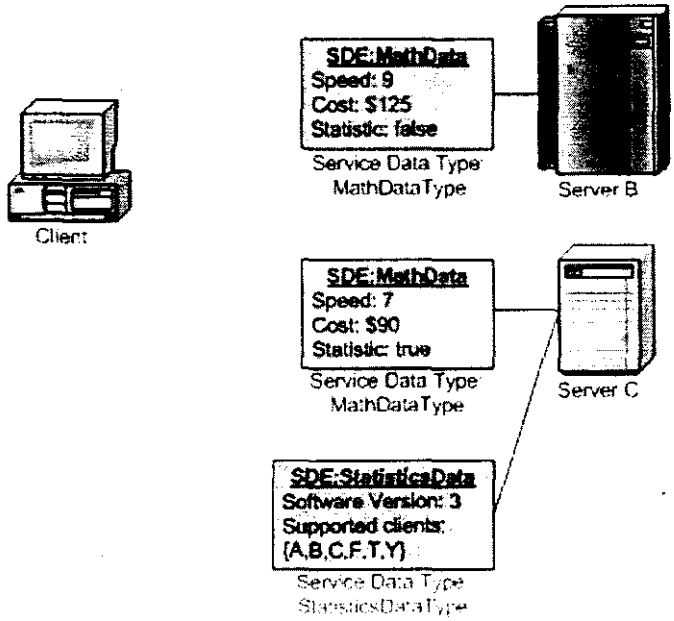
Chú ý rằng dữ liệu dịch vụ có thể được chia thành 2 loại :

+ Thông tin trạng thái : kết quả phép toán, các kết quả trung gian, thông tin về quá trình thực hiện, ...

+ Siêu dữ liệu dịch vụ: dữ liệu hệ thống, các giao diện mà nó cung cấp, giá thực hiện dịch vụ, ...

SDE SystemInfo thuộc loại siêu dữ liệu dịch vụ, còn SDE LastResult thuộc loại thông tin trạng thái.

- Ví dụ 2 về dữ liệu dịch vụ:



Trong hình vẽ trên, ta có 3 dịch vụ MathService. Hai dịch vụ đầu, mỗi dịch vụ có một SDE là MathData, cả hai SDE này có cùng kiểu thông tin : tốc độ, giá và thống kê. Dịch vụ thứ 3 có hai SDE : MathData và StatisticsData. Chú ý rằng SDE StatisticsData chứa các thông tin khác kiểu với MathData : phiên bản phần mềm và các client được chấp nhận.

Client khi có nhu cầu thực hiện dịch vụ sẽ hỏi từng MathService về SDE MathData, sau đó sẽ so sánh và chọn ra dịch vụ phù hợp. Nếu client quan tâm đến các thông tin về thống kê thì nó sẽ họn dịch vụ thứ 3 (vì chỉ có nó là hỗ trợ việc cung cấp các thông tin về thống kê).

Chú ý rằng việc xác định xem dịch vụ nào là tốt hơn không phải là của client, nó được dịch vụ chỉ mục thực hiện.

- Dữ liệu dịch vụ được triển khai như thế nào (con't)?

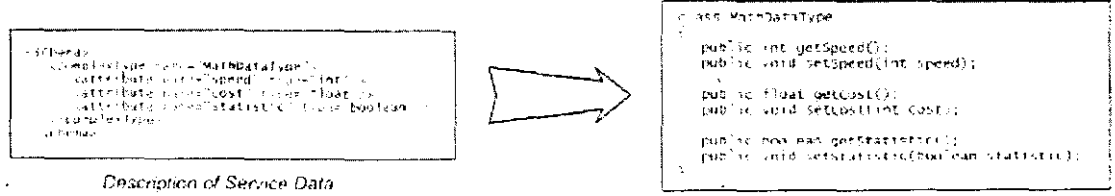
Vấn đề đặt ra là làm thế nào để triển khai dữ liệu dịch vụ. Chẳng hạn để thêm SDE MathData, ta có thể thêm các thuộc tính tốc độ, giá và thống kê vào mã thực thi của dịch vụ MathService và truy vấn chúng bằng các phương thức get/set. Tuy nhiên, đây không phải là cách làm hay để giải quyết điều này. Một vấn đề gặp phải khi sử dụng cách làm này là chúng ta phải giải quyết như thế nào trong trường hợp có nhiều SDE hay là không có SDE nào cả.

Giải pháp được đề xuất là dùng file mô tả dịch vụ lưới GWSDL. Một trong những mở rộng của GWSDL so với WSDL là ta có thể gắn SDE vào portType, xác định số giá trị của mỗi SDE với các thuộc tính khác nhau. Loại dữ liệu của mỗi SDE được thể hiện dưới cú pháp XML.

Chẳng hạn với SDE LastResult, kiểu dữ liệu là nguyên (cơ bản), nên chỉ cần chỉ ra trong file GWSDL kiểu dữ liệu là <xsd: int>

Với SDE SystemInfo, kiểu dữ liệu phức tạp hơn : nó được thể hiện dưới dạng cấu trúc với một tập các thuộc tính (trường). Ta phải dùng phần <types> của lược đồ XML trong file GWSDL để định nghĩa trước kiểu này trước khi dùng. Phần định nhĩa này cũng có thể lưu trong 1 file lược đồ XML riêng (.xsd), sau đó nó sẽ được import vào file GWSDL.

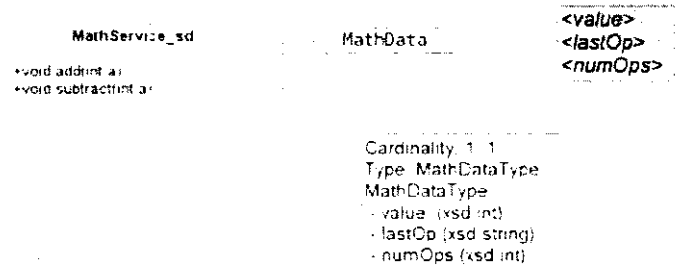
Việc truy vấn SDE là rất dễ dàng. Nếu SDE có một kiểu dữ liệu cơ sở (nguyên, thực, xâu,...), ta có thể truy nhập trực tiếp. Nếu SDE có nhiều giá trị, kết quả trả về sẽ là một mảng. Còn nếu SDE có kiểu dữ liệu phức, ta có thể dùng các 'SDE Java Bean' để truy nhập từng thành phần trong kiểu dữ liệu phức này. SDE Java Bean sẽ được tự sinh khi bạn biên dịch.



Description of Service Data Element in XML Schema

Java Bean

2.3.2.2. Ví dụ xây dựng dữ liệu dịch vụ cho MathService



Dịch vụ MathService_sd có 1 SDE:

+ SDE này chỉ có 1 giá trị:

Cardinality: 1 .. 1

+ Kiểu dữ liệu là phức, được gọi là MathDataType, bao gồm các thông tin :

. Kết quả phép toán

. Tên phép toán thực hiện cuối cùng (cộng hay trừ)

. Số lần thực hiện tính toán (số thao tác)

Ta có thể định nghĩa kiểu MathDataType trong file GWSDL, tuy nhiên trong trường hợp này ta dùng 1 file riêng gọi là MathSDE.xsd, toàn bộ mã nguồn của file này như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="MathData"
     targetNamespace="http://MathService_sd/MathService_sdService/MathSDE"
     xmlns:tns="http://MathService_sd/MathService_sdService/MathSDE"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

<wsdl:types>
<schema
targetNamespace="http://MathService_sd/MathService_sdService/MathSDE"
     attributeFormDefault="qualified"
     elementFormDefault="qualified"
     xmlns="http://www.w3.org/2001/XMLSchema">

     <complexType name="MathDataType">
          <sequence>
```

```
            <element name="value" type="int"/>
            <element name="lastOp" type="string"/>
            <element name="numOps" type="int"/>
        </sequence>
    </complexType>

</schema>
</wsdl:types>

</wsdl:definitions>
```

File này được lưu trong .../MathService_sd/schema/gt3ide/MathService_sdService. Đây là 1 file XML bình thường. Có một mô tả GWSDL với 1 thẻ <types> đơn giản không có tham số, portType hay ràng buộc. Thẻ <types> này có duy nhất 1 kiểu là MathDataType.

```
<wsdl:types>
<schema
targetNamespace="http://www.globus.org/namespaces/2004/02/progtutori
al/MathService_sd/MathSDE"
    attributeFormDefault="qualified"
    elementFormDefault="qualified"
    xmlns="http://www.w3.org/2001/XMLSchema">

    <complexType name="MathDataType">
        <sequence>
            <element name="value" type="int"/>
            <element name="lastOp" type="string"/>
            <element name="numOps" type="int"/>
        </sequence>
    </complexType>

</schema>
</wsdl:types>
```

Nếu bạn muốn tạo một kiểu SDD riêng, bạn cần viết 1 file lược đồ XML mới và thay đổi thuộc tính targetNamespace để tương ứng với nó.

Lớp Java Bean được ant sinh ra khi biên dịch tương ứng có dạng như sau :

```
public class MathDataType implements java.io.Serializable {
private int value; // attribute
private java.lang.String lastOp; // attribute
private int numOps; // attribute
public MathDataType() {
}
public int getValue() {
return value;
}
public void setValue(int value) {
this.value = value;
}
public java.lang.String getLastOp() {
return lastOp;
}
public void setLastOp(java.lang.String lastOp) {
this.lastOp = lastOp;
}
public int getNumOps() {
return numOps;
}
public void setNumOps(int numOps) {
this.numOps = numOps;
}
}
```

Chú ý là file này được tự sinh, bạn không phải viết.

+ Giao diện dịch vụ :

Thêm dữ liệu dịch vụ liên quan đến khía cạnh giao diện, nên ta phải viết file GWSDL mới (khác với cơ chế nhà cung cấp ở trên). Tuy nhiên, file này về cơ bản là giống với file GWSDL trong ví dụ cũ. Toàn bộ mã nguồn của nó như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<definitions                                    name="MathService_sdService"
targetNamespace="http://MathService_sd/MathService_sdService"
xmlns:tns="http://MathService_sd/MathService_sdService"
xmlns:data="http://MathService_sd/MathService_sdService/MathSDE"
xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExt
ensions"
xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">

<import                                    location="../../ogsi/ogsi.gwsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
<import                                                location="MathSDE.xsd"
namespace="http://MathService_sd/MathService_sdService/MathSDE"/>

<types>
<xsd:schema
targetNamespace="http://MathService_sd/MathService_sdService"
attributeFormDefault="qualified"        elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema">

    <!--BEGIN ELEMENT DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->
    <xsd:element name="subtract">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="arg1" type="xsd:int"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="subtractResponse">
        <xsd:complexType/>
    </xsd:element>
    <xsd:element name="add">
        <xsd:complexType>
```

```
                <xsd:sequence>
                    <xsd:element name="arg1" type="xsd:int"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="addResponse">
            <xsd:complexType/>
        </xsd:element>
        <!--END ELEMENT DEFINITIONS -->
</xsd:schema>
</types>

<!--BEGIN MESSAGE DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->

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

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

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

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

<!--END MESSAGE DEFINITIONS -->
```

```
<gwsdl:portType                                    name="MathService_sdPortType"
extends="ogsi:GridService">
      <!--BEGIN OPERATION DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -
->
      <operation name="subtract">
            <input message="tns:subtractInputMessage"/>
            <output message="tns:subtractOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <operation name="add">
            <input message="tns:addInputMessage"/>
            <output message="tns:addOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <!--END OPERATION DEFINITIONS -->
        <sd:serviceData    name="MathData"    type="data:MathDataType"
minOccurs="1" maxOccurs="1" mutability="mutable" modifiable="false"
nillable="false">
  </sd:serviceData>
</gwsdl:portType>
</definitions>
```

Chú ý các thay đổi sau:

Đầu tiên là thay đổi targetNamespace: MathService_sd

Thêm vào hai namespace mới: một ứng với file .xsd (tương ứng targetNamespace trong file .xsd) và một namespace OGSI cho tập các định nghĩa liên quan đến dữ liệu dịch vụ.

```
<definitions                                       name="MathService_sdService"
targetNamespace="http://MathService_sd/MathService_sdService"
xmlns:tns="http://MathService_sd/MathService_sdService"
xmlns:data="http://MathService_sd/MathService_sdService/MathSDE"
xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExt
ensions"
xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Sau đó, import file mô tả kiểu dữ liệu MathDataType. Chú ý namespace cũng phải tương ứng với targetNamespace trong file .xsd

```
<import location="../../ogsi/ogsi.gwsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>

<import location="MathSDE.xsd"
namespace="http://MathService_sd/MathService_sdService/MathSDE"/>
```

Cuối cùng, thêm một thẻ mới (từ không gian tên của dữ liệu dịch vụ) trong thẻ <gwsdl:portType>, đó là <sd :serviceData> cho phép xác định các thuộc tính của mỗi SDE trong dịch vụ lưới.

```
<gwsdl:portType name="MathService_sdPortType"
extends="ogsi:GridService">

< !-- operations --!>

...

  <sd:serviceData name="MathData"
                  type="data:MathDataType"
                  minOccurs="1"
                  maxOccurs="1"
                  mutability="mutable"
                  modifiable="false"
                  nillable="false">

  </sd:serviceData>

</gwsdl:portType>
```

Thuộc tính đầu tiên (name) là tên của SDE, phải là duy nhất trong dịch vụ lưới đang xét. Thuộc tính tiếp theo (type) chỉ kiểu của SDE này, kiểu này được xác định trong file .xsd. Chú ý cách dùng tương ứng giữa data namespace và targetNamespace trong file .xsd.

Các thuộc tính tiếp theo chỉ các tính chất của SDE

. minOccurs : số giá trị nhỏ nhất của SDE

. maxOccurs : số giá trị lớn nhất của SDE. Giá trị của thuộc tính này có thể không bị chặn. Lúc đó ta sẽ có kết quản trả về là một mảng không kích thước.

. modifiable : true hoặc false, qui định client có thể thay đổi số giá trị của SDE không

. nillable : true hoặc false, qui định số giá trị của SDE có thể là NULL không

. mutability : thuộc tính này có thể nhận các giá trị sau

_ static : giá trị của SDE được cung cấp bởi mô tả GWSDL

_ constant : giá trị của SDE được thiết lập khi dịch vụ lưới được tạo ra, và không thay đổi giá trị từ đó về sau.

_ extendable : có thể thêm các phần tử mới vào SDE, nhưng không được xóa đi

_ mutable : các phần tử mới có thể được thêm vào hoặc xóa đi.

Trong ví dụ của chúng ta, SDE MathData chỉ có duy nhất 1 giá trị (cardinality : 1 .. 1), có thể thay đổi trong quá trình hoạt động của dịch vụ lưới, nhưng không thể là NULL và client không có quyền thay đổi SDE này.

Cuối cùng, do các giá trị tính toán đã được thể hiện trong SDE, nên phương thức getValue() là không còn cần thiết, ta có thể truy nhập giá trị thông qua SDE. Chú ý rằng getValue cũng được loại bỏ trong file GWSDL.

+ Ánh xạ không gian tên:

Vì ta đã viết lại một giao diện mới, với một không gian tên mới, nên ta cần ánh xạ lại không gian tên này vào gói Java để stubs có thể được sinh ra một cách đúng đắn. Ta cũng thêm ánh xạ cho MathDataType targetNamespace. Toàn bộ nội dung của file .../MathService_sd/namespace2package.mappings như sau:

```
http\://MathService_sd/MathService_sdService=MathService_sd.stubs

http\://MathService_sd/MathService_sdService/bindings=MathService_sd
.stubs.bindings

http\://MathService_sd/MathService_sdService/service=MathService_sd.
stubs.service

http\://MathService_sd/MathService_sdService/MathSDE=MathService_sd.
stubs.servicedata
```

+ Viết mã thực thi dịch vụ :

```
package MathService_sd.impl;


import org.globus.ogsa.ServiceData;

import org.globus.ogsa.impl.ogsi.GridServiceImpl;

import org.globus.ogsa.GridContext;

import org.globus.ogsa.GridServiceException;


import MathService_sd.stubs.MathService_sdPortType;

import MathService_sd.stubs.servicedata.MathDataType;

import java.rmi.RemoteException;
```

```
public class MathService_sdImpl extends GridServiceImpl implements
MathService_sdPortType
{
      private ServiceData mathDataSDE;
      private MathDataType mathDataValue;

      public MathService_sdImpl()
      {
            super("Simple MathService with Service Data");
      }

      public void postCreate(GridContext context) throws
GridServiceException
      {
            // Call base class's postCreate
            super.postCreate(context);

            // Create Service Data Element
            mathDataSDE =
this.getServiceDataSet().create("MathData");

            // Create a MathDataType instance and set intial values
            mathDataValue = new MathDataType();
            mathDataValue.setLastOp("NONE");
            mathDataValue.setNumOps(0);
            mathDataValue.setValue(0);

            // Set the value of the SDE to the MathDataType instance
            mathDataSDE.setValue(mathDataValue);

            // Add SDE to Service Data Set
```

```
            this.getServiceDataSet().add(mathDataSDE);
        }


        // This method updates the MathData SDE increasing the
        // number of operations by one
        private void incrementOps()
        {
            int numOps = mathDataValue.getNumOps();
            mathDataValue.setNumOps(numOps + 1);
        }

        public void add(int a) throws RemoteException
        {
            mathDataValue.setLastOp("Addition");
            incrementOps();
            mathDataValue.setValue(mathDataValue.getValue() + a);
        }

        public void subtract(int a) throws RemoteException
        {
            mathDataValue.setLastOp("Subtraction");
            incrementOps();
            mathDataValue.setValue(mathDataValue.getValue() - a);
        }
}
```

Ta cần thay đổi rất nhiều trong file thực thi này so với các ví dụ trước.

Tuy nhiên khai báo lớp vẫn như cũ

```
public class MathService_sdImpl extends GridServiceImpl implements
MathService_sdPortType
```

Lớp có hai thuộc tính riêng mới, một là SDE (kiểu ServiceData) và một là giá trị của SDE (kiểu MathDataType)

```
    private ServiceData mathDataSDE;
    private MathDataType mathDataValue;
```

SDE được khởi tạo nhờ phương thức postCreate() - một phương thức đặc biệt, được thực hiện theo cơ chế gọi ngược (callback) - nó sẽ được thực thi ngay sau khi dịch vụ được khởi tạo. Đây là nơi ta thiết lập các giá trị khởi đầu cho SDE.

```
    public void postCreate(GridContext context) throws
GridServiceException
    {
        // Call base class's postCreate
        super.postCreate(context);

        // Create Service Data Element
        mathDataSDE =
this.getServiceDataSet().create("MathData");

        // Create a MathDataType instance and set intial values
        mathDataValue = new MathDataType();
        mathDataValue.setLastOp("NONE");
        mathDataValue.setNumOps(0);
        mathDataValue.setValue(0);

        // Set the value of the SDE to the MathDataType instance
        mathDataSDE.setValue(mathDataValue);

        // Add SDE to Service Data Set
        this.getServiceDataSet().add(mathDataSDE);
    }
```

Các bước phải thực hiện để khởi tạo một SDE và thêm nó vào tập dữ liệu dịch vụ là :

1. Tạo một SDE mới. Chú ý rằng ta không khởi tạo trực tiếp mà thông qua phương thức khởi tạo của tập dữ liệu dịch vụ. SDE này sẽ được khởi tạo với giá trị rỗng, không có giá trị. Tên của SDE là MathData

2. Thiết lập giá trị cho SDE. Giá trị của SDE sẽ là một giá trị thuộc kiểu MathDataType mà ta tạo ra.

3. Thiết lập giá trị khởi đầu cho MathDataType. Trong ví dụ của chúng ta, phép toán cuối cùng là NONE và số phép toán được thực hiện là 0.

4. Thêm SDE vào tập các dữ liệu dịch vụ.

Các phương thức add và subtracted thay đổi dữ liệu dịch vụ (giá trị, toán tử thực hiện cuối cùng và số phép toán được thực hiện) mỗi khi nó được gọi. Chúng ta thực hiện điều này qua thuộc tính riêng mathDataValue và phương thức bổ sung incrementOps()

```
public void add(int a) throws RemoteException
{
      mathDataValue.setLastOp("Addition");
      incrementOps();
      mathDataValue.setValue(mathDataValue.getValue() + a);
}
```

Phương thức riêng incrementOps() chỉ đơn giản tăng số phép toán được thực hiện lên 1.

```
// This method updates the MathData SDE increasing the
// number of operations by one
private void incrementOps()
{
      int numOps = mathDataValue.getNumOps();
      mathDataValue.setNumOps(numOps + 1);
}
```

+ File mô tả triển khai

File GWSDL thay đổi nhiều : lớp mới, lớp cơ sở mới, đường dẫn cho lược đồ mới (tương ứng với các file ta vừa viết). Toàn bộ nội dung của file này như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions                                    name="MathService_sdService"
targetNamespace="http://MathService_sd/MathService_sdService"
xmlns:tns="http://MathService_sd/MathService_sdService"
xmlns:data="http://MathService_sd/MathService_sdService/MathSDE"
xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExt
ensions"
xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">
```

```
<import                                    location="../../ogsi/ogsi.gwsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
<import                                                location="MathSDE.xsd"
namespace="http://MathService_sd/MathService_sdService/MathSDE"/>

<types>
<xsd:schema
targetNamespace="http://MathService_sd/MathService_sdService"
attributeFormDefault="qualified"       elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema">

      <!--BEGIN ELEMENT DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->
      <xsd:element name="subtract">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element name="arg1" type="xsd:int"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="subtractResponse">
            <xsd:complexType/>
      </xsd:element>
      <xsd:element name="add">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element name="arg1" type="xsd:int"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="addResponse">
            <xsd:complexType/>
      </xsd:element>
      <!--END ELEMENT DEFINITIONS -->
```

```
</xsd:schema>
</types>

<!--BEGIN MESSAGE DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->

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

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

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

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

<!--END MESSAGE DEFINITIONS -->

<gwsdl:portType                          name="MathService_sdPortType"
extends="ogsi:GridService">
      <!--BEGIN OPERATION DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -
->
      <operation name="subtract">
            <input message="tns:subtractInputMessage"/>
            <output message="tns:subtractOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
```

```
    <operation name="add">
        <input message="tns:addInputMessage"/>
        <output message="tns:addOutputMessage"/>
        <fault name="Fault" message="ogsi:FaultMessage"/>
    </operation>
    <!--END OPERATION DEFINITIONS -->
      <sd:serviceData name="MathData" type="data:MathDataType"
minOccurs="1" maxOccurs="1" mutability="mutable" modifiable="false"
nillable="false">
  </sd:serviceData>
</gwsdl:portType>
</definitions>
```

+ Viết file client

Ta sẽ thay đổi file client sao cho nó có thể truy nhập được SDE MathData. Thay vì chỉ cộng hai số như trước đây, nó còn phải hỏi MathService và SDE MathData. Trước khi thực hiện phép cộng, nó sẽ hiển thị nội dung của SDE (giá trị hiện thời, phép toán đã thực hiện trước đó, và số phép toán đã thực hiện). Toàn bộ mã nguồn của nó như sau:

```
package MathService_sd.impl;

import org.globus.ogsa.ServiceData;
import org.globus.ogsa.impl.ogsi.GridServiceImpl;
import org.globus.ogsa.GridContext;
import org.globus.ogsa.GridServiceException;

import MathService_sd.stubs.MathService_sdPortType;
import MathService_sd.stubs.servicedata.MathDataType;
import java.rmi.RemoteException;

public class MathService_sdImpl extends GridServiceImpl implements
MathService_sdPortType
{
    private ServiceData mathDataSDE;
    private MathDataType mathDataValue;
```

```
    public MathService_sdImpl()
    {
        super("Simple MathService with Service Data");
    }

    public void postCreate(GridContext context) throws
GridServiceException
    {
        // Call base class's postCreate
        super.postCreate(context);

        // Create Service Data Element
        mathDataSDE =
this.getServiceDataSet().create("MathData");

        // Create a MathDataType instance and set intial values
        mathDataValue = new MathDataType();
        mathDataValue.setLastOp("NONE");
        mathDataValue.setNumOps(0);
        mathDataValue.setValue(0);

        // Set the value of the SDE to the MathDataType instance
        mathDataSDE.setValue(mathDataValue);

        // Add SDE to Service Data Set
        this.getServiceDataSet().add(mathDataSDE);
    }

    // This method updates the MathData SDE increasing the
    // number of operations by one
```

```
    private void incrementOps()
    {
        int numOps = mathDataValue.getNumOps();
        mathDataValue.setNumOps(numOps + 1);
    }

    public void add(int a) throws RemoteException
    {
        mathDataValue.setLastOp("Addition");
        incrementOps();
        mathDataValue.setValue(mathDataValue.getValue() + a);
    }

    public void subtract(int a) throws RemoteException
    {
        mathDataValue.setLastOp("Subtraction");
        incrementOps();
        mathDataValue.setValue(mathDataValue.getValue() - a);
    }
}
```

Đầu tiên, ta cần lấy SDE tên là MathData, ta sử dụng một phương thức của dịch vụ lưới tên là findServiceData, và một lớp 'helper' để chuyển tên thành cái mà dịch vụ lưới có thể hiểu được.

```
ExtensibilityType extensibility =
math.findServiceData(QueryHelper.getNamesQuery("MathData"));
```

Chú ý là ta triệu gọi phương thức findServiceData() trực tiếp từ math - là một MathPortType. Do ngoài hai phương thức là add() và subtract() mà ta viết, MathPortType còn kết thừa từ portTye cơ sở là GridService. Cho nên ta có thể sử dụng math để triệu gọi các phương thức của dịch vụ lưới.

Một điểm cần nhớ nữa là phương thức findServiceData không trả về một lớp kiểu MathDataType, mà là lớp ExtensibilityType. Ta cần chuyển nó sang MathDataType sử dụng lớp 'helper'

```
ServiceDataValuesType serviceData =
AnyHelper.getAsServiceDataValues(extensibility);
```

```
                MathDataType mathData = (MathDataType)
AnyHelper.getAsSingleObject(serviceData, MathDataType.class);
```

Bây giờ ta đã có đối tượng MathDataType, ta có thể dùng nó như một đối tượng cục bộ

```
                System.out.println("Value: " + mathData.getValue());
                System.out.println("Previous operation: " +
mathData.getLastOp());
                System.out.println("# of operations: " +
mathData.getNumOps());
```

- Dữ liệu dịch vụ của một dịch vụ lưới

Ngoài các dữ liệu dịch vụ mà ta phải thêm vào (như MathData), tất cả các dịch vụ lưới đều có một tập các SDE chung mô tả các thuộc tính phải có của một dịch vụ lưới, chẳng hạn GSH của một thể hiện. Các SDE này là một phần của GridService portType (nhớ rằng : nếu muốn một portType là một dịch vụ lưới, nó phải kế thừa từ GridService portType).

Một số SDE tiêu biểu là :

. gridServiceHandle : SDE đa trị, trả về nhiều GSH của dịch vụ lưới

. factoryLocator : SDE đơn trị, trả về factory đã tạo ra dịch vụ lưới. Nếu một dịch vụ lưới không được tạo ra từ factory nào cả, giá trị của SDE này sẽ là null.

. terminationTime : SDE đơn trị, trả về thông tin về thời gian kết thúc của dịch vụ lưới.

. serviceDataNames : SDE đa trị, trả về tên tất cả các SDE của dịch vụ lưới.

. interfaces : SDE đa trị, trả về tên tất cả các giao diện (portType) được thực thi bởi dịch vụ lưới.

GrridService PortType có thêm một số SDE khác, chẳng hạn gridServiceReference, findServiceDataExtensibility, setServiceDataExtensibility, ...

- Viết client in ra các dữ liệu dịch vụ của một dịch vụ lưới

Tham số đầu vào là GSH của dịch vụ.

```
package MathService_sd.impl;

import org.gridforum.ogsi.OGSIServiceGridLocator;
import org.gridforum.ogsi.GridService;
import org.gridforum.ogsi.ExtensibilityType;
import org.gridforum.ogsi.HandleType;
import org.gridforum.ogsi.ServiceDataValuesType;
import org.gridforum.ogsi.LocatorType;
import org.gridforum.ogsi.TerminationTimeType;
```

```
import org.gridforum.ogsi.ExtendedDateTimeType;
import org.globus.ogsa.utils.AnyHelper;
import org.globus.ogsa.utils.QueryHelper;
import java.net.URL;
import javax.xml.namespace.QName;

public class PrintGridServiceData
{
    public static void main(String[] args)
    {
        try
        {
            // Get command-line arguments
            URL GSH = new java.net.URL(args[0]);

            // Get a reference to the GridService portType
            OGSIServiceGridLocator locator = new
OGSIServiceGridLocator();
            GridService gridService =
locator.getGridServicePort(GSH);

            // gridServiceHandle
            // GSH of this instance (can have more than one)
            ExtensibilityType extensibility =
gridService.findServiceData(QueryHelper.getNamesQuery("gridServiceHa
ndle"));
            ServiceDataValuesType serviceData =
AnyHelper.getAsServiceDataValues(extensibility);
            Object[] gridServiceHandles =
AnyHelper.getAsObject(serviceData);
            for(int i=0; i<gridServiceHandles.length; i++)
            {
```

```
                    HandleType gridServiceHandle = (HandleType)
gridServiceHandles[i];
                    System.out.println("gridServiceHandle:    "    +
gridServiceHandle);
               }



               // factoryLocator
               //  Locator  for  the  factory  that  created  this
instance
               extensibility                                       =
gridService.findServiceData(QueryHelper.getNamesQuery("factoryLocato
r"));
               serviceData                                         =
AnyHelper.getAsServiceDataValues(extensibility);
               if (serviceData.get_any() == null)
                    System.out.println("factoryLocator:          Not
created by a factory!");
               else
               {
                    Object[]            factoryLocators            =
AnyHelper.getAsObject(serviceData);
                    for(int i=0; i<factoryLocators.length; i++)
                    {
                         LocatorType        factoryLocator         =
(LocatorType) factoryLocators[i];
                         System.out.println("factoryLocator:   "   +
factoryLocator.getHandle(0));
                    }
               }



               // terminationTime
               // The termination time for this service
```

```
                extensibility =
gridService.findServiceData(QueryHelper.getNamesQuery("terminationTi
me"));
                serviceData =
AnyHelper.getAsServiceDataValues(extensibility);
                TerminationTimeType terminationTime =
(TerminationTimeType) AnyHelper.getAsSingleObject(serviceData);
                ExtendedDateTimeType after, before;
                after = terminationTime.getAfter();
                before = terminationTime.getBefore();

                if
(after.getInfinityTypeValue().getValue().equals("infinity"))
                    System.out.println("terminationTime (after):
The service plans to exist indefinitely");
                else if
(after.getDateTimeValue().before(terminationTime.getTimestamp().getT
ime()))
                    System.out.println("terminationTime (after):
May terminate at any time.");
                else
                    System.out.println("terminationTime (after): "
+ after.getDateTimeValue().getTime());

                if
(before.getInfinityTypeValue().getValue().equals("infinity"))
                    System.out.println("terminationTime (before):
The service has no plans to terminate.");
                else if
(before.getDateTimeValue().before(terminationTime.getTimestamp().get
Time()))
                    System.out.println("terminationTime (before):
The service is trying to terminate.");
                else
                    System.out.println("terminationTime (before):
" + before.getDateTimeValue().getTime());
```

```
                System.out.println("terminationTime (timestamp): " +
terminationTime.getTimestamp().getTime());




                // serviceDataNames
                // Names of Service Data offered by this Grid
Service
                extensibility =
gridService.findServiceData(QueryHelper.getNamesQuery("serviceDataNa
me"));
                serviceData =
AnyHelper.getAsServiceDataValues(extensibility);
                Object[] serviceDataNames =
AnyHelper.getAsObject(serviceData);
                for(int i=0; i<serviceDataNames.length; i++)
                {
                     QName serviceDataName = (QName)
serviceDataNames[i];
                     System.out.println("serviceDataName: " +
serviceDataName);
                }



                // interfaces
                // Names of interfaces exposed by this Grid Service
                extensibility =
gridService.findServiceData(QueryHelper.getNamesQuery("interface"));
                serviceData =
AnyHelper.getAsServiceDataValues(extensibility);
                Object[] interfaces =
AnyHelper.getAsObject(serviceData);
                for(int i=0; i<interfaces.length; i++)
```

```
                {
                    QName iface = (QName) interfaces[i];
                    System.out.println("interface: " + iface);
                }

            }catch(Exception e)
            {
                System.out.println("ERROR!");
                e.printStackTrace();
            }
        }
}
```

Chú ý cuối cùng là dịch vụ này không được viết theo mô hình factory, hãy để ý cách mà nó thực thi GridService portType và MathPortType.

2.3.3. Cơ chế thông báo

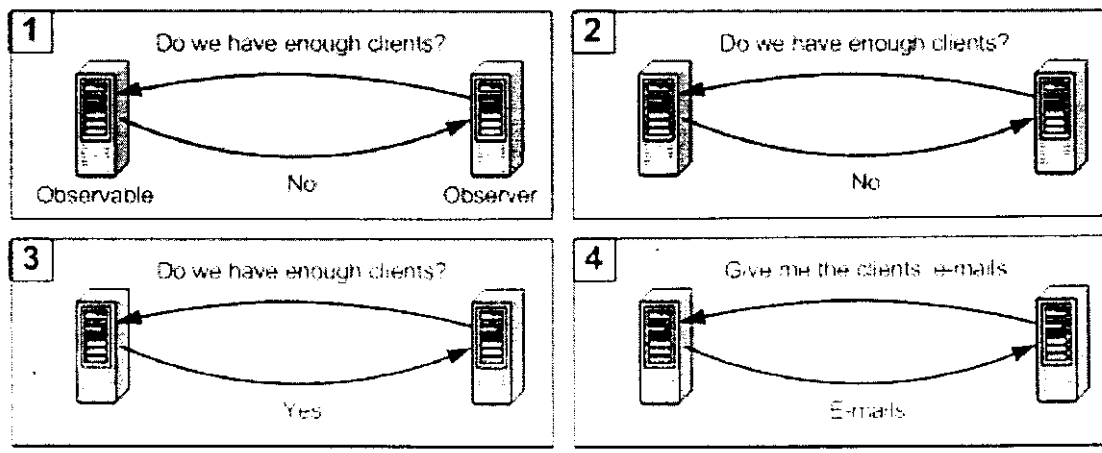
2.3.3.1. Cơ chế thông báo là gì?

+ Cơ chế thông báo liên quan chặt chẽ đến dữ liệu dịch vụ, nó cho phép client nhận được thông báo về các thay đổi phía server (dịch vụ lưới).

Đây không phải là một kỹ thuật mới trong công nghệ phần mềm. Thậm chí nó còn là một kỹ thuật hết sức thông dụng trong các khuôn mẫu thiết kế với các tên gọi khác nhau như: Observer/Observable, Model-View-Controller, ...

+ Cơ chế thông báo được thực hiện như thế nào?

Một kỹ thuật đơn giản để client biết các thay đổi trên server là dùng phương pháp thăm dò (polling). Cứ định kỳ sau một khoảng thời gian, client (observer) sẽ gửi yêu cầu đến server (observable) hỏi xem có thay đổi gì không.

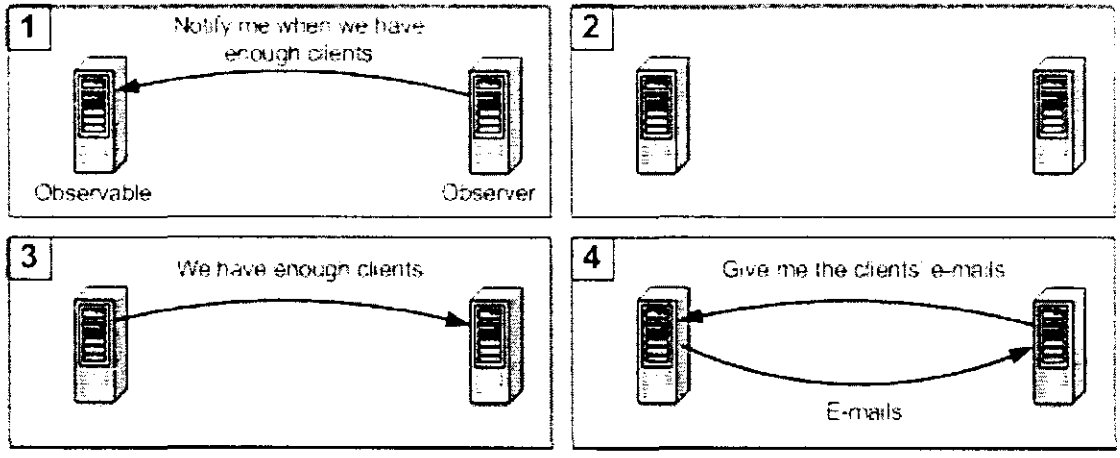


Tuy nhiên, nhược điểm của kỹ thuật này là:

_ Nếu khoảng thời gian định kỳ trên là rất nhỏ thì tải của mạng và việc chiếm dụng CPU sẽ tăng lên.

_ Nếu có nhiều client, đồng thời server cũng có nhiều thay đổi thì nó có thể bị quá tải vì việc trả lời các client.

Giải pháp cho vấn đề này là kỹ thuật thông báo (notification). Thay vì việc định kỳ gửi yêu cầu đến server, client chỉ cần gửi một yêu cầu ban đầu đề nghị server thông báo khi có thay đổi. Server sẽ gửi thông báo ngay khi có thay đổi.



+ Hai kiểu thông báo: kéo (push) và đẩy (pull)

_ Đẩy (pull): Server chỉ đơn thuần gửi thông báo là có thay đổi đến cho client, thông báo này chỉ thể hiện đó là thay đổi gì. Sau đó, client mới gửi yêu cầu xử lý tiếp theo đến cho server, chẳng hạn trong hình vẽ trên là "Give me the clients' email", và server gửi kết quả cho client. Cách tiếp cận này có vẻ như là có dư thừa việc gửi thông báo. Vì server có thể gửi luôn kết quả trong thông báo thứ nhất cho client. Tuy nhiên, kỹ thuật này là hiệu quả trong một số ưu trường hợp sau:

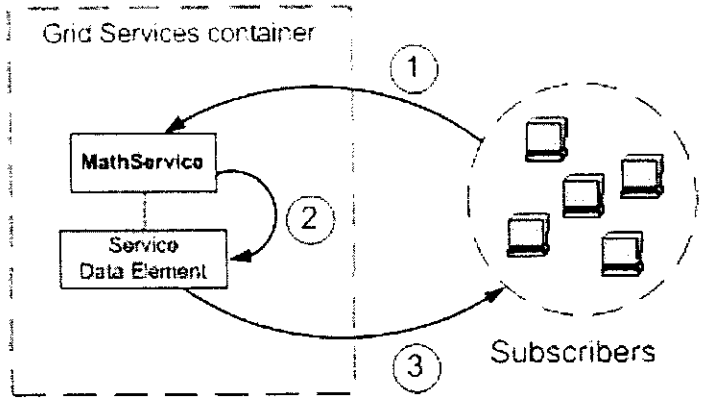
. Mỗi client cần nhận thông tin về thay đổi từ nhiều server

. Khi client không cần nhận dữ liệu từ server (client có thể chọn nhận hoặc lờ đi thông báo).

_ Kéo (push): Đây là kỹ thuật cho phép dữ liệu được gửi kèm với thông báo. Kỹ thuật này có ích trong trường hợp mỗi client cần nhận các thông tin giống nhau mỗi khi có thay đổi xảy ra. Nhìn chung, kỹ thuật kéo cho phép client có nhiều quyền hơn trong việc nhận dữ liệu sau khi có thông báo về thay đổi. Kỹ thuật đẩy giới hạn loại thông tin mà client có thể nhận được, tuy nhiên nó lại hiệu quả khi ta muốn tiết kiệm một lần gửi nhận thông báo.

+ Cơ chế thông báo trong GT3:

Cơ chế thông báo trong GT3 liên quan chặt chẽ đến dữ liệu dịch vụ. Trên thực tế, client chỉ quan tâm đến thay đổi trong một vài thông tin, hay một SDE cụ thể của dịch vụ đó.



Hình vẽ trên thể hiện các thao tác cần làm để thực hiện cơ chế thông báo cho dịch vụ lưới.
(1). addListener: ánh xạ lời gọi của client đến một SDE cụ thể (được xác định trong lời gọi).
(2). notifyChange: mỗi khi có thay đổi, dịch vụ MathService sẽ yêu cầu SDE gửi thông báo cho các client tương ứng cần thông báo từ SDE đó.
(3). deliverNotification: SDE gửi thông báo về thay đổi đến các client.
Qui trình 3 bước trên bao gồm cả việc gửi các dữ liệu dịch vụ cho client, đây là cách tiếp cận của kỹ thuật kéo. Chú ý rằng, GT3 chỉ hỗ trợ kỹ thuật kéo, mặc dù nó cũng có thể cho phép kỹ thuật đẩy được thực thi bằng cách gửi yêu cầu đến một SDE gọi là "dummy SDE". Trong GT3, observable được gọi là notification source, còn observer được gọi là notification sink.
Ví dụ sau được thực hiện trên phần mã có sẵn của dịch vụ MathService_sd. Lưu ý rằng, khi mà đã có dịch vụ với cơ chế dữ liệu dịch vụ, thì việc thêm cơ chế thông báo là rất đơn giản.

2.3.3.2. Phía server

Dịch vụ MathService_sd_notif
+ Định nghĩa giao diện dịch vụ gwsdl:
Việc thêm cơ chế thông báo không làm thay đổi giao diện dịch vụ. Mặc dù cần phải gửi đi các thông báo cho client, tuy nhiên ta không cần phải định nghĩa thêm các phương thức mới. Điều này được thực hiện thông qua các hàm có sẵn của một portType chuẩn gọi là NotificationSource. Ta đơn giản chỉ cần thêm phần in đậm sau vào file GWSDL:

```
<gwsdl:portType name="MathPortType" extends="ogsi:GridService
ogsi:NotificationSource">
<!-- <operation>s -->
<!-- <serviceData> -->
</gwsdl:portType>
```

Các phần còn lại giống như file MathService_sd (chú ý thay đổi namspace cho phù hợp: từ MathService_sd sang MathService_sd_notif). Toàn bộ mã nguồn của file này như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathService_sd_notifService"
targetNamespace="http://MathService_sd_notif/MathService_sd_notifSer
vice"
```

```
xmlns:tns="http://MathService_sd_notif/MathService_sd_notifService"
xmlns:data="http://MathService_sd_notif/MathService_sd_notifService/
MathSDE"
xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExt
ensions"
xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">


<import                                    location="../../ogsi/ogsi.gwsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
<import                                                location="MathSDE.xsd"
namespace="http://MathService_sd_notif/MathService_sd_notifService/M
athSDE"/>


<types>
<xsd:schema
targetNamespace="http://MathService_sd_notif/MathService_sd_notifSer
vice"                                    attributeFormDefault="qualified"
elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema">


      <!--BEGIN ELEMENT DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->
      <xsd:element name="subtract">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element name="arg1" type="xsd:int"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="subtractResponse">
            <xsd:complexType/>
      </xsd:element>
      <xsd:element name="add">
            <xsd:complexType>
```

```
                <xsd:sequence>
                    <xsd:element name="arg1" type="xsd:int"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="addResponse">
            <xsd:complexType/>
        </xsd:element>
        <!--END ELEMENT DEFINITIONS -->
</xsd:schema>
</types>

<!--BEGIN MESSAGE DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->

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

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

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

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

<!--END MESSAGE DEFINITIONS -->
```

```
<gwsdl:portType                    name="MathService_sd_notifPortType"
extends="ogsi:GridService ogsi:NotificationSource">
      <!--BEGIN OPERATION DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -
->
      <operation name="subtract">
            <input message="tns:subtractInputMessage"/>
            <output message="tns:subtractOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <operation name="add">
            <input message="tns:addInputMessage"/>
            <output message="tns:addOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <!--END OPERATION DEFINITIONS -->
<sd:serviceData       name="MathData"       type="data:MathDataType"
minOccurs="1" maxOccurs="1" mutability="mutable" modifiable="false"
nillable="false">
  </sd:serviceData>
</gwsdl:portType>
</definitions>
```

+ File thực thi java

Điểm khác biệt duy nhất so với file thực thi MathService_sd là SDE MathData sẽ thông báo cho tất cả các client tương ứng với nó mỗi khi phương thức add hay subtract được triệu gọi.

```
public void add(int a) throws RemoteException
{
mathDataValue.setLastOp("Addition");
incrementOps();
mathDataValue.setValue(mathDataValue.getValue() + a);
mathDataSDE.notifyChange();
}
```

Toàn bộ mã nguồn của nó như sau:

```
package MathService_sd_notif.impl;

import org.globus.ogsa.ServiceData;
import org.globus.ogsa.impl.ogsi.GridServiceImpl;
import org.globus.ogsa.GridContext;
import org.globus.ogsa.GridServiceException;

import MathService_sd_notif.stubs.MathService_sd_notifPortType;
import MathService_sd_notif.stubs.servicedata.MathDataType;
import java.rmi.RemoteException;

public class MathService_sd_notifImpl extends GridServiceImpl
{
    private ServiceData mathDataSDE;
    private MathDataType mathDataValue;

    public MathService_sd_notifImpl()
    {
        super("MathService_sd_notif Service with notification");
    }

    // Insert implementation here
    public void postCreate(GridContext context) throws
GridServiceException
    {
        // Call base class's postCreate
        super.postCreate(context);

        // Create Service Data Element
        mathDataSDE =
this.getServiceDataSet().create("MathData");
```

```
        // Create a MathDataType instance and set intial values
        mathDataValue = new MathDataType();
        mathDataValue.setLastOp("NONE");
        mathDataValue.setNumOps(0);
        mathDataValue.setValue(0);

        // Set the value of the SDE to the MathDataType instance
        mathDataSDE.setValue(mathDataValue);

        // Add SDE to Service Data Set
        this.getServiceDataSet().add(mathDataSDE);
    }


    // This method updates the MathData SDE increasing the
    // number of operations by one
    private void incrementOps()
    {
        int numOps = mathDataValue.getNumOps();
        mathDataValue.setNumOps(numOps + 1);
    }

    public void add(int a) throws RemoteException
    {
        mathDataValue.setLastOp("Addition");
        incrementOps();
        mathDataValue.setValue(mathDataValue.getValue() + a);
        mathDataSDE.notifyChange();
    }

    public void subtract(int a) throws RemoteException
```

```
    {
        mathDataValue.setLastOp("Subtraction");
        incrementOps();
        mathDataValue.setValue(mathDataValue.getValue() - a);
        mathDataSDE.notifyChange();
    }


}
```

+ File mô tả triển khai wsdd:

Mã nguồn của nó như sau:

```
<?xml version="1.0"?>
<deployment                                    name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

    <service     name="MathService_sd_notif"     provider="Handler"
style="wrapped">
        <parameter name="name" value="MathService_sd_notif"/>
        <parameter                                      name="className"
value="MathService_sd_notif.stubs.MathService_sd_notifPortType"/>
        <parameter                                     name="schemaPath"
value="schema/gt3ide/MathService_sd_notifService/MathService_sd_noti
f_service.wsdl"/>
        <parameter                          name="operationProviders"
value="org.globus.ogsa.impl.ogsi.NotificationSourceProvider" />
        <parameter                                  name="baseClassName"
value="MathService_sd_notif.impl.MathService_sd_notifImpl"/>

        <parameter name="allowedMethods" value="*"/>
        <parameter name="persistent" value="true"/>
        <parameter                                      name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>
    </service>
```

```
</deployment>
```

2.3.3.3. Phía client

File client phức tạp hơn một chút so với các file trong các ví dụ trước đây.

. Đầu tiên, phải viết tất cả mã trong phương thức main

. Client cần thực thi phương thức deliverNotification, đây là phương thức mà dịch vụ lưới sẽ gọi mỗi khi có thay đổi.

Ta sẽ viết 2 file client. ClientListener là rất quan trọng, nó sẽ gửi thông báo yêu cầu đến MathService và nhận các thông báo về thay đổi. Client MathAdder rất đơn giản, nó chỉ triệu gọi phương thức add(). Bằng cách này, ta có thể có một vài client cùng chờ thông báo tại cùng một thời điểm, và có thể thấy chúng cập nhật như thế nào khi ta chạy client thực hiện phép cộng.

_ ClientListener:

```
package MathService_sd_notif.impl;

import org.globus.ogsa.client.managers.NotificationSinkManager;
import org.globus.ogsa.NotificationSinkCallback;
import org.globus.ogsa.impl.core.service.ServicePropertiesImpl;
import org.globus.ogsa.utils.AnyHelper;
import org.gridforum.ogsi.ExtensibilityType;
import org.gridforum.ogsi.HandleType;
import org.gridforum.ogsi.ServiceDataValuesType;

import MathService_sd_notif.stubs.servicedata.MathDataType;

import java.rmi.RemoteException;

public class ClientListener extends ServicePropertiesImpl implements
NotificationSinkCallback {
    public static void main(String[] args)
    {
        try
        {
            // Get command-line arguments
            HandleType GSH = new HandleType(args[0]);
```

```
                ClientListener clientListener = new
ClientListener(GSH);
            }catch(Exception e)
            {
                System.out.println("ERROR!");
                e.printStackTrace();
            }
        }

        public ClientListener(HandleType GSH) throws Exception
        {
            // Start listening to the MathService
            NotificationSinkManager notifManager =
NotificationSinkManager.getManager();

        notifManager.startListening(NotificationSinkManager.MAIN_THREAD
);
            String sink = notifManager.addListener("MathData",
null, GSH, this);
            System.out.println("Listening...");

            // Wait for key press
            System.in.read();

            // Stop listening
            notifManager.removeListener(sink);
            notifManager.stopListening();
            System.out.println("Not listening anymore!");
    }

        public void deliverNotification(ExtensibilityType any) throws
RemoteException
        {
```

```
            try
            {
                  // Service Data has changed. Show new data.
                  ServiceDataValuesType serviceData =
AnyHelper.getAsServiceDataValues(any);
                  MathDataType mathData = (MathDataType)
AnyHelper.getAsSingleObject(serviceData, MathDataType.class);

                  // Write service data
                  System.out.println("Current value: " +
mathData.getValue());
                  System.out.println("Previous operation: " +
mathData.getLastOp());
                  System.out.println("# of operations: " +
mathData.getNumOps());
            }catch(Exception exc)
            {
                  System.out.println("ERROR!");
                  exc.printStackTrace();
            }
      }

}
```

Chú ý một số điểm sau:

```
public class ClientListener
extends ServicePropertiesImpl implements NotificationSinkCallback
```

Không như client ở các ví dụ trước, client này kế thừa một lớp và thực thi một giao diện.

```
. extends ServicePropertiesImpl:
```

Đây là lớp cơ sở của GridServiceImpl. Ta cần kế thừa một lớp phía server do client vừa triệu gọi MathService (tới SDE phù hợp) và nhận thông báo trả về (deliverNotification) – nó đóng vai trò của cả client và server.

```
. implements NotificationSinkCallback:
```

Giao diện này cần được thực thi bởi lớp muốn nhận thông báo. Một trong những yêu cầu của giao diện này là ta cần thực thi phương thức deliverNotification.
Phương thức main rất đơn giản, nó nhận duy nhất một đối số (GSH của MathService) và tạo một lớp MathListener. Tất cả phần "nghe" được định nghĩa trong hàm khởi tạo của lớp MathListener
Chú ý đoạn mã sau:

```
NotificationSinkManager notifManager =
NotificationSinkManager.getManager();

notifManager.startListening(NotificationSinkManager.MAIN_THREAD);

String sink = notifManager.addListener("MathData", null, GSH, this);
```

Lớp NotificationSinkManager thực hiện toàn bộ quá trình đệ trình yêu cầu nhận thông báo. Nó tiến hành thông qua 2 bước: báo cho bộ quản lý biết "bắt đầu nghe" và nói cho nó biết nó phải nghe cái gì. Lưu ý phương thức addListener với các đối số:

. Tên của SDE muốn đăng ký

. Khoảng thời gian cần nhận thông báo (timeout): là null nếu không muốn ngừng nghe.

. GSH của dịch vụ lưới chứa SDE mà ta muốn đăng ký

. Lớp phụ trách việc nhận thông báo
Một khi bạn đang nghe, bạn có hủy việc nhận thông báo bằng cách ấn phím bất kỳ. Quá trình hủy nhận thông báo được thực hiện trong hai bước: xóa listener mà ta đã tạo trước đó và ngừng tiến trình quản lý việc nghe.

```
notifManager.removeListener(sink);

notifManager.stopListening();
```

Phương thức deliverNotification có một tham số kiểu ExtensibilityType, đây là SDE sẽ được gửi kèm với thông báo. Như đã đề cập ở phần dữ liệu dịch vụ, đầu tiên ta phải chuyển kiểu ExtensibilityType thành MathDataType sử dụng các lớp helper.

```
ServiceDataValuesType serviceData =
AnyHelper.getAsServiceDataValues(any);

MathDataType mathData = (MathDataType)
AnyHelper.getAsSingleObject(serviceData, MathDataType)
```

Sau khi biên dịch, chạy file client bằng lệnh

```
# cd .../MathService_sd_notif/build/classes

#  java  -Dorg.globus.ogsa.schema.root=http://localhost:8080/
MathService_sd_notif/impl/ClientListener
http://localhost:8080/ogsa/services/MathService_sd_notif
```

Chú ý tham số *Dorg.globus.ogsa.schema.root*

Tham số này thiết lập trình URL cơ sở cho trình chứa dịch vụ lưới. Nếu mọi việc đều tốt đẹp, bạn sẽ thấy xuất hiện thông báo "Listening..." Nhớ rằng client này chỉ đơn thuần nghe các thông báo. Bây giờ, ta sẽ tiến hành tạo các thay đổi sử dụng client MathAdder để quan sát xem các thông báo được gửi đi như thế nào. Để quan sát các thông báo khi có nhiều listener, bạn có thể chạy nhiều MathListener cùng lúc.

_ MathAdder: Mã nguồn như sau:

```
package MathService_sd_notif.impl;


import
MathService_sd_notif.stubs.service.MathService_sd_notifServiceGridLo
cator;
import MathService_sd_notif.stubs.MathService_sd_notifPortType;
import java.net.URL;


public class MathAdder {
      public static void main(String[] args)
      {
            try
            {
                  // Get command-line arguments
                  URL GSH = new java.net.URL(args[0]);
                  int a = Integer.parseInt(args[1]);


                  // Get a reference to the Grid Service instance
                  MathService_sd_notifServiceGridLocator
mathServiceLocator = new MathService_sd_notifServiceGridLocator();
                  MathService_sd_notifPortType          math          =
mathServiceLocator.getMathService_sd_notifServicePort(GSH);


                  // Call remote method 'add'
                  math.add(a);
                  System.out.println("Added " + a);
            }catch(Exception e)
            {
```

```
                System.out.println("ERROR!");
                e.printStackTrace();
            }
        }

}
```

Mỗi lần chạy MathAdder, bạn sẽ thấy sự thay đổi của giá trị trả về và số phép toán đã thực hiện.

2.3.4. Dịch vụ tạm thời

Trong phần này, chúng ta sẽ nghiên cứu cách xây dựng dịch vụ ngắn hạn sử dụng mô hình nhà máy/thể hiện – một trong những tính năng được cung cấp bởi chuẩn OGSI.

Thêm khả năng ngắn hạn vào dịch vụ MathService:

+ File mô tả triển khai dịch vụ wsdd:

Chuyển một dịch vụ thành một nhà máy các dịch vụ trong suốt là một trong những việc dễ dàng nhất trong GT3. Ta không cần thay đổi cả file giao diện dịch vụ lẫn file thực thi. Điều duy nhất cần làm là sửa file mô tả triển khai wsdd.

Để ý file mô tả triển khai của dịch vụ MathService nếu khôgn dùng cơ chế ngắn hạn:

```
<?xml version="1.0"?>
<deployment                                    name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
      <service name="MathService" provider="Handler" style="wrapped">
            <parameter name="name" value="MathService (Factory)"/>
            <parameter name="allowedMethods" value="*"/>
            <parameter name="persistent" value="true"/>
            <parameter                                    name="className"
value="org.gridforum.ogsi.Factory"/>
            <parameter                                 name="baseClassName"
value="org.globus.ogsa.impl.ogsi.GridServiceImpl"/>
            <parameter .                                  name="schemaPath"
value="schema/ogsi/ogsi_factory_service.wsdl"/>
            <parameter                                  name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>
      </service>
```

```
</deployment>
```

File mô tả triển khai dịch vụ sau thể hiện chính xác các dịch vụ giống nhau được tạo ra từ mô hình nhà máy. Chú ý rằng điều này có nghĩa là ta phải có một dịch vụ nhà máy tồn tại cố định và ta sẽ sử dụng nó để tạo ra các thể hiện tạm thời.

```
<?xml version="1.0"?>

<deployment                                    name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"

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

      <service name="MathService" provider="Handler" style="wrapped">

            <parameter name="name" value="MathService (Factory)"/>

            <parameter     name="instance-name"     value="MathService
(Instance)"/>

            <parameter                              name="instance-className"
value="MathService.stubs.MathServicePortType"/>

            <parameter                             name="instance-schemaPath"
value="schema/gt3ide/MathServiceService/MathService_service.wsdl"/>

            <parameter                         name="instance-baseClassName"
value="MathService.impl.MathServiceImpl"/>


      <!-- Start common parameters -->

            <parameter name="allowedMethods" value="*"/>

            <parameter name="persistent" value="true"/>

            <parameter                                      name="className"
value="org.gridforum.ogsi.Factory"/>

            <parameter                                  name="baseClassName"
value="org.globus.ogsa.impl.ogsi.GridServiceImpl"/>

            <parameter                                    name="schemaPath"
value="schema/ogsi/ogsi_factory_service.wsdl"/>

            <parameter                                   name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>

            <parameter                                name="factoryCallback"
value="org.globus.ogsa.impl.ogsi.DynamicFactoryCallbackImpl"/>

            <parameter                               name="operationProviders"
value="org.globus.ogsa.impl.ogsi.FactoryProvider"/>

      </service>
```

```
</deployment>
```

Hãy so sánh 2 file triển khai dịch vụ.

Đầu tiên, ta thêm tiền tố "instance-" vào các tham số name, schemaPath, baseClassName và className. Bằng cách này, ta đã báo cho trình chứa biết rằng, các tham số này sẽ là của các thể hiện dịch vụ được tạo ra bởi nhà máy.

Sau đó, ta thêm một khối các tham số trong phần "common parameters block". Chú ý cách mà schemaPath, baseClassName và className tham chiếu tới mã của nhà máy (được cung cấp bởi GT3) và ta không phải viết thêm bất kì một đoạn mã nào. Tất cả mã cần thiết để thực thi mô hình nhà máy/thể hiện đã được GT3 cung cấp.

- Phía client

+ Do giao diện dịch vụ là không thay đổi, cho nên ta có thể sử dụng lại file client của MathService trong ví dụ đầu tiên. Toàn bộ mã nguồn của nó là:

```
package MathService.impl;

import MathService.stubs.service.MathServiceServiceGridLocator;
import MathService.stubs.MathServicePortType;
import java.net.URL;

public class MathServiceClient1 {
    public static void main(String[] args)
    {
    try
    {
        //    Get command-line arguments
        URL GSH = new java.net.URL(args[0]);
        int a = Integer.parseInt(args[1]);
        //    Get a reference to the MathService instance
        MathServiceServiceGridLocator mathServiceLocator = new
MathServiceServiceGridLocator();
        MathServicePortType math =
mathServiceLocator.getMathServiceServicePort(GSH);
        //    Call remote method 'add'
        math.add(a);
        System.out.println("Added " + a);
        //    Get current value through remote method 'getValue'
```

```
            int value = math.getValue();
            System.out.println("Current value: " + value);
        }
        catch(Exception e)
        {
            System.out.println("ERROR!");
            e.printStackTrace();
        }
}
}
```

Tuy nhiên, client này chỉ làm việc với các dịch vụ có giao diện của MathService, trong ví dụ này là các dịch vụ tạm thời. Cho nên, trước khi triệu dọi từ client, ta cần phải tạo một thể hiện của dịch vụ bằng cách dùng lệnh

```
# ogsi-create-service
```

Cách tiếp cận nhà máy/thể hiện rất có ích khi client muốn tự động tạo một dịch vụ, dùng nó cho một mục đích cụ thể và hủy nó sau khi thực hiện xong công việc.

Bạn có thể hủy một thể hiện bằng cách dùng lệnh

```
# ogsi-destroy-service
```

+ File client thứ hai này không giao tiếp trực tiếp với một thể hiện đã có sẵn. Thay vào đó, đầu tiên nó kết nối tới nhà máy dịch vụ MathService, yêu cầu tạo một thể hiện dịch vụ, sử dụng nó và sau đó hủy đi. Toàn bộ mã nguồn của nó là:

```
package MathService.impl;

import org.gridforum.ogsi.OGSIServiceGridLocator;
import org.gridforum.ogsi.Factory;
import org.gridforum.ogsi.LocatorType;
import org.globus.ogsa.utils.GridServiceFactory;

import MathService.stubs.service.MathServiceServiceGridLocator;
import MathService.stubs.MathServicePortType;
import java.net.URL;
```

```
public class MathServiceClient2 {
    public static void main(String[] args).
    {
    try
    {
        //    Get command-line arguments
        URL GSH = new java.net.URL(args[0]);
        int a = Integer.parseInt(args[1]);

        // Create a new instance
        OGSIServiceGridLocator gridLocator=new
OGSIServiceGridLocator();
        Factory fac=gridLocator.getFactoryPort(GSH);
        GridServiceFactory mathFac=new GridServiceFactory(fac);
        LocatorType loc=mathFac.createService();

        // Get a reference to the MathService instance
        MathServiceServiceGridLocator mathServiceLocator = new
MathServiceServiceGridLocator();
        MathServicePortType math =
mathServiceLocator.getMathServiceServicePort(loc);

        //    Call remote method 'add'
        math.add(a);
        System.out.println("Added " + a);
        //    Get current value through remote method 'getValue'
        int value = math.getValue();
        System.out.println("Current value: " + value);

        // Destroy the instance
        math.destroy();
```

```
        }
        catch(Exception e)
        {
                System.out.println("ERROR!");
                e.printStackTrace();
        }
}
}
```

Chú ý cách kết nối tới nhà máy dịch vụ MathService:

```
OGSIServiceGridLocator gridLocator = new OGSIServiceGridLocator();
Factory factory = gridLocator.getFactoryPort(GSH);
GridServiceFactory mathFactory = new GridServiceFactory(factory);
```

Mục đích của 3 dòng lệnh này là tham chiếu tới nhà máy. Sau đó, ta tạo một thể hiện mới với nhà máy này:

```
LocatorType locator = mathFactory.createService();
MathServiceGridLocator mathLocator = new MathServiceGridLocator();
MathPortType math = mathLocator.getMathService(locator);
```

Đầu tiên ta gọi phương thức createService() - là một phương thức của nhà máy. Phương thức này sẽ trả về định vị của thể hiện, cái mà ta sẽ dùng để tham chiếu tới MathPortType trong hai dòng lệnh còn lại. Sau 3 dòng lệnh này, ta đã sẵn sàng để triệu gọi thể hiện của dịch vụ. Việc triệu gọi là hoàn toàn giống như ở client trước. Tuy nhiên, sau khi dùng xong thể hiện, ta phải hủy nó. Điều này được thực hiện bằng phương thức destroy() của portType dịch vụ lưới. Phương thức này ra lệnh cho một dịch vụ lưới hủy chính nó. Do MathPortType kế thừa từ GridService (như là tất cả các dịch vụ lưới khác) nên ta có thể triệu gọi phương thức destroy() trực tiếp từ MathPortType.

```
math.destroy();
```

2.3.5. Cơ chế lưu vết

Khi triển khai dịch vụ lưới trên nền GT3.0, ta có thể tận dụng một đặc tính rất hay được trình chứa hỗ trợ là cơ chế lưu vết. Cơ chế này sẽ đưa ra các thông tin về quá trình thực thi dịch vụ. Các thông tin có thể là cảnh báo, thông tin về lỗi... được đưa ra màn hình hay ghi vào một file để dễ bảo trì về sau. Để minh họa tính chất này, ta vẫn sử dụng ví dụ ở phần nhà cung cấp chức năng.

Cơ chế bảng lưu vết chia các thông tin về dịch vụ thành các mức độ khác nhau. Các khoá có sáu mức (xếp theo chiều tăng của mức độ cảnh báo):

- Dò lỗi (debug).
- Theo dõi (trace).

- Thông báo (info).
- Cảnh báo (warn).
- Lỗi (error).
- Sự cố nghiêm trọng (fatal).

Cách sử dụng: khi viết file thực thi dịch vụ ta khai báo sử dụng thêm hai gói:

```
org.apache.commons.logging.Log
org.apache.commons.logging.LogFactory
```

+Trước khi thực hiện các hàm dịch vụ, ta tạo một chỉ dẫn của lớp bằng lệnh:

```
static Log logger =
LogFactory.getLog(MathService_logImpl.class.getName());
```

+Sau đó, bất cứ chỗ nào cần có chỉ dẫn, ta chỉ cần sử dụng đối tượng logger. Ví dụ như:

```
logger.info( "Information")
logger.warn("Warning").
```

Mã nguồn của file này như sau:

```
package MathService_log.impl;

import org.globus.ogsa.impl.ogsi.GridServiceImpl;
import MathService_log.stubs.MathService_logPortType;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import java.rmi.RemoteException;

public class MathService_logImpl extends GridServiceImpl implements
MathService_logPortType
{
    public MathService_logImpl()
    {
        super("MathService_log");
    }

    // Insert implementation here
```

```
    // Create this class's logger
    static Log logger =
LogFactory.getLog(MathService_logImpl.class.getName());

    private int value = 0;

    public void add(int a) throws RemoteException
    {
        logger.info("Addition invoked with parameter a=" +
String.valueOf(a));
        if (a==0)
            logger.warn("Adding zero doesn't modify the internal
value!");
        value = value + a;
    }

    public void subtract(int a) throws RemoteException
    {
        logger.info("Addition invoked with parameter a=" +
String.valueOf(a));
        if (a==0)
            logger.warn("Subtracting zero doesn't modify the
internal value!");
        value = value - a;
    }

    public int getValue() throws RemoteException
    {
        logger.info("getValue() invoked");
        return value;
    }
}
```

Theo mặc định, các thông tin khoá được in ra màn hình của người sử dụng. Ta có thể lưu các thông tin này vào một file văn bản. Thực hiện như sau:
+Tìm file $GLOBUS_LOCATION/ogsilogging.properties
+Thêm dòng sau:

```
MathService_log.impl.MathService_logImpl=file_name,info
```

Dòng tham số trên có ý nghĩa là, tất cả các thông tin khoá có mức độ từ info trở lên (tức là bao gồm info, warn, error, fatal) được ghi ra file có tên là file_name. Đường dẫn tới file này được xác định như sau:
+Tìm file $GLOBUS_LOCATION/ogsilogging_parms.properties
+Trong tuỳ chọn logDestinationBasePath =..., ta thêm vào đường dẫn tới file đầu ra. Ví dụ như file đầu ra đặt ở /tmp/file_name. Ta điền như sau:

```
logDestination=/tmp/
```

Mã nguồn các file khác:
File mô tả dịch vụ GWSDL:

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions                                    name="MathService_logService"
targetNamespace="http://MathService_log/MathService_logService"
xmlns:tns="http://MathService_log/MathService_logService"
xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExt
ensions"
xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">


<import                                    location="../../ogsi/ogsi.gwsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>


<types>

<xsd:schema
targetNamespace="http://MathService_log/MathService_logService"
attributeFormDefault="qualified"        elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema">

      <!--BEGIN ELEMENT DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->

      <xsd:element name="getValue">

            <xsd:complexType/>
```

```
        </xsd:element>
        <xsd:element name="getValueResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="value" type="xsd:int"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="subtract">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="arg1" type="xsd:int"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="subtractResponse">
            <xsd:complexType/>
        </xsd:element>
        <xsd:element name="add">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="arg1" type="xsd:int"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="addResponse">
            <xsd:complexType/>
        </xsd:element>
        <!--END ELEMENT DEFINITIONS -->
</xsd:schema>
</types>
```

```
<!--BEGIN MESSAGE DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->

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

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

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

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

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

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

<!--END MESSAGE DEFINITIONS -->

<gwsdl:portType                          name="MathService_logPortType"
extends="ogsi:GridService">
```

```
      <!--BEGIN OPERATION DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -
->
      <operation name="getValue">
            <input message="tns:getValueInputMessage"/>
            <output message="tns:getValueOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <operation name="subtract">
            <input message="tns:subtractInputMessage"/>
            <output message="tns:subtractOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <operation name="add">
            <input message="tns:addInputMessage"/>
            <output message="tns:addOutputMessage"/>
            <fault name="Fault" message="ogsi:FaultMessage"/>
      </operation>
      <!--END OPERATION DEFINITIONS -->
</gwsdl:portType>
</definitions>
```

File mô tả triển khai WSDD

```
<?xml version="1.0"?>
<deployment                                    name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

      <service        name="MathService_log"        provider="Handler"
style="wrapped">
            <parameter name="name" value="MathService_log"/>
            <parameter                                    name="className"
value="MathService_log.stubs.MathService_logPortType"/>
```

```
            <parameter                                    name="schemaPath"
value="schema/gt3ide/MathService_logService/MathService_log_service.
wsdl"/>

            <parameter                                  name="baseClassName"
value="MathService_log.impl.MathService_logImpl"/>

            <parameter name="allowedMethods" value="*"/>
            <parameter name="persistent" value="true"/>
            <parameter                                   name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>
      </service>
</deployment>
```

- File Client

```
package MathService_log.impl;

import
MathService_log.stubs.service.MathService_logServiceGridLocator;
import MathService_log.stubs.MathService_logPortType;
import java.net.URL;

public class MathService_logClient {
      public static void main(String[] args)
      {
      try
      {
            //    Get command-line arguments
            URL GSH = new java.net.URL(args[0]);
            int a = Integer.parseInt(args[1]);
            //    Get a reference to the MathService instance
            MathService_logServiceGridLocator   mathServiceLocator   =
new MathService_logServiceGridLocator();
```

```
            MathService_logPortType                    math                    =
mathServiceLocator.getMathService_logServicePort(GSH);
            //      Call remote method 'add'
            math.add(a);
            System.out.println("Added " + a);
            //      Get current value through remote method 'getValue'
            int value = math.getValue();
            System.out.println("Current value: " + value);
      }
      catch(Exception e)
      {
            System.out.println("ERROR!");
            e.printStackTrace();
      }
}
}
```

2.3.6. Cơ chế quản lý vòng đời dịch vụ

Vòng đời của một dịch vụ được tính từ khi nó được khởi tạo cho đến khi nõ bị hủy đi. Với mô hình nhà máy/thể hiện trong OGSI, ta phải đối mặt với vấn đề quản lý vòng đời dịch vụ. Đối với các dịch vụ lưới thuần túy, vòng đời của nó là rất đơn giản: nó tồn tại cùng với trình chứa dịch vụ lưới, nên được tạo ra khi trình chứa được kích hoạt và bị hủy đi khi trình chứa ngừng. Tuy nhiên với các dịch vụ tạm thời, việc quản lý vòng đời dịch vụ trở nên không còn tầm thường nữa. Một số dịch vụ phải đáp ứng yêu cầu không chỉ sống trong thời gian sống của client mà còn phải sống trong cả thời gian sống của server chứa nó. Điều này có nghĩa là nếu server được khởi động lại, thể hiện dịch vụ nào phải được nạp lại giá trị mà nó có trước khi server bị tắt.

GT3 cung cấp các công cụ cần thiết cho việc quản lý vòng đời dịch vụ lưới. Chẳng hạn ta có thể cho thể hiện dịch vụ chạy một số đoạn mã ngay trước khi nó khởi tạo hay ngay trước khi bị hủy (để nạp và ghi các giá trị bên trong lên bộ nhớ thứ cấp).

Phần mã mà ta sử dụng trong ví dụ sau là sự kết hợp của hai ví dụ ở hai phần trước: dịch vụ tạm thời và cơ chế lưu vết. Ta sử dụng lại phần mã thực thi trong ví dụ về cơ chế lưu vết và file triển khai dịch vụ trong ví dụ về dịch vụ tạm thời.

2.3.6.1. Các phương thức callback

Đây là một trong những phương pháp quản lý vòng đời thể hiện dịch vụ được sử dụng trong GT3. Những phương thức này được triệu gọi vào các thời điểm xác định trong vòng đời thể hiện (chẳng hạn trong quá trình khởi tạo hoặc hủy thể hiện dịch vụ).

Việc triển khai các phương thức callback là rất đơn giản. Ta cần thực thi giao diện GridServiceCallback - giao diện chứa tất cả các phương thức callback. Lớp sau thực thi tất cả các giao diện callback.

```
// ...
import org.globus.ogsa.GridServiceCallback;
// ...
public class MathImpl extends GridServiceImpl
implements MathPortType, GridServiceCallback
{
// ...
// Callback methods
public void preCreate(GridServiceBase base) throws
GridServiceException
{
super.preCreate(base);
logger.info("Instance is going to be created (preCreate)");
}
public void postCreate(GridContext context) throws
GridServiceException
{
super.postCreate(context);
logger.info("Instance has been created (postCreate)");
}
public void activate(GridContext context) throws
GridServiceException
{
super.activate(context);
logger.info("Instance has been activated (activate)");
}
public void deactivate(GridContext context) throws
GridServiceException
{
super.deactivate(context);
```

```
logger.info("Instance has been deactivated (deactivate)");
}
public void preDestroy(GridContext context) throws
GridServiceException
{
super.preDestroy(context);
logger.info("Instance is going to be destroyed (preDestroy)");
}
}
```

Đầu tiên, chú ý cách gọi các phương thức callback cơ sở từ các phương thức callback mà ta xây dựng. Ví dụ, trong phương thức postCreate:

```
public void postCreate(GridContext context) throws
GridServiceException
{
super.postCreate(context);
logger.info("Instance has been created (postCreate)");
}
```

Điều này là rất quan trọng, nó giúp các phương thức callback cơ sở trong lớp cơ sở GridServiceImpl cũng được gọi. Các phương thức callback của GridServiceImpl phụ trách việc khởi tạo rất nhiều giá trị bên trong (bao gồm cả dữ liệu dịch vụ), do đó nếu chúng không được triệu gọi có thể sẽ gây ra các kết quả không mong muốn.

Chú ý: không cần gọi các phương thức callback của lớp cơ sở khi dùng mô hình nhà cung cấp. Nhớ rằng nhà cung cấp không kế thừa GridServiceImpl, nhưng nó vẫn chạy trong nền (ta đã mô tả trong file wsdd). Trình chứa dịc vụ lưới sẽ đảm bảo các phương thức callback của GridServiceImpl và nhà cung cấp sẽ được triệu gọi khi cần thiết.

Các phương thức callback được gọi:

. preCreate: phương thức này được triệu gọi khi dịch vụ lưới bắt đầu tiến trình khởi tạo. Lúc này, các thông số cấu hình của nó chưa được nạp.

. postCreate: phương thức này được gọi khi một dịch vụ đã được khởi tạo và tất cả các thông số cấu hình của nó đã được thiết lập.

. activate: mặc định là mọi dịch vụ lưới đều ở trạng thái không được kích hoạt (nghĩa là chúng vẫn không được nạp vào bộ nhớ). Phương thức callback này được triệu gọi khi một dịch vụ được kích hoạt.

. deactivate: phương thức này được triệu gọi trước khi dịch vụ chuyển sang trạng thái không kích hoạt.

. preDestroy: phương thức này được triệu gọi ngay trước khi dịch vụ bị hủy.

Toàn bộ mã nguồn của file này như sau:

```
package MathService_lifeCycle.impl;

import org.globus.ogsa.GridServiceBase;
import org.globus.ogsa.GridServiceException;
import org.globus.ogsa.GridContext;
import org.globus.ogsa.GridServiceCallback;
import org.globus.ogsa.impl.ogsi.GridServiceImpl;

import MathService_lifeCycle.stubs.MathService_lifeCyclePortType;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import java.rmi.RemoteException;

public class MathService_lifeCycleImpl extends GridServiceImpl
implements MathService_lifeCyclePortType, GridServiceCallback
{
    public MathService_lifeCycleImpl()
    {
        super("MathService_lifeCycle");
    }

    // Insert implementation here
    // Create this class's logger
    static Log logger =
LogFactory.getLog(MathService_lifeCycleImpl.class.getName());

    private int value = 0;

    public void add(int a) throws RemoteException
    {
```

```
            logger.info("Addition invoked with parameter a=" +
String.valueOf(a));
            if (a==0)
                  logger.warn("Adding zero doesn't modify the internal
value!");
            value = value + a;
      }

      public void subtract(int a) throws RemoteException
      {
            logger.info("Addition invoked with parameter a=" +
String.valueOf(a));
            if (a==0)
                  logger.warn("Subtracting zero doesn't modify the
internal value!");
            value = value - a;
      }

      public int getValue() throws RemoteException
      {
            logger.info("getValue() invoked");
            return value;
      }

      // Callback methods
      public void preCreate(GridServiceBase base) throws
GridServiceException
      {
            super.preCreate(base);
            logger.info("Instance is going to be created
(preCreate)");
      }
```

```
    public void postCreate(GridContext context) throws
GridServiceException
    {
        super.postCreate(context);
        logger.info("Instance has been created (postCreate)");
    }

    public void activate(GridContext context) throws
GridServiceException
    {
        super.activate(context);
        logger.info("Instance has been activated (activate)");
    }

    public void deactivate(GridContext context) throws
GridServiceException
    {
        super.deactivate(context);
        logger.info("Instance has been deactivated
(deactivate)");
    }

    public void preDestroy(GridContext context) throws
GridServiceException
    {
        super.preDestroy(context);
        logger.info("Instance is going to be destroyed
(preDestroy)");
    }
}
```

+ File mô tả triển khai dịch vụ:

File mô tả triển khai dịch vụ trong trường hợp này rất giống với trường hợp dịch vụ tạm thời. Sự thay đổi duy nhất là GSH của dịch vụ và baseClassName (lớp mà ta vừa viết: ví dụ cơ chế lưu vết cộng thêm các phương thức callback).

```
<?xml version="1.0"?>

<deployment                                    name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"

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


      <service     name="MathService_lifeCycle"     provider="Handler"
style="wrapped">

            <parameter     name="name"     value="MathService_lifeCycle
(Factory)"/>

            <parameter                                  name="instance-name"
value="MathService_lifeCycle (Instance)"/>

            <parameter                               name="instance-className"
value="MathService_lifeCycle.stubs.MathService_lifeCyclePortType"/>

            <parameter                              name="instance-schemaPath"
value="schema/gt3ide/MathService_lifeCycleService/MathService_lifeCy
cle_service.wsdl"/>

            <parameter                         name="instance-baseClassName"
value="MathService_lifeCycle.impl.MathService_lifeCycleImpl"/>

            <parameter name="instance-deactivation" value="10000"/>


            <parameter name="allowedMethods" value="*"/>

            <parameter name="persistent" value="true"/>

            <parameter                                      name="className"
value="org.gridforum.ogsi.Factory"/>

            <parameter                                   name="baseClassName"
value="org.globus.ogsa.impl.ogsi.GridServiceImpl"/>

            <parameter                                     name="schemaPath"
value="schema/ogsi/ogsi_factory_service.wsdl"/>

            <parameter                                   name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>

            <parameter                                name="factoryCallback"
value="org.globus.ogsa.impl.ogsi.DynamicFactoryCallbackImpl"/>

            <parameter                               name="operationProviders"
value="org.globus.ogsa.impl.ogsi.FactoryProvider"/>
```

```
        <parameter                          name="lifecycleMonitorClass"
value="MathService_lifeCycle.impl.MathLifecycleMonitor"/>
    </service>
</deployment>
```

+ Trước khi biên dịch dịch vụ này, thêm dòng sau vào cuối file $GLOBUS_LOCATION/ogsilogging.properties để tạo khả năng lưu vết cho lớp vừa xây dựng:

```
MathService_lifeCycle.impl.MathService_lifeCycleImpl=console,info
```

2.3.6.2. Theo dõi vòng đời dịch vụ

Các phương thức callback vừa xây dựng khá thú vị, song nó không có ích cho việc kế thừa nhiều lắm. Giả sử bạn muốn dùng lại hai phương thức preCreate và postCreate cho một vài dịch vụ khác, cách duy nhất mà bạn có thể làm là cho các dịch vụ đó kế thừa từ nó. Tuy nhiên các dịch vụ này đã kế thừa từ dịch vụ lưới, nên cách làm này là không khả thi.

Giải pháp được đề xuất trong GT3 là dùng cơ chế theo dõi còng đời dịch vụ. Lớp theo dõi vòng đời dịch vụ thực thi ServiceLifecycleMonitor, một giao diện với các phương thức callback sẽ được triệu gọi tại các thời điểm nhất định trong vòng đời dịch vụ. Ta không cần kế thừa dịch vụ này hay là tham chiếu tới nó trong mã nguồn. Chỉ cần thêm một dòng vào file mô tả triển khai thể hiện ta cần gọi theo dõi vòng đời dịch vụ khi có các sự kiện đặc biệt đó xảy ra. Tất nhiên là ta có thể dùng cùng một đối tượng theo dõi vòng đời cho nhiều dịch vụ lưới khác nhau.

Đoạn mã sau chỉ một lớp theo dõi vòng đời dịch vụ đơn giản, chỉ là ghi các thông báo ra một file.

```
package MathService_lifeCycle.impl;


import org.globus.ogsa.GridServiceException;
import org.globus.ogsa.ServiceLifecycleMonitor;
import org.globus.ogsa.GridContext;


import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;


public class MathLifecycleMonitor implements ServiceLifecycleMonitor
{
    // Create this class's logger
    static                Log                logger                =
LogFactory.getLog(MathLifecycleMonitor.class.getName());
```

```
    public void create(GridContext context) throws
GridServiceException
    {
        logger.info("Instance is going to be created (create)");
    }

    public void destroy(GridContext context) throws
GridServiceException
    {
        logger.info("Instance is going to be destroyed
(destroy)");
    }

    public void preCall(GridContext context) throws
GridServiceException
    {
        logger.info("Service is going to be invoked (preCall)");
    }

    public void postCall(GridContext context) throws
GridServiceException
    {
        logger.info("Service invocation has finished
(postCall)");
    }

    public void preSerializationCall(GridContext context)
    {
        logger.info("Input parameters are going to be
deserialized (preSerializationCall)");
    }
```

```
    public void postSerializationCall(GridContext context)
    {
        logger.info("Input parameters have been deserialized
(postSerializationCall)");
    }

}
```

Để các thông báo được ghi ra, cần thêm dòng sau vào cuối file $GLOBUS_LOCATION/ogsilogging.properties

```
MathService_lifeCycle.impl.MathLifecycleMonitor =console,info
```

Ngoài ra, để thông báo cho trình chứa dịch vụ lưới biết bạn muốn sử dụng đối tượng theo dõi vòng đời dịch vụ này, bạn cần thêm dòng tham số sau vào file mô tả triển khai:

```
<parameter name="lifecycleMonitorClass"
value="MathService_lifeCycle.impl.MathLifecycleMonitor"/>
```

- Ta có thể điều khiển một số tham số của vòng đời dịch vụ trong file mô tả triển khai. Ví dụ, dòng sau cho phép quyết định khi nào thể hiện dịch vụ chuyển sang trạng thái không kích hoạt:

```
<parameter name="instance-deactivation" value="10000"/>
```

Đơn vị thời gian được đo là mili giây. Sau khi được kích hoạt, theo mặc định thì một dịch vụ lưới sẽ tồn tại ở trạng thái kích hoạt mãi. Với dòng mã trên, dịch vụ sẽ chỉ ở trạng thái kích hoạt trong 10 giây, sau đó chuyển sang trạng thái không kích hoạt. Trong một số trường hợp, ta cần giải phóng thể hiện một khoảng thời gian, sau đó nạp lại để tiết kiệm tài nguyên. Mã nguồn file mô tả triển khai như sau:

```
<?xml version="1.0"?>
<deployment name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

    <service name="MathService_lifeCycle" provider="Handler"
style="wrapped">
        <parameter name="name" value="MathService_lifeCycle
(Factory)"/>
        <parameter name="instance-name"
value="MathService_lifeCycle (Instance)"/>
```

```
            <parameter                              name="instance-className"
value="MathService_lifeCycle.stubs.MathService_lifeCyclePortType"/>
            <parameter                             name="instance-schemaPath"
value="schema/gt3ide/MathService_lifeCycleService/MathService_lifeCy
cle_service.wsdl"/>
            <parameter                         name="instance-baseClassName"
value="MathService_lifeCycle.impl.MathService_lifeCycleImpl"/>
            <parameter name="instance-deactivation" value="10000"/>


            <parameter name="allowedMethods" value="*"/>
            <parameter name="persistent" value="true"/>
            <parameter                                    name="className"
value="org.gridforum.ogsi.Factory"/>
            <parameter                                 name="baseClassName"
value="org.globus.ogsa.impl.ogsi.GridServiceImpl"/>
            <parameter                                   name="schemaPath"
value="schema/ogsi/ogsi_factory_service.wsdl"/>
            <parameter                                  name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>
            <parameter                               name="factoryCallback"
value="org.globus.ogsa.impl.ogsi.DynamicFactoryCallbackImpl"/>
            <parameter                             name="operationProviders"
value="org.globus.ogsa.impl.ogsi.FactoryProvider"/>


            <parameter                         name="lifecycleMonitorClass"
value="MathService_lifeCycle.impl.MathLifecycleMonitor"/>
      </service>
</deployment>
```

2.4. Lập trình dịch vụ lưới có cơ chế bảo mật

2.4.1. Giới thiệu

Bảo mật mức truyền thông điệp trong GT3 dựa trên các chuẩn bảo mật Web Service và XML (WS-Security, XML Encryption và XML Signature). GT3 cung cấp hai cơ chế chứng thực mức truyền thông điệp khác nhau: an toàn các hội thoại trong môi trường lưới (GSI Secure Conversation) và an toàn các thông điệp trong môi trường lưới (GSI Secure Message).

Với phương thức an toàn các hội thoại, một ngữ cảnh bảo mật được thiết lập trước tiên giữa client và service. Ngữ cảnh này sau đó được dùng để kí, xác nhận, mã hóa và giải mã lên các thông điệp.

Với phương thức bảo mật thông điệp, một thông điệp được kí và mã hóa theo chuẩn giấy ủy nhiệm X509. Mã hóa trong trường hợp này được thực hiện theo hai bước.

- Một khóa không đối xứng có kích cỡ là 128 bits, được sinh ra dùng kỹ thuật mã hóa không đối xứng AES, để mã hóa thân các thông điệp.

- Chính khóa mã không đối xứng này lại được mã hóa bởi các giải thuật RSA/OAEP, sử dụng khóa công khai đã xác định của người nhận.

An toàn các hội thoại (GSI Secure Conversation) đòi hỏi mức bảo mật phức tạp hơn an toàn các thông điệp (GSI Secure Message). An toàn thông điệp cũng dễ dàng cài đặt hơn, bởi vậy nó phù hợp hơn cho các tương tác yêu cầu - đáp ứng đơn giản(simple request-response).

Trong một cuộc truyền thông bảo mật trên môi trường lưới, bạn cần thiết lập cơ chế bảo mật phía service (Server), client và cơ chế quản lý GRIM (GRIM handle) một cách phù hợp. Khi đó dịch vụ bảo mật của bạn mới có thể hoạt động được. Sau đây là công thức tổng quát nhất cho bảo mật mức truyền thông điệp:

Grim policy handling + credential delegation + authorization = message security

2.4.2. Viết dịch vụ bảo mật đầu tiên

- Phía server

Các công việc để cho phép bảo mật phía server cũng hoàn toàn tương tự như bạn viết một dịch lưới thông thường, chỉ có điều là ta thêm vài kỹ thuật nhỏ để hỗ trợ bảo mật ở phía server.

+ Đặc tả giao diện cho dịch vụ

File đặc tả giao diện của dịch vụ hoàn toàn giống như file GWSDL mà chúng ta đã xây dựng ở các ví dụ trước, không cần thiết phải tạo mới khi thêm bảo mật cho dịch vụ. Chúng sẽ bao gồm các phương thức add, subtract, và getValue.

+ Cài đặt dịch vụ

Ta sẽ sử dụng một OperationProvider để cài đặt cho dịch vụ. Nó sẽ bao gồm cài đặt cho các phương thức add, subtract, và getValue được đặc tả ở giao diện, các phương thức gọi ngược (callback methods) được thừa kế từ lớp OperationProvider (ở đây ta chỉ cài đặt một phương thức postCreate), và một phương thức private logSecurityInfo được dùng để ghi ra các thông tin bảo mật khi dịch vụ hoạt động. Ở trong ví dụ này, các thông tin được đưa ra là không cần thiết, nhưng nó sẽ hữu ích cho các ví dụ tiếp theo.

Chúng ta sẽ thảo luận kĩ hơn về các thành phần liên quan trong mã nguồn. Cài đặt của các phương thức chính, thể hiện chức năng của dịch vụ là rất đơn giản. Chỉ khác là ta triệu gọi phương thức logSecurityInfo trong tất cả các phương thức này, để ghi lại các thông tin về bảo mật khi thực hiện chúng.

```
public void add(int a) throws RemoteException
{
  logSecurityInfo("add");
  value = value + a;
}

public void subtract(int a) throws RemoteException
{
  logSecurityInfo("subtract");
  value = value - a;
}

public int getValue() throws RemoteException
{
  logSecurityInfo("getValue");
  return value;
}
```

Phương thức gọi ngược postCreate đơn giản chỉ gọi logSecurityInfo:

```
public void postCreate(GridContext context) throws
GridServiceException
{
  logSecurityInfo("postCreate");
}
```

Cuối cùng, phương thức logSecurityInfo ghi một số thông tin bảo mật tới log của trình chứa. Như đã đề cập ở trên, phương thức này không có gì liên quan cho tới khi chúng ta chuyển tới các ví dụ bảo mật tiếp theo. Không có gì đặc biệt trong đoạn mã này, ngoại trừ nó ghi ra định danh của người gọi (caller's identity) tới log, như phần in đậm trong đoạn mã dưới đây:

```
private void logSecurityInfo(String methodName)
{
  Subject subject;
  logger.info("SECURITY INFO FOR METHOD '" + methodName + "'");
```

```
    // Print out the caller
    String identity = SecurityManager.getManager().getCaller();
    logger.info("The caller is:" + identity);

    // Print out the caller's subject
    subject = JaasSubject.getCurrentSubject();
    logger.info("INVOCATION SUBJECT");
    logger.info(subject==null?"NULL":subject.toString());

    // Print out service subject
    logger.info("SERVICE SUBJECT");
    subject = SecurityManager.getManager().getServiceSubject(base);
    logger.info(subject==null?"NULL":subject.toString());

    // Print out system subject
    logger.info("SYSTEM SUBJECT");
    try{
      subject = SecurityManager.getManager().getSystemSubject();
      logger.info(subject==null?"NULL":subject.toString());
    }catch(Exception e)
    {
        logger.warn("Unable to obtain service subject");
    }
}
```

Như vậy, ta có thể thấy cài đặt bảo mật cho ứng dụng lưới không hề ảnh hưởng gì mã nguồn phía sever, ít nhất là trong ví dụ này. Tuy nhiên, đối với các kịch bản bảo mật phức tạp, đòi hỏi ta phải thêm một số thay đổi mã nguồn phía server.

Các tham số mô tả triển khai dịch vụ

Để thông báo cho trình chứa biết là dịch vụ của chúng ta sắp hỗ trợ bảo mật, ta cần thêm các tham số sau vào mô tả:

File cấu hình bảo mật (tham số securityConfig)

Phương thức chứng thực được sử dụng (tham số authorization)

Tham số securityConfig

Tham số này cho biết nơi ta có thể tìm thấy file cấu hình bảo mật (security configuration file). Đây là một file XML, bao gồm chi tiết các cấu hình bảo mật, như là mức bảo mật đòi hỏi khi ta triệu gọi phương thức. Ở ví dụ này, để đơn giản, ta sử dụng file cấu hình bảo mật mặc định của GT3. File cấu hình mặc định này sẽ đưa ra các cấu hình bảo mật cần thiết cho dịch vụ. Chúng ta sẽ thảo luận kĩ hơn về file cấu hình này trong phần tiếp theo.

Để thông báo với trình chứa là ta sử dụng file cấu hình bảo mật mặc định, cần thêm những dòng sau vào file mô tả triển khai server-deploy.wsdd:

```
<parameter name="securityConfig"
  value="org/globus/ogsa/impl/security/descriptor/gsi-security-
config.xml"/>
```

Tham số authorization

Tham số này cho phép ta xác định kiểu chứng thực GSI phía server được dùng trong dịch vụ này. Hiện tại, ta sử dụng kiểu chứng thực đơn giản nhất: no authorization. Các kiểu chứng thực khác sẽ được thảo luận ở các phần sau.

Để sử dụng no authorization, ta cần thêm các dòng sau vào mô tả triển khai:

```
<parameter name="authorization" value="none"/>
```

File mô tả triển khai đầy đủ

```
<?xml version="1.0"?>
<deployment                                    name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service              name="progtutorial/security/first/MathService"
provider="Handler" style="wrapped">
    <parameter name="name" value="Secure MathService"/>
    <parameter                                          name="schemaPath"
value="schema/progtutorial/MathService/Math_service.wsdl"/>
    <parameter                                           name="className"
value="org.globus.progtutorial.stubs.MathService.MathPortType"/>
    <parameter name="operationProviders"

value="org.globus.progtutorial.services.security.first.impl.MathProv
ider"/>
```

```
    <parameter                                          name="baseClassName"
value="org.globus.ogsa.impl.ogsi.GridServiceImpl"/>


    <parameter name="securityConfig"
      value="org/globus/ogsa/impl/security/descriptor/gsi-security-
config.xml"/>
    <parameter name="authorization" value="none"/>


    <!-- Start common parameters -->
    <parameter name="allowedMethods" value="*"/>
    <parameter name="persistent" value="true"/>
    <parameter                                          name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>
  </service>
</deployment>
```

- Phía client

Client được sử dụng để triệu gọi dịch vụ bảo mật này sẽ là client mà ta đã sử dụng trong các ví dụ trước. Chú ý là trước khi sử dụng, bạn phải dùng grid-proxy-init để khởi tạo một giấy ủy quyền cho client.

Để cấu hình cho client triệu gọi được các phương thức bảo mật cơ bản này, cần thêm vào các dòng sau:

```
((Stub)math)._setProperty(Constants.GSI_SEC_CONV,Constants.ENCRYPTIO
N);
((Stub)math)._setProperty(Constants.AUTHORIZATION,NoAuthorization.ge
tInstance());
ProxyPolicyHandler          grimPolicyHandler          =          new
IgnoreProxyPolicyHandler();
```

Các dòng mã này sẽ cấu hình lớp stub(môt đối tượng MathPortType) để sử dụng bảo mật. Cụ thể chúng thực hiện như sau:

Dòng đầu cho biết stub sử dụng mức bảo mật là mã hóa (full-blown encryption). GSI cho phép chúng ta lựa chọn các mức bảo mật khác nhau, mã hóa chỉ là một trong số chúng. Ví dụ, ta có thể lựa chọn chữ kí điện tử để bảo vệ tính toàn vẹn của thông điệp. Cụ thể về các mức này, thảo luận ở các phần tiếp theo.

Dòng thứ hai cho biết stub sử dụng cơ chế chứng thực: no authorization ở phía client. Chú ý là các chứng thực ở phía client và server là khác nhau. Nếu bạn chưa rõ, quay trở lại phần giới thiệu về GSI.

Dòng thứ ba là thiết lập GRIM_POLICY_HANDLER cho stub. Ở đây, ta sử dụng là "ignore" handle, tức là lờ đi cơ chế quản lý của GRIM.

Chú ý: Do giấy chứng nhận GRIM không phải chuẩn X509, cho nên nếu không có dòng 3, bạn sẽ gặp lỗi sau:

org.globus.gsi.proxy.ProxyPathValidatorException: Unknown policy: 1.3.6.1.4.1.3536.1.1.1.7

Chi tiết để khắc phục dòng lỗi này, bạn có thể xem phần FAQ, hoặc phần các cơ chế quản lý GRIM.

- Triển khai và thực thi dịch vụ.

Triển khai dịch vụ ở tài khoản globus:

```
ant deploy \ -
Dgar.name=$TUTORIAL_DIR/build/lib/org_globus_progtutorial_services_s
ecurity_first.gar
```

Kích hoạt logging

Khi sử dụng lớp logging phía dịch vụ, chúng ta cần kích hoạt logging cho dịch vụ của chúng ta. Thêm dòng sau vào cuối file $GLOBUS_LOCATION/ogsilogging.properties:

```
org.globus.progtutorial.services.security.first.impl.MathProvider=co
nsole,info
```

Khởi tạo trình chứa

Trước khi khởi tạo trình chứa, bảo đảm rằng bạn đã có một giấy ủy quyền cho tài khoản globus. Sau khi đã có giấy ủy quyền, khởi tạo trình chứa trên tài khoản người dùng globus:

```
globus-start-container
```

Thực thi phía client

Trước khi thực hiện bất cứ một ứng dụng client nào, chú ý là ta đã có một giấy ủy quyền cho tài khoản người dùng gc_user.

Bây giờ, thực hiện client:

```
java \
-classpath ./build/classes/:$CLASSPATH \
org/globus/progtutorial/clients/MathService/ClientGSIConvEncrypt \
http://127.0.0.1:8080/ogsa/services/progtutorial/security/first/Math
Service \
5
```

Nếu mọi chuyện tốt đẹp, bạn sẽ nhìn thấy ở phía client:

```
Added 5

Subtracted 1

Current value: 4
```

Những thông tin sau ở phía server

```
INFO: SECURITY INFO FOR METHOD 'add'

INFO: The caller is:/O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor


INFO: INVOCATION SUBJECT

INFO: Subject:

        Principal:      /O=Globus/OU=GT3      Tutorial/CN=Globus      3
Administrator

        Private                                                 credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@ae1393


INFO: SERVICE SUBJECT

INFO: NULL


INFO: SYSTEM SUBJECT

INFO: Subject:

        Principal:      /O=Globus/OU=GT3      Tutorial/CN=Globus      3
Administrator

        Private                                                 credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@ae1393
```

Chú ý: định danh của người gọi trong giấy chứng nhận là gc_user, và định danh của hệ thống phụ thuộc vào định danh trong giấy chứng nhận của người dùng globus.

2.4.3. Các bước để viết một dịch vụ bảo mật lưới

Sau khi thực hiện thành công dịch vụ bảo mật đầu tiên của mình, chắc hẳn bạn sẽ muốn tự mình viết thêm nhiều cơ chế bảo mật khác nhau cho dịch vụ.

Dưới đây sẽ là các bước cơ bản nhất để bạn viết được một dịch vụ bảo mật. Chúng tôi sẽ sử dụng ví dụ trên để minh họa cho các bước.

Chú ý: trước khi thực hiện những bước này, bảo đảm cơ sở hạ tầng bảo mật GSI của bạn đã cấu hình đúng, các giấy chứng nhận của host, người dùng globus và gc_user đã được ký bởi một nhà CA trong lưới.

- Lập trình bảo mật phía server.

Để dịch vụ của bạn cung cấp các tính năng bảo mật, bạn cần phải thực hiện những bước sau ở phía server:

Bước 1. Xác định file cấu hình bảo mật cho dịch vụ của bạn (mặc định hoặc bạn tự viết): ở ví dụ trên, ta đã sử dụng file cấu hình mặc định của hệ thống (gsi-security-config.xml). Trong các tiếp theo, bạn sẽ biết thay đổi các cấu hình bảo mật cho dịch vụ cho riêng mình khi viết các file cấu hình này.

Bước 2. Viết file mô tả triển khai wsdd: bước này là rất quan trọng, chứa các thuộc tính bảo mật thiết yếu cho dịch vụ. Mô tả chi tiết sẽ được trình bày ở các tiếp theo. Trong ví dụ trên, các tham số đáng chú ý:

File cấu hình bảo mật (tham số securityConfig)

Phương thức chứng thực được sử dụng (tham số authorization)

Bước 3. Cài đặt mã nguồn cho dịch vụ để hỗ trợ bảo mật nếu cần thiết: ở ví dụ này, do quá đơn giản nên không cài đặt bảo mật trong file thực thi của dịch vụ. Đáng chú ý là đoạn mã lấy định danh của người sử dụng dịch vụ:

```
// Print out the caller
String identity = SecurityManager.getManager().getCaller();
logger.info("The caller is:" + identity);
```

- Lập trình bảo mật phía client.

Để phía client hỗ trợ bảo mật, bạn phải xác định các cơ chế bảo mật thích hợp, bao gồm các bước sau:

Bước 4. Lựa chọn mức bảo mật cho client: ở ví dụ trên, bạn đã lựa chọn mức bảo mật mã hóa cho client thông qua dòng lệnh

```
((Stub)math)._setProperty(Constants.GSI_SEC_CONV,Constants.ENCRYPTIO
N);
```

Bước 5. Lựa chọn cơ chế chứng thực cho client: client được thiết lập cơ chế chứng thực đơn giản nhất: no authorization

```
((Stub)math)._setProperty(Constants.AUTHORIZATION,NoAuthorization.ge
tInstance());
```

Bước 6. Lựa chọn cơ chế quản lý GRIM: cơ chế ủy nhiệm GRIM được lờ đi trong ví dụ này

```
ProxyPolicyHandler grimPolicyHandler = new
IgnoreProxyPolicyHandler();
```

Sau khi đã thực hiện đủ 6 bước trên, bạn sẽ triển khai dịch vụ và thực hiện client để triệu gọi dịch vụ. Cách thức triển khai và thực thi, bạn có thể xem lại ví dụ trên. Tiếp theo, chúng tôi sẽ đi vào chi tiết các bước trong quy trình viết dịch vụ bảo mật cho lưới. Các ví dụ minh họa cho các phần tiếp theo cũng được thực hiện theo sườn của 6 bước trên để các bạn tiện theo dõi.

2.4.3.1. Viết file cấu hình bảo mật riêng

Trong ví dụ bảo mật đầu tiên sử dụng cơ chế bảo mật khá đơn giản: mọi thứ đều được mã hóa, không có chứng thực được đưa ra. Trong phần này, chúng ta sẽ đề cập tới những cơ chế phức tạp hơn. Ví dụ, chúng ta có thể xác định một số phương thức của dịch vụ được bảo mật khi triệu gọi, trong khi đó một số phương thức khác lại không cần bảo mật chút nào cả. Điều này được thực hiện bằng cách tạo ra file cấu hình bảo mật (security configuration) của riêng bạn.

- Giới thiệu

Trong phần trước, ta đã sử dụng file cấu hình bảo mật mặc định, mọi thứ đều được bảo mật khi triệu gọi:

```
<service                name="progtutorial/security/first/MathService"
provider="Handler" style="wrapped">


<-- ... -->


  <parameter name="securityConfig"
    value="org/globus/ogsa/impl/security/descriptor/gsi-security-
config.xml"/>


  <-- ... -->


<service>
```

File (gsi-security-config.xml) đi kèm trong gói Globus Toilkit. Tuy nhiên, để có thể thêm quyền điều khiển ở phía dịch vụ (hơn là mọi thứ đều được bảo mật), chúng ta sẽ tự viết file cấu hình cho riêng mình. Trong file cấu hình này, ta có thể điều khiển các khía cạnh bảo mật dựa trên mỗi phương thức sau đây:

Phương thức chứng thực (Authentication method): Chúng ta có thể xác định phương thức chứng thực nào được sử dụng khi client triệu gọi phương thức. Ví dụ, ta có thể xác định phương thức add được triệu gọi với cơ chế mã hóa đầy đủ, trong khi đó, phương thức subtract chỉ đơn giản sử dụng chữ kí điện tử khi được triệu gọi (bảo đảm tính toàn vẹn, nhưng không có tính bí mật). Ta cũng có thể xác định các phương thức khác hoàn toàn không có bảo mật nào cả.

Định danh thời gian thực thi (Runtime identity): Mỗi dịch vụ luôn chạy dưới một định danh xác định. dịch vụ phải chạy dưới định danh nào, mặc dù ứng dụng thực tiễn của tính năng này chỉ phát huy khi chúng ta sử dụng ủy nhiệm.

Trong phần này, ta sẽ viết hai file cấu hình bảo mật minh họa cho hai khía cạnh trên, một file sẽ thể hiện các phương thức chứng thực, mà file kia thể hiện các định danh thời gian thực

thi. Ta sẽ viết hai dịch vụ để mô tả cho các file cấu hình này, thử nghiệm chúng trên các client khác nhau để xem kết quả như thế nào. File đặc tả giao diện GWSDL không cần phải viết mới, sử dụng như ở ví dụ đầu tiên:

- Tùy chọn các phương thức chứng thực

File cấu hình là một file XML đơn giản, với phần tử root như sau:

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

xmlns:math="http://www.globus.org/namespaces/2004/02/progtutorial/Ma
thService">

  <-- ... -->

<securityConfig>
```

Chú ý là ta phải khai báo không gian tên của dịch vụ (math). Điều này là cần thiết khi ta tham chiếu tới các phương thức của dịch vụ (file cấu hình bảo mật phải nhận biết được nó phục vụ cho dịch vụ nào).

Phần tử <securityConfig> bao gồm vài phần tử <method> . Mỗi một phần tử <method> sẽ cho ta cấu hình các lựa chọn bảo mật cho một phương thức riêng biệt. Bây giờ, ta sẽ hiệu chỉnh các phương thức chứng thực của 3 phương thực trong dịch vụ MathService của chúng ta: add, subtract, và getValue.

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

xmlns:math="http://www.globus.org/namespaces/2004/02/progtutorial/Ma
thService">

  <method name="math:add">
    <-- ... -->
  </method>

  <method name="math:subtract">
   . <-- ... -->
  </method>

  <method name="math:getValue">
```

```
    <-- ... -->
  </method>


</securityConfig>
```

Chú ý: Tên của mỗi phương thức trong file cấu hình có tiền tố là không gian tên của dịch vụ.

Mỗi phần tử <method> chứa một tập các thẻ để cấu hình cho dịch vụ đó. Để thay đổi các phương thức chứng thực, ta cần thay đổi phần tử <auth-method> bên trong phần tử <method>. Phần tử này có thể chứa các phần tử XML sau đây:

<none>: phương thức được triệu gọi không có bảo mật.

<pkey>: Sử dụng chứng thực khóa công khai (public-key). Trong GT3.2, loại chứng thực này gọi là bảo mật các thông điệp (GSI Secure Message).

<gsi>: Đây là mức bảo mật các hội thoại (GSI Secure Conversation).

Cả 3 phần tử trên đều có thể được sử dụng như các phần tử trống (<none/>, <pkey/>, <gsi/>). Tuy nhiên, phần tử <gsi> có thể chứa các phần tử khác để có thêm các tùy chọn cấu hình cho mức bảo mật hội thoại GSI.

Trong ví dụ này, ta chỉ sử dụng các phần tử chứng thực <none> và <gsi>. Phần tử bảo mật thông điệp - GSI Secure Message (<pkey>) cũng sử dụng cho bảo mật hội thoại, nhưng có ít tính năng hơn phần tử bảo mật hội thoại <gsi> (GSI Secure Conversation). Ví dụ, nó không cung cấp mã hóa (encryption)và ủy nhiệm (delegation). Tuy nhiên, nó lại có tốc độ thực hiện nhanh hơn, cho nên có thể chấp nhận được nếu hiệu suất thực hiện là quan trọng.

+ Không có chứng thực (No authentication)

Chúng ta bắt đầu với ví dụ đơn giản nhất, phương thức getValue không cần bảo mật.

```
<method name="math:getValue">


  <auth-method>
    <none/>
  </auth-method>
</method>
```

+ Chứng thực GSI (GSI authentication)

Phần tử <gsi> có thể chứa một phần tử <protection-level>, cho phép ta xác định mức độ bảo vệ cho cuộc hội thoại: toàn vẹn hay là bí mật, hoặc cả bí mật và toàn vẹn. Để xác định mức độ nào ta muốn, chỉ cần thêm các phần tử trống (<integrity/>, <privacy/>) vào bên trong phần tử <protection-level>:

<integrity/>: Cuộc hội thoại được bảo đảm toàn vẹn bằng cách sử dụng vào chữ kí điện tử. Thông điệp khi đó sẽ không được mã hóa (tính bí mật không được bảo đảm). Điều này nghĩa là client stub phải thiết lập Constants.GSI_SEC_CONV là Constants.SIGNATURE.

<privacy/>: Hội thoại bảo đảm tính bí mật bằng cách mã hóa các thông điệp. Điều này nghĩa là client stub phải thiết lập Constants.GSI_SEC_CONV to là Constants.ENCRYPTION.

Ví dụ dưới đây, tính bí mật được bảo đảm cho phương thức subtract, khi client muốn triệu gọi nó:

```
<method name="math:subtract">
  <auth-method>
    <gsi>
      <protection-level>
        <integrity/>
      </protection-level>
    </gsi>
  </auth-method>
</method>
```

Với cấu hình này, các client triệu gọi subtract chỉ sử dụng mức bảo vệ toàn vẹn, nếu client sử dụng mã hóa, sẽ không gọi được subtract. Để cho phép cả tính toàn vẹn và bí mật, ta thêm cả hai phần tử này vào trong phần tử <protection-level>, như ví dụ về cấu hình phương thức add dưới đây:

```
<method name="math:add">
  <auth-method>
    <gsi>
      <protection-level>
        <integrity/>
      privacy/>
      </protection-level>
    </gsi>
  </auth-method>
</method>
```

Đây là nội dung của toàn bộ file:

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

```
xmlns:math="http://www.globus.org/namespaces/2004/02/progtutorial/MathService">

<method name="math:add">
  <auth-method>
    <gsi>
      <protection-level>
        <integrity/>
        <privacy/>
      </protection-level>
    </gsi>
  </auth-method>
</method>

<method name="math:subtract">
  <auth-method>
    <gsi>
      <protection-level>
        <integrity/>
      </protection-level>
    </gsi>
  </auth-method>
</method>

<method name="math:getValue">
  <auth-method>
    <none/>
  </auth-method>
</method>

<!-- Default for other methods -->
```

```
<auth-method>
  <gsi/>
</auth-method>

</securityConfig>
```

+ File mô tả triển khai wsdd

File này chỉ thay đổi tên các dịch vụ, tham số securityConfig được thiết lập giá trị là đường dẫn tới các file cấu hình vừa viết.

Dưới đây là đoạn mã minh họa một số thay đổi:

```
<service          name="progtutorial/security/first/MathAuthService"
provider="Handler" style="wrapped">

  <-- ... -->

  <parameter name="securityConfig"

value="org/globus/progtutorial/services/security/first/config/securi
ty-config-runas.xml"/>

  <-- ... -->

<service>
<service          name="progtutorial/security/first/MathRunAsService"
provider="Handler" style="wrapped">

  <-- ... -->

  <parameter name="securityConfig"

value="org/globus/progtutorial/services/security/first/config/securi
ty-config-auth.xml"/>
```

```
<-- ... -->

<service>
```

+ Cài đặt dịch vụ

Ta vẫn sử dụng một OperationProvider để cài đặt cho dịch vụ. Mã nguồn không có gì thay đổi so với ví dụ trước.

+ Biên dịch và triển khai

+ Viết mã phía Client

Chúng ta sẽ triệu gọi các phương thức trong dịch vụ này với 3 client khác nhau. Điều này cho phép ta biết được, server sẽ từ chối client như thế nào nếu không đáp ứng được các đặc tả trong file cấu hình khi client triệu gọi phương thức từ xa. Cụ thể, các client là:

Encryption Client : được cấu hình để yêu cầu một hội thoại được mã hóa (tính bí mật).

Signed Client : được cáu hình để yêu cầu một hội thoại được ký (tính toàn vẹn)

No Security Client : được cấu hình để yêu cầu hội thoại không cần bảo mật.

Client ClientGSIConvEncrypt đã sử dụng trong ví dụ trước. Hai client khác (ClientGSIConvSigned và ClientNoSecurity) cũng tương tự thế, chỉ phân biệt ở thuộc tích stub được thiết lập cho cấu hình bảo mật.

_ Client mã hóa (Encryption client)

Cấu hình thuộc tính stub như sau:

```
((Stub)math)._setProperty(Constants.GSI_SEC_CONV,Constants.ENCRYPTIO
N);
((Stub)math)._setProperty(Constants.AUTHORIZATION,NoAuthorization.ge
tInstance());
```

Hãy chạy thử nó:

```
java \
-classpath ./build/classes/:$CLASSPATH \
org/globus/progtutorial/clients/MathService/ClientGSIConvEncrypt \
http://127.0.0.1:8080/ogsa/services/progtutorial/security/first/Math
AuthService \
5
```

Bạn thấy đầu ra như sau:

```
Added 5
```

```
ERROR: GSI Secure Conversation (signature only) authentication
required for
"{http://www.globus.org/namespaces/2004/02/progtutorial/MathService}
subtract" operation.


Current value: 5
```

Bây giờ, thử theo dõi những gì xảy ra:

Ta triệu gọi được phương thức add vì nó cho phép cuộc hội thoại được mã hóa và ký.

Ta không triệu gọi được subtract bởi nó chỉ cho phép cuộc hội thoại được ký.

Không có vấn đề gì với getValue vì không có bảo mật nào được đòi hỏi.

_ Client được ký (Signed Client)

Client này khởi tạo triệu gọi bảo mật chỉ sử dụng chữ ký điện tử, nghĩa là bảo đảm tính toàn vẹn, nhưng không bí mật. Để thực hiện, ta thiết lập thuộc tính của stub như sau:

```
((Stub)math)._setProperty(Constants.GSI_SEC_CONV,Constants.SIGNATURE
);
((Stub)math)._setProperty(Constants.AUTHORIZATION,NoAuthorization.ge
tInstance());
```

Biên dịch và triển khai client:

```
javac \
-classpath ./build/classes/:$CLASSPATH \
org/globus/progtutorial/clients/MathService/ClientGSIConvSigned.java
java \
-classpath ./build/classes/:$CLASSPATH \
org/globus/progtutorial/clients/MathService/ClientGSIConvSigned \
http://127.0.0.1:8080/ogsa/services/progtutorial/security/first/Math
AuthService \
5
```

Bạn có thể thấy kết quả như sau:

```
Added 5


Subtracted 1
```

```
Current value: 9
```

Những gì vừa mới xảy ra?

Chúng ta có thể triệu gọi add và subtract bởi các phương thức này đều cho phép các hội thoại được ký.

Không có vấn đề gì với getValue vì không có bảo mật nào được đòi hỏi.

_ Không có bảo mật (No security)

Client này không cần bảo mật một chút gì cả. Chúng ta không phải thiết lập bất kì thuộc tính stub nào của client.Ta có thể biên dịch và triển khai ngay:

```
javac \
-classpath ./build/classes/:$CLASSPATH \
org/globus/progtutorial/clients/MathService/ClientNoSecurity.java
java \
-classpath ./build/classes/:$CLASSPATH \
org/globus/progtutorial/clients/MathService/ClientNoSecurity \
http://127.0.0.1:8080/ogsa/services/progtutorial/security/first/Math
AuthService \
5
```

Bạn có thể thấy kết quả như sau:

```
ERROR: GSI Secure Conversation authentication required for
"{http://www.globus.org/namespaces/2004/02/progtutorial/MathService}
add" operation.


ERROR: GSI Secure Conversation (signature only) authentication
required for
"{http://www.globus.org/namespaces/2004/02/progtutorial/MathService}
subtract" operation.


Current value: 9
```

Cả add và subtract đều không được phép vì chúng yêu cầu một cuộc hội thoại bảo mật. Ta cũng không có vấn đề gì với truy nhập getValue khi mà không có bảo mật nào được đòi hỏi.

- Tùy chọn các định danh (runtime identity)

Phần thứ hai mà ta có thể thay đổi trong file cấu hình là các định danh thời gian thực (runtime identity)của dịch vụ. Điều này cho phép ta điều khiển định danh của dịch vụ trong

khi triệu gọi>. Ứng dụng thực tiễn của ví dụ này có thể bạn chưa thấy ngay, nó sẽ liên quan khi ta thực hiện ủy quyền giấy ủy nhiệm.

Trước tiên, khi triệu gọi một dịch vụ, có ba đối tượng liên quan. Nhớ là, mỗi đối tượng chứa một tên định danh (distinguished name) đặc trưng cho riêng nó.

Chủ thể của hệ thống (System subject): Đây là chủ thể của hệ thống hay của trình chứa. Nếu ta không cấu hình rõ ràng cho trình chứa sử dụng tập các giấy ủy nhiệm khác nhau, chủ thể này sẽ lấy giá trị là định danh của đối tượng đang chạy trên trình chứa. Ví dụ, nếu chúng ta sử dụng tài khoản globus để chạy trình chứa, định danh của hệ thống sẽ là O=Globus, OU=GT3 Tutorial, CN=Globus 3 Administrator.

Chủ thể của của dịch vụ (Service subject): Đây là chủ thể của một dịch vụ riêng biệt. Một trình chứa có thể có nhiều chủ thể dịch vụ khác nhau. Chủ thể này thường là null, nếu ta không xác định giấy ủy nhiệm cho dịch vụ hay thực hiện ủy nhiệm.

Chủ thể của của đối tượng triệu gọi: Chủ thể này phụ thuộc vào định danh thời gian thực được thiết lập trong file cấu hình. Nếu không có file cấu hình bảo mật, chủ thể này sẽ là null.

Thay đổi định danh thời gian thực qua file cấu hình sẽ thay đổi giá trị của định danh của đối tượng triệu gọi. Chúng ta có thể cho định danh này là định danh của một trong ba đối tượng sau:

Định danh của người gọi (caller's identity): Thiết lập định danh triệu gọi trùng với định danh của người gọi (clietn tạo nên cuộc gọi)

Định danh của hệ thống (system's identity): Thiết lập định danh triệu gọi trùng với định danh của hệ thống.

Định danh của hệ thống (service's identity): Thiết lập định danh triệu gọi trùng với định danh của dịch vụ (nếu dịch vụ không có định danh, định danh của hệ thống sẽ được sử dụng).

Để thiết lập chúng, phần tử <method> chứa một phần tử là <run-as>. Phần tử này có thể chứa một phần tử trống <caller-identity/>, <system-identity/>, or <service-identity/>, để xác định thực thể thời gian thực của phương thức.

Ta sẽ cấu hình 3 phương thức của ta với các định danh thời gian thực khác nhau. File cấu hình như dưới đây:

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

xmlns:math="http://www.globus.org/namespaces/2004/02/progtutorial/Ma
thService">

<method name="math:add">
  <run-as>
    <caller-identity/>
  </run-as>
```

```
</method>

<method name="math:subtract">
  <run-as>
    <system-identity/>
  </run-as>
</method>

<method name="math:getValue">
  <run-as>
    <service-identity/>
  </run-as>
</method>

</securityConfig>
```

+ Biên dịch và triển khai

Biên dịch và triển khai như ví dụ trước

+ Phía Client

Ta cần triệu gọi 3 phương thức, theo dõi log phía server để xem định danh của hệ thống, dịch vụ, client trong khi triệu gọi. Ta sử dụng trực tiếp 3 client ở phần trước : $TUTORIAL_DIR/org/globus/progtutorial/clients/MathService/ClientGSIConvEncrypt.java

Thực hiện client

```
java \
-classpath ./build/classes/:$CLASSPATH \
org/globus/progtutorial/clients/MathService/ClientGSIConvEncrypt \
http://127.0.0.1:8080/ogsa/services/progtutorial/security/first/MathRunAsService \
5
```

Ta thử xem những gì xảy ra phía server.

Logs của phương thức add (đang chạy với định danh của người gọi)

Bạn có thể thấy như sau khi phương thức add được triệu gọi:

```
INFO: SECURITY INFO FOR METHOD 'add'
```

```
INFO: The caller is:/O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor

INFO: INVOCATION SUBJECT
INFO: Subject:
        Principal: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
        Private                                          credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@dea768

INFO: SERVICE SUBJECT
INFO: NULL

INFO: SYSTEM SUBJECT
INFO: Subject:
        Principal:     /O=Globus/OU=GT3     Tutorial/CN=Globus     3
Administrator
        Private                                          credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@2d0483
```

Phương thức subtract (chạy với định danh của hệ thống)

Bạn có thể thấy như sau khi phương thức subtract được triệu gọi:

```
INFO: SECURITY INFO FOR METHOD 'subtract'
INFO: The caller is:/O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor

INFO: INVOCATION SUBJECT
INFO: Subject:
        Principal:     /O=Globus/OU=GT3     Tutorial/CN=Globus     3
Administrator
        Private                                          credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@2d0483

INFO: SERVICE SUBJECT
INFO: NULL

INFO: SYSTEM SUBJECT
```

```
INFO: Subject:
        Principal:      /O=Globus/OU=GT3      Tutorial/CN=Globus      3
Administrator
        Private                                                 credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@2d0483
```

Định danh của người gọi được giả định là định danh của hệ thống. Khi trình chứa được chạy bởi tài khoản globus,định danh của đối tượng gọi là định danh của người dùng globus.

Phương thức getValue(chạy với định danh của dịch vụ)

Bạn có thể thấy như sau khi phương thức subtract được triệu gọi:

```
INFO: SECURITY INFO FOR METHOD 'getValue'
INFO: The caller is:/O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor

INFO: INVOCATION SUBJECT
INFO: Subject:
        Principal:      /O=Globus/OU=GT3      Tutorial/CN=Globus      3
Administrator
        Private                                                 credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@2d0483

INFO: SERVICE SUBJECT
INFO: NULL

INFO: SYSTEM SUBJECT
INFO: Subject:
        Principal:      /O=Globus/OU=GT3      Tutorial/CN=Globus      3
Administrator
        Private                                                 credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@2d0483
```

Đinh danh triệu gọi là định danh của dịch vụ. Tuy nhiên, bởi vì dịch vụ không có định danh (NULL), định danh sẽ là định danh của hệ thống (tài khoản globus)

2.4.3.2. Viết file mô tả triển khai (service-deploy.wsdd)

Phần này sẽ hướng dẫn bạn thiết lập các cơ chế bảo mật khi viết mô tả triển khai dịch vụ. Các cơ chế này được đặt ra để hạn chế quyền truy nhập tới dịch vụ. Chỉ có những client nào có đủ thẩm quyền mới có thể sử dụng dịch vụ.

- Cơ chế không chứng thực dịch vụ

Các thuộc tính bảo mật phía service được xác định trong file mô tả triển khai (chú ý là các thay đổi ở WSDL là không cần thiết). File mô tả server-deploy.wsdd chứa một lối vào cho mỗi dịch vụ. Mỗi dịch vụ được mô tả với cơ chế chứng thực khác nhau thông qua tham số "authorization", ở phần <service> trong file mô tả. Các cơ chế chứng thực được cung cấp phía server là: none, seft và gridmap.

Trong trường hợp này nó được thiết lập là none, phía server sẽ không đòi hỏi chứng thực, nghĩa là bất cứ client nào cũng có thể truy nhập và sử dụng dịch vụ, như mô tả dưới đây:

```
<service name="my/service" provider="Handler" style="wrapped">
  ...
  <parameter name="authorization" value="none"/>
  ...
 </service>
```

Các ví dụ trên đều sử dụng cơ chế chứng thực này.

- Cơ chế dịch vụ tự chứng thực

Nếu dịch vụ của bạn dựa trên các thành phần của Globus Toolkit, với phiên bản 3.2.1 hoặc cao hơn, nếu dịch vụ thay mặt người dùng, sử dụng giấy ủy quyền của người dùng để triệu gọi các dịch vụ khác, dịch vụ này phải được chứng thực phải được xác định trước khi ủy quyền. Nếu ủy quyền cho một dịch vụ, mà bạn lại thiết lập cơ chế chứng thực là none, sẽ bị lỗi sau đây:

```
Failure unspecified at GSS-API level (Mechanism level: Cannot
request delegation without authorization (target name null)
```

Khi tham số "authorization" được thiết lập là "seft", chỉ có những người dùng cso định danh giống với định danh của dịch vụ mới được phép truy nhập dịch vụ đó, hay nói cách khác, dịch vụ chỉ chứng thực cho chính mình.

Cơ chế được mô tả như sau:

```
<service name="my/service" provider="Handler" style="wrapped">
  ...
  <parameter name="authorization" value="self"/>
  ...
 </service>
```

- Chứng thực dịch vụ sử dụng gridmap file.

Nếu tham số authorization được thiết lập là gridmap, định danh của người dùng sẽ được xác định trong một file gridmap để chứng thực. Mỗi dịch vụ có một file gripmap riêng, chứa danh

sách của những người được phép truy nhập dịch vụ. Chú ý là gridmap file có thể được cập nhật tự động sử dụng SecurityManager API.

Định dạng của gridmap là rất đơn giản. Nó gồm các dòng, mỗi dòng ứng với một user được truy nhập dịch vụ. Mỗi dòng có hai trường, tách biệt nhau bằng một dấu cách: tên định danh và tên tài khoản. .

Ví dụ tài khoản sau trong gridmap được truy nhập dịch vụ:

```
"/O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor" borja
```

Để thực hiện chứng thực gridmap, ta thêm một thẻ mới gọi là <gridmap> như sau:

```
 <service          name="tutorial/security/gridmap/MathFactoryService"
provider="Handler" style="wrapped">


    <!-- ... -->


    <parameter name="authorization" value="gridmap"/>
    <parameter name="gridmap" value="gridmapfile"/>


    <!-- ... -->


  </service>
```

Khi gridmap trong $GLOBUS_LOCATION, ta có thể chỉ đường dẫn tương đối. Khi nó ở bên ngoài thư mục $GLOBUS_LOCATION, ta phải chỉ ra đường dẫn tuyệt đối.

Gridmap giải quyết vấn đề chứng thực các yêu cầu tới dịch vụ. Đối nới chứng thực đa phương, dịch vụ phải có các giấy ủy nhiệm. Mỗi dịch vụ phải được cấu hình có một tập các giấy ủy nhiệm riêng biệt bằng cách thêm lối vào trong file service-config.wsdd. Ví dụ sau xác định vị trí của proxy trong trường giá trị của tham số "serviceProxy":

```
<parameter                                          name="serviceProxy"
value="/usr/local/dev/globus/tmp/x509up_u80042"/>
```

Tiếp cận này bị hạn chế bởi thời gian sống của proxy, cho nên giải pháp tốt nhất có thể sử dụng giấy chứng thực của dịch vụ. Điều này đươc thực hiện bằng cách cung cấp vị trí của certificate và khóa tương ứng của nó. Ví dụ sau xác định vị trí của giấy chứng nhận dịch vụ và trường giá trị của các tham số "serviceCert" và "serviceKey":

```
<parameter                                          name="serviceCert"
value="/usr/local/dev/globus/services/.globus/servicecert.pem"/>

<parameter                                          name="serviceKey"
value="/usr/local/dev/globus/services/.globus/servicekey.pem"/>
```

Tiếp cận này có thể giải quyết vấn đề với thời gian sống hạn chế của proxy. Tuy nhiên, file khóa (private key) trong mô tả triển khai phải không được mã hóa, điều này dẫn tới một số vấn đề bảo mật đối với nhà quản trị. Chú ý là vị trí của file khóa được xác định trong một thư mục có quyền truy nhập hạn chế. Sự phân quyền file cho phép người dùng có quyền đọc với trình chứa, nhưng hạn chế truy nhập với những người dùng khác.

Cơ chế chứng thực ở trên có thể thiết lập trên dịch vụ (services), xưởng chế tác (factories), hay các thực thể (instances). Ví dụ sau mô tả việc chứng thực gridmap như thế nào để có thể thiết lập được trên xưởng chế tác sử dụng gridmap mặc định. Điều này cho phép người sử dụng được chứng thực có thể truy nhập tới xưởng chế tác, nhưng chỉ có người dùng mà tạo ra một thực thể mới được phép thực hiện phương thức trên thực thể đó.

```
<parameter name="authorization" value="gridmap"/>

<parameter name="instance-authorization" value="self"/>

<parameter name="serviceProxy" value="/tmp/x509up_u800042"/>
```

+ Sử dụng thực thể gridmap phía dịch vụ

Một cách tiếp cận khác có khả năng linh động hơn là xác định gridmap file cho mỗi thực thể của dịch vụ để sử dụng chứng thực. Gridmap file này chứa một tập con người dùng của gridmap file toàn thể, có nội dung hạn chế truy nhập tới một dịch vụ riêng biệt.

```
<parameter name="instance-authorization" value="gridmap"/>

<parameter                                    name="instance-gridmap"
value="/home/globus/gridAdmin/myService/gridmap"/>
```

Giấy chứng nhận của trình chứa có thể xác định trong khối <globalConfiguration> của mô tả triển khai bằng cách thêm tham số containerProxy hay tham số containerCert và containerKey. Mỗi dịch vụ trước tiên sẽ kiểm tra các tham số credential trong phần <service>. Nếu không có tham số credential nào cả, dịch vụ sẽ dựa trên thư viện bảo mật để lấy giấy ủy nhiệm. Trong trường hợp này, thư viện bảo mật sẽ cố gắng để giấy chứng nhận proxy của người dùng đang chạy trên trình chứa. Nếu các giấy ủy nhiệm thay đổi trong khi trình chứa và dịch vụ đang chạy, chúng sẽ tự động được tải lại.

2.4.3.3. Cài đặt dịch vụ

Ngoài việc định nghĩa các tham số trong file mô tả triển khai, bạn còn có một số tùy chọn khi cài đặt dịch vụ. Đôi khi nó sẽ là rất hữu ích khi bạn hỗ trợ các tính năng bảo mật cho dịch vụ.

- Thiết lập sở hữu dịch vụ với giấy ủy nhiệm của người gọi

Liên kết giấy ủy nhiệm của người dùng với một thực thể dịch vụ sử dụng SecurityManager API có thể được thực hiện như sau:

```
import org.globus.ogsa.impl.security.SecurityManager;

...

public void serviceMethod(...) {

   ...
```

```
    SecurityManager.getManager().setServiceOwnerFromContext(this);
  }
  ...
```

Một kịch bản quen thuộc khác là có các thực thể của dịch vụ được tạo bởi factory sử dụng giấy ủy nhiệm khi triệu gọi toán tử Create. Trong ví dụ này, có thể thực hiện trong phương thức postCreate của thực thể dịch vụ như sau:

```
import org.globus.ogsa.impl.security.SecurityManager;
 ...
 public void postCreate(GridContext context) throws
GridServiceException {
    super.postCreate(context);
    SecurityManager manager = SecurityManager.getManager();
    manager.setServiceOwnerFromContext(this, context);
    ...
 }
 ...
```

và thiết lập định danh thời gian thực của JAAS như phần trước.

- Lấy định danh của người gọi

Để lấy định danh hiện tại của người gọi, sử dụng:

```
import org.globus.ogsa.impl.security.SecurityManager;
 ...
 public void serviceMethod() {
    String identity = SecurityManager.getManager().getCaller();
 }
 ...
```

- Lấy đối tượng JAAS hiện tại

GT3 cho phép các phương thức dịch vụ được triệu gọi sử dụng các chủ thể null, hệ thống/trình chứa hay chủ thể dịch vụ, chủ thể người gọi. Về chi tiết các chủ thể của JAAS, ta đã trình bày ở phần 7. Để lấy chủ thể JAAS Subject, sử dụng:

```
import javax.security.auth.Subject;
 import org.globus.gsi.jaas.JaasSubject;
 ...
 public void serviceMethod() {
```

```
    Subject subject = JaasSubject.getCurrentSubject();
}
...
```

Chủ thể hiện tại phụ thuộc vào chính sách run-as trong file cấu hình bảo. mật. Về chi tiết, xin xem lại phần trước. Nếu không có mô tả bảo mật được thiết lập cho dịch vụ, chủ thể hiện tại sẽ là null.

- Thực hiện công việc như một đối tượng JAAS riêng biệt

Để thực hiện một phần code với một đối tượng Subject được đưa ra, thực hiện như sau:

```
import javax.security.auth.Subject;
import java.security.PrivilegedAction;
import org.globus.gsi.jaas.JaasSubject;
...
public void method() {
  // create a new subject or obtain it from somewhere
  Subject subject = ...;
  JaasSubject.doAs(subject, new PrivilegedAction() {
                              public Object run() {
                                // do work here
                                return null;
                              }
                   });
}
...
```

2.4.3.4. Các cơ chế bảo mật phía Client.

Đối với bảo mật phía server, các cơ chế bảo mật được kích hoạt tại thời điểm triển khai, nhưng đối với phía client, cần phải có những thay đổi về mã nguồn. Các thay đổi này được thực hiện bằng cách thiết lập các thuộc tính của stub client.

- Bảo mật hội thoại (GSI Secure Conversation)

Để sử dụng bảo mật hội thoại, bạn cần thêm gói sau vào

```
import org.globus.ogsa.impl.security.Constants;
```

Đối với mã hóa, thiết lập:

```
stub._setProperty(Constants.GSI_SEC_CONV,Constants.ENCRYPTION);
```

Đối với toàn vẹn, thiết lập:

```
stub._setProperty(Constants.GSI_SEC_CONV,Constants.SIGNATURE);
```

- Bảo mật thông điệp (GSI Secure Message)

Để sử dụng bảo mật thông điệp, bạn cần thêm gói sau vào

```
import org.globus.ogsa.impl.security.Constants;
```

Để sử dụng chứng thực khóa công khai (<pkey> trong file cấu hình bảo mật), hãy còn gọi là bảo mật thông điệp:

```
stub._setProperty(Constants.GSI_XML_SIGNATURE,Boolean.TRUE);
```

Để bỏ kích hoạt nó, sử dụng:

```
stub._setProperty(Constants.GSI_XML_SIGNATURE,Boolean.FALSE);
```

- Chứng thực

+ Để sử dụng không chứng thực:

```
stub._setProperty(Constants.AUTHORIZATION,NoAuthorization.getInstanc
e());
```

Và thêm gói NoAuthorization:

```
import org.globus.ogsa.impl.security.authorization.NoAuthorization;
```

+ Để sử dụng tự chứng thực (seft):

```
stub._setProperty(Constants.AUTHORIZATION,SelfAuthorization.getInsta
nce());
```

Và thêm gói SeftAuthorization:

```
import
org.globus.ogsa.impl.security.authorization.SeftAuthorization;
```

+ Để sử dụng chứng thực host:

```
stub._setProperty(Constants.AUTHORIZATION,HostAuthorization.getInsta
nce());
```

Và thêm gói HostAuthorization:

```
import
org.globus.ogsa.impl.security.authorization.HostAuthorization;
```

+ Đối với chứng thực định danh

Chứng thực định danh có thể được sử dụng để chứng thực bất cứ định danh nào. Để thiết lập chứng thực định danh, tạo một đối tượng IdentityAuthorization mới và sử dụng nó khi thiết lập các thuộc tính của chứng thực như sau:

```
IdentityAuthorization authz = new
IdentityAuthorization("/C=US/O=UTAustin/OU=TACC/CN=Jeff R.
Mausolf/UID=mausolf");


gridFactory.getStub()._setProperty(Constants.AUTHORIZATION, authz);
```

và thêm vào gói Identity Authorization:

```
import
org.globus.ogsa.impl.security.authorization.IdentityAuthorization;
```

- Ủy nhiệm

Để thực hiện ủy nhiệm cho client, trước hết import gói sau vào:

```
import org.globus.axis.gsi.GSIConstants;
```

Sau đó, thiết lập các thuộc tính của stub.

Để thực hiện không ủy nhiệm:

```
stub._setProperty(GSIConstants.GSI_MODE,GSIConstants.GSI_MODE_NO_DEL
EG);
```

Để thực hiện ủy nhiệm đầy đủ:

```
stub._setProperty(GSIConstants.GSI_MODE,GSIConstants.GSI_MODE_FULL_D
ELEG);
```

Để thực hiện ủy nhiệm hạn chế:

```
stub._setProperty(GSIConstants.GSI_MODE,GSIConstants.GSI_MODE_LIMITE
D_DELEG);
```

2.4.4. Quản lý giấy chứng nhận GRIM.

Giấy ủy quyền GRIM (Ánh xạ thực thể tài nguyên lưới - Grid Resource Identity Mapper) là giấy chứa các chính sách ủy quyền để liệt kê các thực thể (tên định danh/tên đối tượng) lưu giữ một phần khóa bí mật của GRIM. Kiến trúc GRIM cho phép cơ chế ủy quyền truy cập tài nguyên (e.g. host). Điều này được thực hiện thông qua một chương trình setuid, có quyền truy nhập giấy ủy nhiệm dài hạn của tài nguyên, và chương trình tạo ra một giấy ủy quyền GRIM chứa danh sách của các thực thể. Danh sách các thực thể đưa ra bao gồm ánh xạ <định danh> <người dùng> được chứa trong grid-mapfile, để xem người dùng có hợp lệ trên tài nguyên hay không. Thông tin về GRIM có thể tìm ở địa chỉ:

http://www.globus.org/security/gsi3/GRIM-05.doc

Loại giấy ủy quyền này thường được sử dụng bởi GRAM, nhưng trong tương lai, nó sẽ được sử dụng trong các dịch vụ cấp cao hơn.

- Cài đặt phía Client.

Khi GRIM chứa các chính sách ủy quyền không theo tiêu chuẩn, nó cần một bộ quản lý đặc biệt trong khi chứng thực. Bộ quản lý đặc biệt này sẽ cài đặt một thực thi để giải quyết vấn đề ủy quyền GRIM. Hiện tại, có hai quản lý chính sách GRIM, một loại bảo đảm một trong các thực thể trong danh sách các thực thể là khớp với thực thể được đưa ra, và loại kia lờ đi nội dung các chính sách của GRIM. Chúng ta không nên dùng loại thứ hai, khi nó làm hỏng chính sách bảo mật mà GRIM đưa ra. Để sử dụng loại thực thi đầu tiên, làm như sau:

```
import org.globus.gsi.proxy.ProxyPolicyHandler;
import org.globus.ogsa.handlers.GrimProxyPolicyHandler;
...
String identity = "identity you expect to see in GRIM policy"
ProxyPolicyHandler          grimPolicyHandler          =          new
GrimProxyPolicyHandler(identity);
stub._setProperty(Constants.GRIM_POLICY_HANDLER, grimPolicyHandler);
```

Ở đây, "stub" là đối tượng RPC stub như thường lệ. Bạn có thể lựa chọn bỏ qua thực thể mà hàm khởi tạo của GrimProxyPolicyHandler đưa ra, khi đó thực thể của giấy ủy quyền hiện tại sẽ được sử dụng. Để sử dụng lờ đi các thực thi của GRIM, thực hiện như sau:

```
import org.globus.gsi.proxy.ProxyPolicyHandler;
import org.globus.gsi.proxy.IgnoreProxyPolicyHandler;
...
ProxyPolicyHandler          grimPolicyHandler          =          new
IgnoreProxyPolicyHandler();
stub._setProperty(Constants.GRIM_POLICY_HANDLER, grimPolicyHandler);
```

- Cài đặt phía Server.

Nếu bạn muốn dịch vụ của bạn chấp nhận chính sách của GRIM, thực hiện như sau:

```
import org.globus.gsi.proxy.IgnoreProxyPolicyHandler;
import
org.globus.ogsa.impl.security.authentication.SecureServiceProperties
Helper;
...
public class OgsiManagement extends GridServiceImpl
{
    ...
    public OgsiManagement()
    {
        super("Management Service");
        SecureServicePropertiesHelper.setGrimProxyPolicyHandler(
                        this, new IgnoreProxyPolicyHandler());
    }
    ...
}
```

Đoạn mã này sẽ lờ đi chính sách của GRIM trong giấy ủy nhiệm mà nó chứng thực. Bạn cũng có thể ép buộc chính sách GRIM trong trường hợp sử dụng GrimProxyPolicyHandler (e.g. lập trình phía client) thay vì IgnoreProxyPolicyHandler.
Lưu ý rằng chứng thực định danh của giấy ủy quyền GRIM là khác so với chứng thực giấy ủy quyền thông thường. Thực thể chứng thực của GRIM không thêm vào bất kì thành phần common name (CN=) nào khi sinh giấy ủy quyền.
Bạn cũng có thể sử dụng giấy ủy quyền GRIM cho ủy nhiệm trình chứa của bạn, bạn phải bảo đảm rằng đã cài đặt GrimContainerHandler (một phần của GRAM), để tạo ra giấy ủy quyền GRIM và làm tươi lại nó sau một khoảng thời gian. Sự cập nhật đòi hỏi khi giấy ủy quyền GRIM có cấu hình vòng đời có giới hạn.

2.4.5. Ủy nhiệm trong dịch vụ lưới.

Trong phần này, chúngt a sẽ có hai ví dụ về ủy nhiệm. Trước khi thử nghiệm những ví dụ này, đây là khoảng thời gian tốt cho bạn xem lại lý thuyết về ủy nhiệm.

Ví dụ thứ nhất rất đơn giản, chỉ là hiệu chỉnh một chút của ví dụ đầu tiên về bảo mật của chúng ta đã biết.

Ví dụ này cho phép ta thực hiện những cơ chế cơ bản trong ủy nhiệm, sử dụng log phía server, để theo dõi chi tiết khi ủy nhiệm.

Tuy nhiên, ví dụ thứ nhất không thể hiện được hết các tính năng của ủy nhiệm, mà ta sẽ thấy rõ hơn ở ví dụ thứ hai. Ví dụ thứ hai gồm hai dịch vụ. Client sẽ ủy quyền giấy ủy nhiệm của nó cho dịch vụ thứ nhất, và dịch vụ thứ nhất sẽ sử dụng giấy ủy nhiệm này để triệu gọi các phương thức ơ dịch vụ thứ hai. Chúng ta có thể theo dõi khi nào thì việc ủy nhiệm được kích hoạt, khi nào ví dụ của ta không hoạt động (bởi vì dịch vụ thứ hai yêu cầu giấy ủy nhiệm của người dùng, không phải là giấy ủy nhiệm của dịch vụ thứ nhất). Và khi ủy nhiệm được kích hoạt, ta sẽ theo dõi log xem chúng hoạt động ra sao.

- Bước đầu tiếp cận với ủy nhiệm

Như đã đề cập ở trên, ví dụ đầu tiên dựa trên dịch vụ bảo mật đầu tiên mà ta đã triên khai.

+ Kích hoạt ủy nhiệm phía client

Việc đầu tiên là thay đổi client để cho phép nó ủy nhiệm tới dịch vụ của nó. Điều này được thực hiện đơn giản bằng cách thiết lập các thuộc tính sau của stub:

```
((Stub)math)._setProperty(GSIConstants.GSI_MODE,GSIConstants.GSI_MOD
E_FULL_DELEG);
```

+ Kích hoạt ủy nhiệm phía server

Để cho việc ủy nhiệm được hoàn thiện, ta cần thực hiện hai bước sau ở phía server:

Mỗi phương thức ta muốn triệu gọi phải được cấu hình để chạy được với định danh của người triệu gọi. Hay nói cách khác, chủy thể triệu gọi phải được thiết lập với định danh của người gọi. Ta đã thực hiện điều này trước đó với phần định danh thời gian thực (runtime identity) của phần file cấu hình bảo mật.

Chúng ta cũng phải cho dịch vụ biết định danh của người gọi. Nhớ là trong phần định danh thời gian thực, chủ thể dịch vụ luôn luôn là NULL, nếu chúng ta không thực hiện uỷ quyền trên dịch vụ. Ta có thể làm điều này bằng cách thêm một dòng code đơn giản.

+ Thiết lập định danh thời gian thực

Việc thiết lập thời gian thực tôi đã giới thiệu ở phần trước của báo cáo. Thiết lập chủ sở hữu dịch vụ

Để cho dịch vụ lấy chủ thể triệu gọi như là chủ thể của nó, ta phải thêm dòng code này vào mỗi phương thức ta muốn thực hiện ủy nhiệm:

```
SecurityManager.getManager().setServiceOwnerFromContext(base);
```

Ví dụ, đối với phương thức add:

```
public void add(int a) throws RemoteException
, SecurityException
{

  SecurityManager.getManager().setServiceOwnerFromContext(base);
  logSecurityInfo("add");
  value = value + a;
}
```

Biên dịch và triển khai

Triển khai nó ở tài khoản globus:

```
ant deploy \
-
Dgar.name=$TUTORIAL_DIR/build/lib/org_globus_progtutorial_services_s
ecurity_delegation_first.gar
```

Cuối cùng, trước khi khởi động lại trình chứa, ta thêm dòng sau vào file $GLOBUS_LOCATION/ogsilogging.properties:

```
org.globus.progtutorial.services.security.delegation_first.impl.Math
Provider=console,info
```

Biên dịch và thực thi phía client

```
java \
-classpath ./build/classes/:$CLASSPATH \
org/globus/progtutorial/clients/MathService/ClientDelegation \
http://127.0.0.1:8080/ogsa/services/progtutorial/security/delegation
/MathService \
```

5

Đầu ra phía client cũng không có gì đặc biệt. Chúng ta sẽ theo dõi log ở phía server để xem việc ủy nhiệm thực hiện như thế nào. Hãy xem phương thức add (chạy dưới định danh của người gọi):

```
INFO: SECURITY INFO FOR METHOD 'add'
INFO: The caller is:/O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor

INFO: INVOCATION SUBJECT
INFO: Subject:
        Principal: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
        Private                                              credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@f4ca49

INFO: SERVICE SUBJECT
INFO: Subject:
        Principal: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
        Private                                              credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@f4ca49

INFO: SYSTEM SUBJECT
INFO: Subject:
        Principal:      /O=Globus/OU=GT3      Tutorial/CN=Globus      3
Administrator
        Private                                              credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@1f88fbd
```

Chú ý rằng định danh của chủ thể dịch vụ không phải là NULL...nó là định danh của người gọi!

Đối với phương thức subtract và getValue, chủ thể dịch vụ cũng không phải là NULL. Tuy nhiên khi chúng ta chạy dưới định danh của hệ thống, ta sẽ thấy chủ thể của tài khoản globus trong chủ thể dịch vụ.

```
INFO: SECURITY INFO FOR METHOD 'subtract'
INFO: The caller is:/O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
INFO: INVOCATION SUBJECT
INFO: Subject:
```

```
        Principal:      /O=Globus/OU=GT3      Tutorial/CN=Globus      3
Administrator
        Private                                                credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@1f88fbd


INFO: SERVICE SUBJECT
INFO: Subject:
        Principal:      /O=Globus/OU=GT3      Tutorial/CN=Globus      3
Administrator
        Private                                                credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@1f88fbd


INFO: SYSTEM SUBJECT
INFO: Subject:
        Principal:      /O=Globus/OU=GT3      Tutorial/CN=Globus      3
Administrator
        Private                                                credential:
org.globus.gsi.gssapi.GlobusGSSCredentialImpl@1f88fbd
```

Ta thấy rằng trong ví dụ này, ủy nhiệm đã thực sự làm việc. Nhưng những gì mà nó thực hiện quả là chưa thuyết phục. Nào, hãy tiếp tục với ví dụ tiếp theo!

- Sử dụng ủy nhiệm để triệu gọi các dịch vụ lẫn nhau

Chúng ta bắt đầu ví dụ này với hai dịch vụ sau:

PhysicsService: Đây là dịch vụ mới mà chương trình sẽ cài đặt.

MathService: Thực tế, đây là dịch vụ MathService với gridmap authorization. Ta không cần thay đổi gì về dịch vụ này cả.

Trong ví dụ của chúng ta, PhysicsService phải chịu trách nhiệm thực hiện những tính toán cực kì phức tạp. Để thực hiện được điều đó, nó cần triệu gọi MathService vài lần. Cụ thể:

Ta sẽ triệu gọi PhysicsService từ tài khoản của người dùng. Trong trường hợp này, tài khoản có một giấy chứng nhận với định danh: O=Globus,OU=GT3 Tutorial,CN=Borja Sotomayor

PhysicsService cho phép bất cứ người dùng nào truy cập các phương thức của nó. Do đó, ứng dụng clietn sẽ không gặp phiền toái gì khi truy cập PhysicsService.

MathService, chỉ cho phép một người dùng truy nhập các phương thức của nó. Chúng ta sử dụng cùng dịch vụ và gridmap trong phần gridmap trước đó, cho nên chỉ có tài khoản gc_user là truy nhập được MathService. Định danh của người dùng trong ví dụ này l O=Globus,OU=GT3 Tutorial,CN=Borja Sotomayor

Điều này có nghĩa là để cho ví dụ hoạt động, PhysicsService phải triệu gọi MathService với định danh sau trong giấy ủy nhiệm: O=Globus,OU=GT3 Tutorial,CN=Borja Sotomayor

Khi PhysicsService đang chạy trong một trình chứa với một tập các giấy ủy nhiệm (kể cả tài khoản globus), PhysicsService chỉ có thể truy cập MathService nếu ứng dụng client (chạy dưới tài khoản người dùng gc_user) ủy quyền giấy ủy nhiệm của nó.

Để kiểm tra tính đúng đắn của ví dụ, ta sẽ thử nghiệm PhysicsService trong hai trường hợp có ủy nhiệm và không có ủy nhiệm.

PhysicsService có một phương thức đơn getAnswerToLifeTheUniverseAndEverything, được triệu gọi bởi ứng dụng client. Phương thức này sẽ triệu gọi phương thức add trong MathService vài lần.

+ PhysicsService

Ta sẽ bắt đầu với dịch vụ Physics, và clietn không thực hiện ủy nhiệm. Sau đó, ta sẽ kích hoạt cơ chế ủy nhiệm cho client, bằng cách chỉ thêm vào một vài dòng code.

_ Giao diện của dịch vụ

Giao diện của dịch vụ PhysicsService rất đơn giản, chỉ có một phương thức.

_ Cài đặt dịch vụ

Cài đặt mã nguồn cho dịch vụ này là tương đối dài, khi ta có một dịch vụ triệu gọi một dịch vụ khác.

Ta sẽ từng bước để theo dõi mã nguồn cài đặt dịch vụ. Trước hết, thay vì cho tất cả mã vào trong phương thức getAnswerToLifeTheUniverseAndEverything, ta sẽ chia nó thành vài phương thức private nhỏ. 'Có thể xem Skeleton' của cài đặt dưới đây:

```
package org.globus.progtutorial.services.security.delegation.impl;

// import statements

public class PhysicsProviderNoDelegation implements
OperationProvider
{
  // Create this class's logger
  static Log logger =
LogFactory.getLog(PhysicsProvider.class.getName());

  // Operation provider properties
  private static final String namespace =
```

```
"http://www.globus.org/namespaces/2004/02/progtutorial/PhysicsServic
e";

  private static final QName[] operations =
    new QName[]
      {new                                    QName(namespace,
"getAnswerToLifeTheUniverseAndEverything")};


  static final String mathFactoryURL =
"http://127.0.0.1:8080/ogsa/services/progtutorial/security/gridmap/M
athService";


  public    int    getAnswerToLifeTheUniverseAndEverything()    throws
RemoteException, SecurityException
  {
  }


  private void makeStubSecure(Object stub)
  {
  }

  private     MathPortType     getReferenceToMathService()     throws
RemoteException
  {
  }

  private void logSecurityInfo()
  {
```

```
    }
}
```

Thử xem những gì các phương thức và thuộc tính đã thực hiện.

Thuộc tính mathFactoryURL

```
static final String mathServiceGSH =
"http://127.0.0.1:8080/ogsa/services/progtutorial/security/gridmap/MathService";
```

Ở đây, ta sử dụng một GSH tĩnh (hard-coded), tuy nhiên trong thực tế không bao giờ thực hiện như vậy. Ta sử dụng như thế cho đơn giản.

getAnswerToLifeTheUniverseAndEverything

Đây là phương thức triệu gọi từ xa có thuộc tính là public. Nó sẽ triệu gọi add trong dịch vụ MathService. Do đó, phần lớn mã của nó được sử dụng để giao tiếp với MathService.

Trước khi bắt đầu, ta cần cặp biến sau đây:

```
MathPortType math;
int answer;
```

Đầu tiên, ta gọi phương thức logSecurityInfo (đầu ra giống như ví dụ trước) để ta kiểm tra xem việc ủy nhiệm có hoạt động hay không:

```
logSecurityInfo();
```

Tiếp đó, ta tham chiếu tới MathPortType.

```
math = getReferenceToMathService();
```

Bây giờ, ta triệu gọi phương thức add trong MathService vài lần, bằng cách thiết lập "answer".

```
for(int i=0; i<7; i++)
{
  logger.info("Invoking 'add' method...");
  math.add(6);
  logger.info("Invoked 'add' method");
}
```

Tiếp, gọi phương thức MathService's getValue để lấy giá trị của "answer"...

```
logger.info("Invoking 'getValue' method...");
answer = math.getValue();
logger.info("Invoked 'getValue' method");
```

Cuối cùng, trả về "answer":

```
return answer;
```

Toàn bộ mọi thứ có thể xem dưới đây:

```
public int getAnswerToLifeTheUniverseAndEverything() throws
RemoteException, SecurityException
{
  MathPortType math;
  int answer;

  logSecurityInfo();

  math = getReferenceToMathService();

  // Find the answer to life, the universe, and everything.
  // This is accomplished by invoking the 'add' method in the
MathService!
  for(int i=0; i<7; i++)
  {
    logger.info("Invoking 'add' method...");
    math.add(6);
    logger.info("Invoked 'add' method");
  }

  logger.info("Invoking 'getValue' method...");
  answer = math.getValue();
  logger.info("Invoked 'getValue' method");

  return answer;
}
```

Phương thức logSecurityInfo

Phương thức logSecurityInfo tương tự như ta đã sử dụng ở các ví dụ trước, với một vài thay đổi nhỏ. Ta sẽ sử dụng nó để ghi ra định danh của người gọi, chủ thể triệu gọi, dịch vụ và hệ thống.

Các phương thức private khác

Các phương thức private còn lại thực hiện nhiều thao tác mà ta thấy ở phía clietn (e.g. lấy tham chiếu của một thực thể, tạo một stub bảo mật...)

Phương thức getReferenceToMathInstance thiết lập một MathPortType stub từ locator.

```
private MathPortType getReferenceToMathService() throws
RemoteException
{
  URL GSH = null;
  try{
   GSH = new java.net.URL(mathServiceGSH);
  } catch(Exception e){ }

  logger.info("Obtaining reference to MathService...");
  MathServiceGridLocator mathLocator = new
MathServiceGridLocator();
  MathPortType math = mathLocator.getMathServicePort(GSH);
  makeStubSecure(math);
  logger.info("Obtained reference to MathService");
  return math;
}
```

Chú ý getReferenceToMathService gọi phương thức makeStubSecure để thiết lập các thuộc tính bảo mật của stub.

```
private void makeStubSecure(Object stub)
{

((Stub)stub)._setProperty(Constants.GSI_SEC_CONV,Constants.ENCRYPTIO
N);

((Stub)stub)._setProperty(Constants.AUTHORIZATION,NoAuthorization.ge
tInstance());
}
```

_ File mô tả triển khai

File WSDD tương tự như file đã sử dụng trong phần file cấu hình bảo mật. File này cũng bao gồm mô tả triển khai cho ví dụ tiếp theo (PhysicsService với ủy nhiệm được kích hoạt)

Tuy nhiên, ta cần tạo một file cấu hình bảo mật để xác định xem các phương thức của dịch vụ chạy dưới định danh nào. Đây là nội dung của nó:

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

xmlns:physics="http://www.globus.org/namespaces/2004/02/progtutorial
/PhysicsService">


<method name="physics:getAnswerToLifeTheUniverseAndEverything">


  <run-as>
    <caller-identity/>
  </run-as>
  <auth-method>
    <gsi/>
  </auth-method>
</method>


<auth-method>
  <gsi/>
</auth-method>


</securityConfig>
```

_ Biên dịch và triển khai

Trước khi triển khai, bạn thêm dòng sau vào file $GLOBUS_LOCATION/ogsilogging.properties

```
org.globus.progtutorial.services.security.delegation.impl.PhysicsNoD
elegationProvider=console,info
```

Triển khai file GAR (từ tài khoản người dùng globus):

```
ant deploy \
```

```
-
Dgar.name=$TUTORIAL_DIR/build/lib/org_globus_progtutorial_services_s
ecurity_delegation.gar
```

+ Client không có ủy nhiệm

Nhớ rằng dịch vụ vẫn chưa hoàn toàn sẵn sàng cho ủy nhiệm. Ta sẽ sử dụng một client không thực hiện ủy nhiệm, để xem MathService từ chối truy cập PhysicsService như thế nào bởi vì nó không sử dụng đúng giấy ủy nhiệm.

Client khá đơn giản, bạn chỉ triệu gọi phương thức getAnswerToLifeTheUniverseAndEverything trong PhysicsService.

```
package org.globus.progtutorial.clients.PhysicsService;


import
org.globus.progtutorial.stubs.PhysicsService.service.PhysicsServiceG
ridLocator;
import org.globus.progtutorial.stubs.PhysicsService.PhysicsPortType;


import org.globus.ogsa.impl.security.Constants;
import org.globus.ogsa.impl.security.authorization.NoAuthorization;


import java.net.URL;
import javax.xml.rpc.Stub;


public class ClientNoDelegation
{
  public static void main(String[] args)
  {
    try
    {
      // Get command-line arguments
      URL GSH = new java.net.URL(args[0]);


      // Get a reference to the MathService instance
      PhysicsServiceGridLocator physicsLocator = new
PhysicsServiceGridLocator();
```

```
      PhysicsPortType                    physics                    =
physicsLocator.getPhysicsServicePort(GSH);


      // Setup security options

((Stub)physics)._setProperty(Constants.GSI_SEC_CONV,Constants.ENCRYP
TION);

((Stub)physics)._setProperty(Constants.AUTHORIZATION,NoAuthorization
.getInstance());


      // Call remote method 'add'
      int                              answer                     =
physics.getAnswerToLifeTheUniverseAndEverything();


      System.out.println("Answer: " + answer);
    }catch(Exception e)
    {
      System.out.println("ERROR:" + e.getMessage());
    }
  }
}
```

Thực hiện client:

```
java \
-classpath ./build/classes/:$CLASSPATH \
org/globus/progtutorial/clients/PhysicsService/ClientNoDelegation \
http://127.0.0.1:8080/ogsa/services/progtutorial/security/delegation
/PhysicsServiceNoDelegation
```

Bạn có thể nhìn thấy lỗi nhỏ sau:

```
org.globus.ogsa.impl.security.authorization.AuthorizationException:
Gridmap authorization failed:
peer "/O=Globus/OU=GT3 Tutorial/CN=Globus 3 Administrator" not in
gridmap file.
```

```
Hãy quan sát kỹ hơn phía server:

INFO: -------- BEGIN SECURITY INFO --------

INFO: Caller: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor

INFO: Invocation subject:/O=Globus/OU=GT3 Tutorial/CN=Borja
Sotomayor

INFO: Service subject:

NULL

INFO: System subject:/O=Globus/OU=GT3 Tutorial/CN=Globus 3
Administrator

INFO: -------- END SECURITY INFO --------
```

Cho dù ta đang chạy dưới định danh của người gọi, chủ thể dịch vụ vẫn là NULL. Khi chủ thể này là NULL, trình chứa sẽ sử dụng chủ thể của dịch vụ (/O=Globus/OU=GT3 Tutorial/CN=Globus 3 Administrator) để triệu gọi MathService. Tuy nhiên, chủ thể đó không có trong gridmap của MathService, và lỗi chứng thực gridmap "Gridmap authorization failed" xảy ra.

Bây giờ, hãy thêm ủy nhiệm cho PhysicsService xem mọi thứ thực hiện như thế nào.

+ Thêm cơ chế ủy nhiệm

Thêm ủy nhiệm trong client

Chúng ta cần làm cho clietn ủy quyền giấy ủy nhiệm của nó. Như đã thấy ở ví dụ trước, ta thực hiện bằng cách thiết lập thuộc tính bảo mật phía stub:

```
((Stub)math)._setProperty(GSIConstants.GSI_MODE,GSIConstants.GSI_MOD
E_FULL_DELEG);
```

Chấp nhận ủy nhiệm phía server

Để dịch vụ sử dụng giấy ủy nhiệm như là chủ sở hữu giấy, ta cần thêm dòng sau vào phương thức getAnswerToLifeTheUniverseAndEverything trước khi gọi logSecurityInfo:

```
SecurityManager.getManager().setServiceOwnerFromContext(base);0
```

Biên dịch và triển khai dịch vụ hoàn toàn giống như trên.

Biên dịch, triển khai và thực hiện client

```
javac \
-classpath ./build/classes/:$CLASSPATH \
org/globus/progtutorial/clients/PhysicsService/ClientDelegation.java
java \
-classpath ./build/classes/:$CLASSPATH \
org/globus/progtutorial/clients/PhysicsService/ClientDelegation \
```

```
http://127.0.0.1:8080/ogsa/services/progtutorial/security/delegation
/PhysicsService
```

Ngay khi thực hiện client, bạn sẽ thấy đầu ra như sau:

```
Answer: 42
Ta sẽ theo dõi log phía server, trước hết của PhysicService:
INFO: -------- BEGIN SECURITY INFO --------
INFO: Caller: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
INFO: Invocation subject:
/O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
INFO: Service subject:
/O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
INFO: System subject:/O=Globus/OU=GT3 Tutorial/CN=Globus 3
Administrator
INFO: -------- END SECURITY INFO --------
```

Chú ý rằng, tại thời điểm này, cả chủ thể triệu gọi và chủ thể dịch vụ đều là chủ thể của người gọi.

Bây giờ, theo dõi tiếp các thông điệp của MathService:

```
INFO: Creating MathService instance...
INFO: Created MathService instance
INFO: Obtaining reference to MathService instance...
INFO: Obtained reference to MathService instance
INFO: Invoking 'add' method...
INFO: 'add' invoked by: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
INFO: Invoked 'add' method
INFO: Invoking 'add' method...
INFO: 'add' invoked by: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
INFO: Invoked 'add' method
INFO: Invoking 'add' method...
INFO: 'add' invoked by: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
INFO: Invoked 'add' method
INFO: Invoking 'add' method...
INFO: 'add' invoked by: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
```

```
INFO: Invoked 'add' method
INFO: Invoking 'add' method...
INFO: 'add' invoked by: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
INFO: Invoked 'add' method
INFO: Invoking 'add' method...
INFO: 'add' invoked by: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
INFO: Invoked 'add' method
INFO: Invoking 'add' method...
INFO: 'add' invoked by: /O=Globus/OU=GT3 Tutorial/CN=Borja Sotomayor
INFO: Invoked 'add' method
INFO: Invoking 'getValue' method...
INFO: 'getValue' invoked by: /O=Globus/OU=GT3 Tutorial/CN=Borja
Sotomayor
INFO: Invoked 'getValue' method
INFO: Destroying MathService instance...
INFO: Destroyed MathService instance
```

Mặc dù là MathService được triệu gọi bởi PhysicsService (chạy dưới tài khoản globus), nó đang sử dụng giấy ủy nhiệm của người gọi khi triệu gọi phương thức add.

2.4.6. Cơ chế thông báo trong dịch vụ lưới có cài đặt bảo mật

- Sink

Sử dụng loại "Secure" của notification sink manager để nhận được thông báo:

```
NotificationSinkManager manager =
NotificationSinkManager.getInstance("Secure");
```

Các thuộc tính bảo mật (các thuộc tính phía client) cho toán tử subscribe (được triệu gọi bởi .addListener()) có thể được thiết lập sử dụng hàm manager.init().

```
Map props = new HashMap();
props.put(GSIConstants.GSI_CREDENTIALS, cred);
...
manager.init(props);
...
manager.addListener(...);
```

Các thuộc tính bảo mật (các thuộc tính phía server) cho local sink có thể được thiết lập sử dụng hàm gọi ngược (callback) của chính nó:

```
NotificationSinkCallback callback = ...;
callback.setProperty(GSIConstants.GSI_CREDENTIALS, cred);
...
```

Mô tả triển khai bảo mật có thể được cấu hình sử dụng thuộc tính "securityConfig" như ví dụ:

```
callback.setProperty("securityConfig",        "myservice/my-security-
config.xml");
```

Mặc định, gsi-security-config.xml được sử dụng

- Source

+ Dứ liệu dịch vụ

Để bảo mật khi thông báo, bạn phải thiết lập các thuộc tính bảo mật trên thực thể dữ liệu dịch vụ (service data) bằng cách sử dụng hàm setProperty() như ví dụ:

```
private ServiceData serviceDataElement;
...
this.serviceDataElement.setProperty(Constants.GSI_SEC_CONV,
                                    Constants.ENCRYPTION);
this.serviceDataElement.setProperty(Constants.AUTHORIZATION,

SelfAuthorization.getInstance());
```

+ Sử dụng NotificationProvider

Bạn không nên sử dụng NotificationProvider API trực tiếp. Chúng tôi giới thiệu bạn đẩy thông tin ra sử dụng các phương thức phù hợp trong đối tượng ServiceData. Nếu bạn vẫn muốn sử dụng NotificationProvider API bạn sẽ phải thiết lập các thuộc tính bảo mật, bạn sẽ phải thông qua đối tượng Map tới hàm notify() phù hợp (notify() hay notifyWithAck()).

2.4.7. Một số lỗi thường gặp

```
'[SEC05] Invocation subject not set'
```

Phương thức dịch vụ đang gọi là SecurityManager.setOwnerFromContext() nhưng thực thể run-as của phương thức chưa được thiết lập. Dịch vụ khi đó có thể mất mô tả triển khai bảo mật.

```
'[SEC06] Invocation subject does not contain private credentials'
```

Phương thức dịch vụ đang gọi SecurityManager.setOwnerFromContext() nhưng thực thể run-as của phương thức chưa có giấy ủy nhiệm riêng. Nếu phương thức đang chạy dưới "callers-identity", bảo đảm là client đang gọi dịch vụ sử dụng an toàn hội thoại (GSI Secure Conversation) và cho phép ủy nhiệm khi gọi.

```
org.globus.gsi.proxy.ProxyPathValidatorException: Unknown policy:
1.3.6.1.4.1.3536.1.1.1.7
```

Lỗi này xảy ra khi cố gắng chứng thực GRIM proxy mà không cài đặt một handler để làm việc với các chính sách của GRIM (chính sách có OID 1.3.6.1.4.1.3536.1.1.1.7). Chỉ dẫn về cài đặt chính sách của GRIM được trình bày trong phần trên.