University of Puerto Rico
Mayaguez Campus
Mayaguez, PR
Electrical and Computer Engineering Department

NetTraveler User Guide

University of Puerto Rico, Mayagüez

7-Dec-06

Angel L. Villalaín-García (std. #802-00-8413)

1. NetTraveler Introduction

NetTraveler is a Distributed Database Middleware System designed for Wide Area Networks environment. NetTraveler middleware system will provide:

- Mechanism for heterogeneous data source integration.
- Efficient query execution:
    - Over remote sites that are either mobile clients or enterprise servers
    - By parallel execution
    - Load balancing

The NetTraveler architecture is composed of several server applications implemented as a Java Web Services (see Figure 1). NetTraveler could be seen as a network of federations.

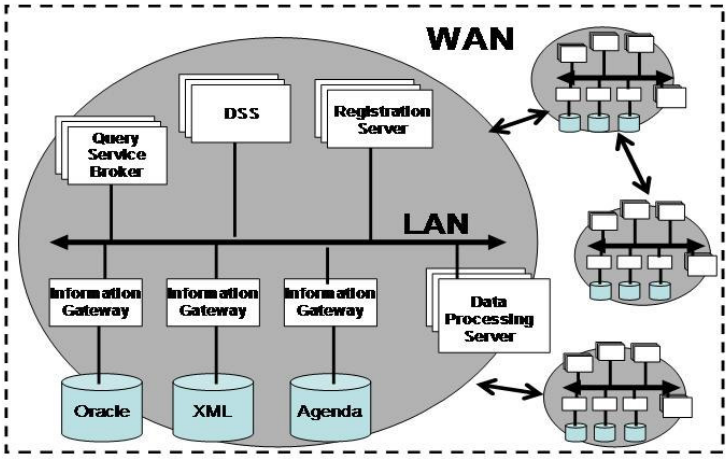


Figure 1 NetTraveler Architecture

Each federation is composed of a set of Query Service Brokers (QSB), a set of Information Gateways (IG), a set of Data Synchronization Servers (DSS), a Registration Server (RS), and the Data Processing Server (DPS). The QSB could be seen as a general purpose query engine, as depicted on Figure 2. It is composed of almost all of the functionality of traditional DBMS query engine, parser, optimizer, operators, but it lacks the actual access to the data. The QSB is in fact the entrance, or the current mechanism to interact with the DMS, is the service in charge of processing query sent to it, and could interact with another QSB, to solve a specific query, in a P2P fashion. The IG provides access to the data. The IG could then be see as the mechanism that implements all of the actual work done by the lower levels presented on Figure 2. The DSS service provides client query recovery mechanism. In case of a client failure the DSS would act as a proxy for the client. When the client returns it would ask the DSS for the results. The RS is an extended catalog manager that also has the responsibility of coordinating the system among federations. And at last the DPS which is a mechanism for interfacing grid services or sensors systems. For our proposed Query Execution engine we will focus our strength on two of the main components of NetTraveler, the Query Service Broker (QSB) and the information Gateway (IG).

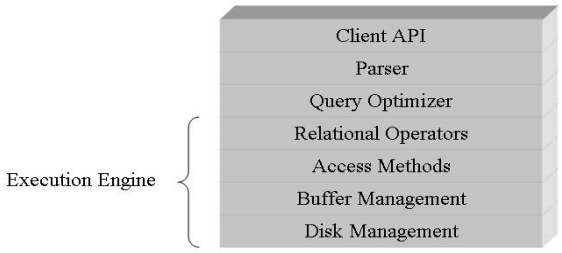


Figure 2 DBMS Architecture

NetTraveler is platform independent, it runs on Linux, Windows, MacOS. Current status of NetTraveler provides only support for JDBC compliance DBMS.

2. Installing NetTraveler

You can download all the NetTraveler code from various locations:

- WALSAIP Website
- CVS Build
    - This version contains always the current status of the project although it is on testing stage. If you would like to have access to the up to date version please use this configuration on eclipse for setting up a cvs.
        - Host: icarus.ece.uprm.edu
        - Path: /home/avillan/ntdemosource/
        - User: avillan
        - Password: sharp-rash
        - Connection type: extssh
        - Port: default CVS port.
    - In this case you will have to select at last projects, NetTraveler project and the Query Browser project if you like to use it as an example.
- Both versions will require you to import it on eclipse.

1. System Requirements

- **Java 2 Standard Edition 5.0**
    1. Download J2SE 5.0 from http://java.sun.com according to your platform.
    2. Follow the installation instructions according to your platform from http://java.sun.com/j2se/1.5.0/install.html.

- **Apache Tomcat**
    1. Download Apache Tomcat from http://tomcat.apache.org. (From version 5.x on)
    2. Follow the installation instruction according to your platform and the version of Apache Tomcat you choose from the following site http://tomcat.apache.org/tomcat-x.x-doc/setup.html, where "x.x" corresponds to the version in use.

- **PostgreSQL**
    1. Although PostgreSQL is our choice you could also install any other DBMS that provides support for JDBC technology. On that case please refer to your documentation for any problems.
    2. Download PostgreSQL from http://www.postgresql.org/ftp/binary/ and select the version that you want to download (we recommend version 8.x.x on). Be sure to remember the version because that would be of used when setting up the jdbc libraries.

3. There are several installation guides for PostgreSQL available on their website, http://www.postgresql.org/docs/techdocs.4.

- **Eclipse**
  1. Download Eclipse 3.2 from http://www.eclipse.org and uncompressed the file on the destination of your choice.

2. Database Setup

Once you have downloaded the code, on the location you used for storing it, please look for a folder called "<user.path>\database", where <user.path> is the location where you save code (see figure 3).

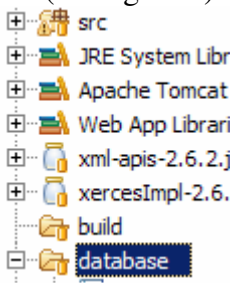


Figure 3 Database Scripts Folder

1. Inside it, there is a file called "nt_meta.sql". This file contains the definition of the database that we need to create for the system.
2. So first, once you have installed and created a user on your DBMS of choice, please create a DB called "nt_meta" for example. With PostgreSQL:

Figure 4 Creating Database nt_meta

3. Once created, and according to the documentation of your DBMS, execute the "nt_meta.sql" script to create all the needed relations for the nt_meta DB, and for the current version of the catalog.

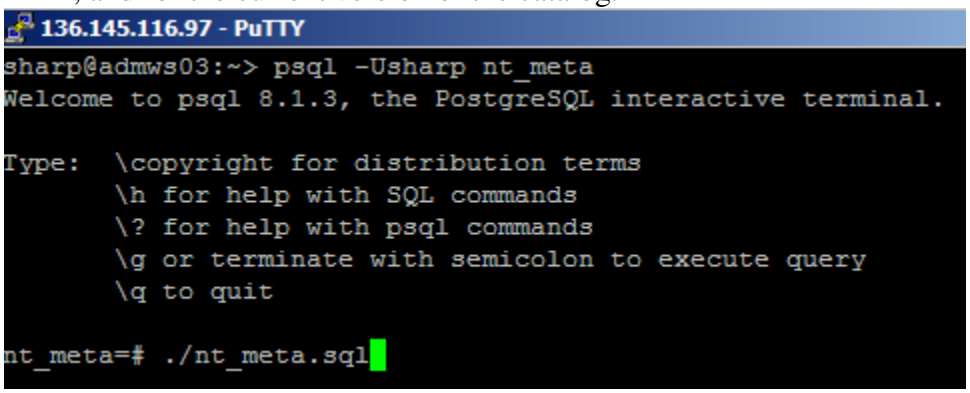


Figure 5 Execution of nt_meta.sql script

4. Please refer to your DBMS documentation in order to assure and configure your DBMS for accepting connection from remote sites, because the process we are showing you requires additional machines in order to configure and deployed a small NetTraveler federation.

3. Installing NetTraveler

- **Catalog Information (See appendix)**

  The Catalog contains several tables used for storing all the metadata concerning to services, relation between services and the remote data sources. Is important to define each relation in order to understand what it is needed to store in each one in order to fully configure the system.
  First we begin with the relation *Site*. Each site or service that you want to define in the system will need a record stored in this relation.
  The id field represents the ID assigned to a service. It must be unique, and right now it must be assigned manually. The ip field as its name implies is the IP of the host where the service relies. The type field represents the type of service that is being registered, if it is a QSB the value must be 0, if it is an IG it must be 1, and DSS 2. The rest of the services are not taken into consideration right now. The port, the port number in which tomcat is running, and the impl field references the implementation of each service. Right now they are implemented only as Web Services, and the value must be 0 then (see Figure 6).

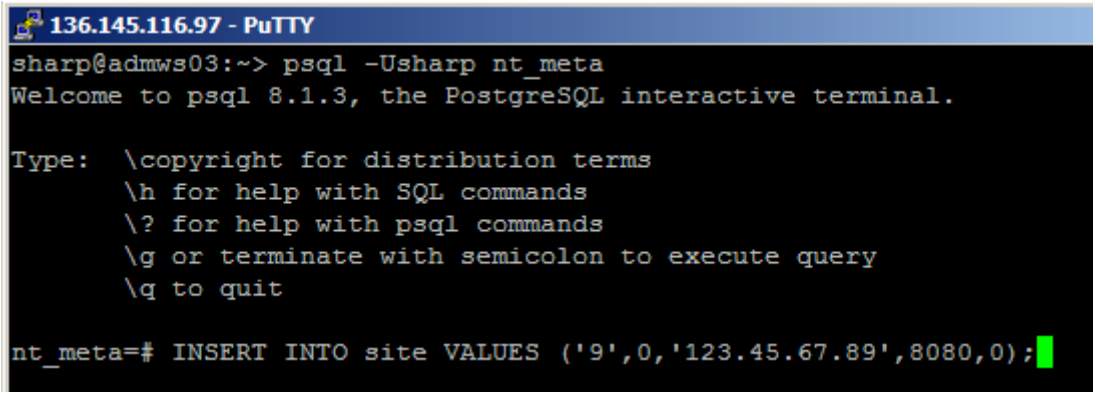


Figure 6 Defining QSB with ID = 9

Peer relation is the next important table. This relation defines the relation between two services that are peers. Peer relation is bidirectional. Hence if a service A is peer of a service B then B is also a peer of A. To maintain this relation, there will be two records peer each peer relation. Each field references the ID of each service.

```
nt_meta=# INSERT INTO peer VALUES('9','10');

nt_meta=# INSERT INTO peer VALUES('10','9');
```

Figure 7 QSB 9 and QSB 10 peer definition

Table knows represents the relation of known IG for each QSB. It is important to understand that each field on this table represents the ID of each corresponding service, qsb field referencing the ID of a QSB and ig field its corresponding ID.

```
nt_meta=# Insert into knows values ('9','10');
```

Figure 8 QSB 9 is knows IG 10

Table relation has the name of each relation on each remote data source. ID field is the id we assigned to each relation, the name field the name of the relation.

```
nt_meta=# Insert into relation values (1,'students');
```

Figure 9 Defining the students table

Table attribute stores the metadata concerning to each relation, in fact it defines each field for each relation. The only field that it is needed to explain in depth is the type field. For more information on the value for this field please refer to the javadoc of java.sql.Types. The rest define the id of each column, the relation id, the name of the field and the position.

```
nt_meta=# select * from attribute;
 id | rid |    name     | type | pos
----+-----+-------------+------+-----
  1 |   1 | sid         |    4 |   0
  2 |   1 | sname       |   12 |   1
  3 |   1 | sage        |    4 |   2
  4 |   2 | measureid   |    4 |   0
  5 |   2 | sensorid    |   12 |   1
  7 |   2 | temperature |   12 |   3
  6 |   2 | locationid  |   12 |   2
(7 rows)
```

Figure 10 Example attribute table and its info

Other important relation is the table keys that store the information of which attribute is a key.

```
nt_meta=# insert into keys (1,1,1,true);
```

Figure 11 Example definition of a key for a table

The replication_meta relation only stores the information concerning on the number of partitions for each replica.

```
nt_meta=# insert into replication_meta values (1,6);
```

Figure 12 Example for specifying that there is a table with 6 buckets

The site_stats table stores information concerning to the actual status of each site, resources, availability, etc.

```
          Table "public.site_stats"
  Column   |         Type          | Modifiers
-----------+-----------------------+-----------
 minvalue  | double precision      | not null
 maxvalue  | double precision      | not null
 perprocess | double precision     | not null
 rtype     | integer               | not null
 id        | character varying(15) | not null
Foreign-key constraints:
    "site_stats_id_fkey" FOREIGN KEY (id) REFERENCES
```

Figure 13 Definition of site_stats table

The last relation is the Store relation. It defines which remote site provides access to that relation. *Rid* references the id of the relation, sid references the site id of the IG which provides access to that relation.

```
nt_meta=# INSERT INTO STORE (1,'10');
```

Figure 14 IG 10 provides access to relation 1

Once the nt_meta database has the necessary information, another step we need to perform is to configure the access to this database. Inside the package *edu.uprm.admg.nettraveler.catalag* there is a property file called *catalog.properties.* This file must be edited with the information concerning the access to your local DBMS where the nt_meta DB relies.

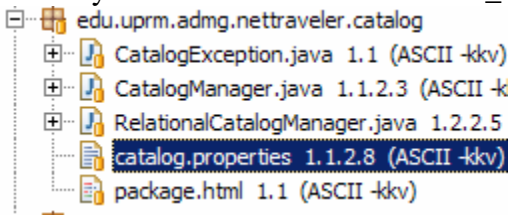


Figure 15 Catalog Properties

ds.driver variable must contains the class implementing the driver for your DMBS. In our case we are using the postgresql driver.
ds.uri variable must point the correct URI for the database. Here is where you can choose to change nt_meta as the name for the catalog and use other instead.
ds.login variable must be the user name and ds.password the password corresponding to that user name.
ds.connection variable represents the number of connections that our system would keep opened to the databases.
ds.transaction defines the transaction level. Leave it on 8 by default.
ds.autocommit defines if the transaction must be immediately committed. By default again, leave the default value.

```
# Properties file for the data source.

# Database connectivity (JDBC)
#ds.driver=com.mysql.jdbc.Driver
ds.driver=org.postgresql.Driver

#ds.uri=jdbc\:mysql\://127.0.0.1:3306/college
#ds.uri=jdbc\:postgresql\://136.145.116.97:5432/nt_meta
ds.uri=jdbc\:postgresql\://127.0.0.1:5432/nt_meta
# User and password
ds.login=sharp
ds.password=sharp-rash

# Number of connections
ds.connections=3

# Transaction Level as defined by
# java.sql.Connection
# Read-Uncommited = 1, Read-Commited = 2, Repeatable-Read = 3, Serializable = 8
ds.transaction=8

#Autocommit
ds.autocommit=true;
```

Figure 16 Catalog.properties

- ## Data Source Setup

  Open the file *ds.properties* placed inside **WebContent\WEB-INF\conf** (see Figure 17). The file structure is similar to file *catalog.properties* shown above. The only difference is the *ds.type*. This variable is must be left with the default value. The rest of the variables represent the same information as presented above. In this case the information must correspond to the data source that we want to integrate.

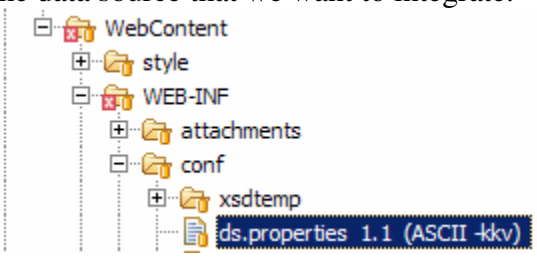


Figure 17 ds.properties location

```
# Properties file for the data source.
# Type of data source
# Values: relational
ds.type = relational
# Database connectivity (JDBC)
#ds.driver=com.mysql.jdbc.Driver
ds.driver=org.postgresql.Driver

#ds.uri=jdbc\:mysql\://127.0.0.1:3306/college
ds.uri=jdbc\:postgresql\://127.0.0.1:5432/nettraveler

# User and password
ds.login=sharp
ds.password=sharp-rash

# Number of connections
ds.connections=20

# Transaction Level as defined by
# java.sql.Connection
# Read-Uncommited = 1, Read-Commited = 2, Repeatable-Read = 4, Serializable = 8
ds.transaction=8

#Autocommit
ds.autocommit=true;
```

Figure 18 ds.properties file

- **Other Important Configuration Files**

  Under **WebContent\WEB-INF\conf\xsdtemp** you can find several xml schema file definitions (*XSD*). This xml schema defines several aspects, from tables, operations, and resources, to actually completely define each service. They represent the next step in order to integrate all configurations within a single entity.

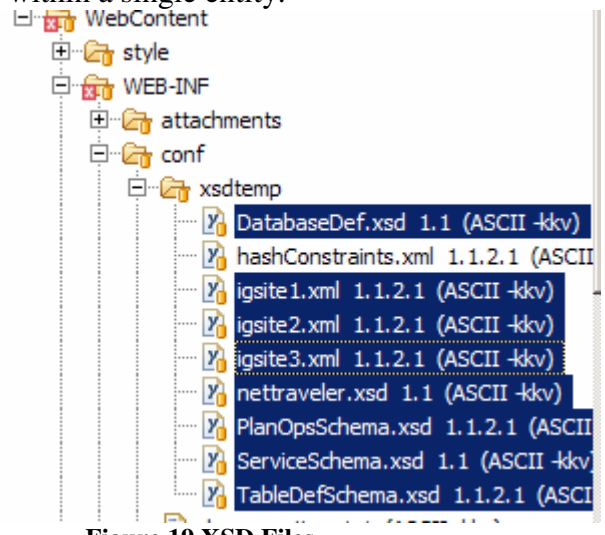


Figure 19 XSD Files

Right now, the *XSD* files are not being used for that purpose, they only serve as a mechanism for defining remote services, which we needed to

take into consideration for our scheduling testing purposes. For that reason, for each remote IG site, that we want to incorporate we need to add a representation for that site on this folder. Please read one of the many *igsite.xml* files in order to understand the structure of the *XSD* (see appendix).

- **Configuring Services**

  Once you have configured the catalog, set up the information of how to access the catalog, and configure the data source access mechanism, you are ready to deploy a service of NetTraveler. But in order to do that you first need to define which service you want to deploy. Before you do it, take your time and open the file *ServiceSchema.xsd* file place on **"WebContent\WEB-INF\conf\"**. Please read the documentation on it in order to understand the information needed for starting and defining a service.

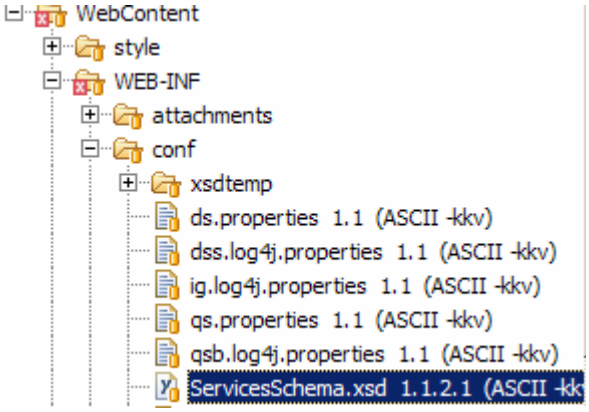


Figure 20 ServicesSchema.xsd

But if you want to avoid that step, you can go directly to the file *servicesdef.xml* inside the same folder. The minimum information that you need to change is those values concerning the IP, the Port and the type of service that you wish to deploy. So if you want to deploy an IG please modify the type parameter and write "IG". The same goes for QSB and DSS.

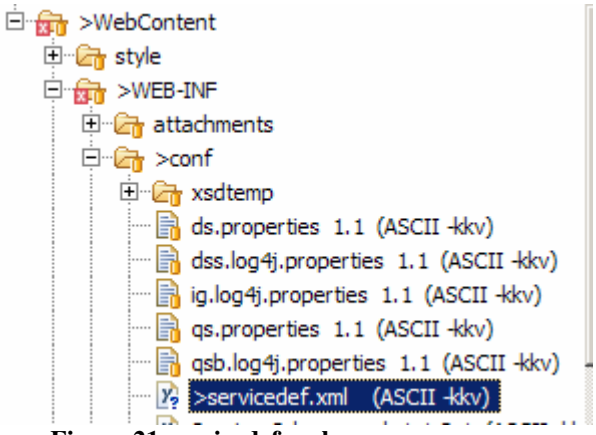


Figure 21 servicedef.xml

Again if you want a better understanding of the other parameters please refer to the *ServiceSchema.xsd* file. Figure 22 shows the configuration for a QSB that is running on port 8090 and with IP 136.145.116.97.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:ServerConfiguration xmlns:tns="http://admg.uprm.edu/ServicesConfiguration"
  <Service>
    <port>8090</port>
    <type>QSB</type>
    <impl>AXIS_1_4</impl>
    <scheme>NONT</scheme>
    <optimizer>edu.uprm.admg.nettraveler.optimizer.TestOptimizer</optimizer>
    <allocation>1.3</allocation>
    <query>20</query>
    <result>100</result>
    <idle>300000000</idle>
    <thread>259200000</thread>
    <id>2</id>
    <reroute>5</reroute>
    <stats>false</stats>
    <ip>136.145.116.97</ip>
  </Service>
</tns:ServerConfiguration>
```

Figure 22 servicedef.xml example

Once you have done that press right click over the name of the project to display a popup menu (Figure 23). Select *Export* option.

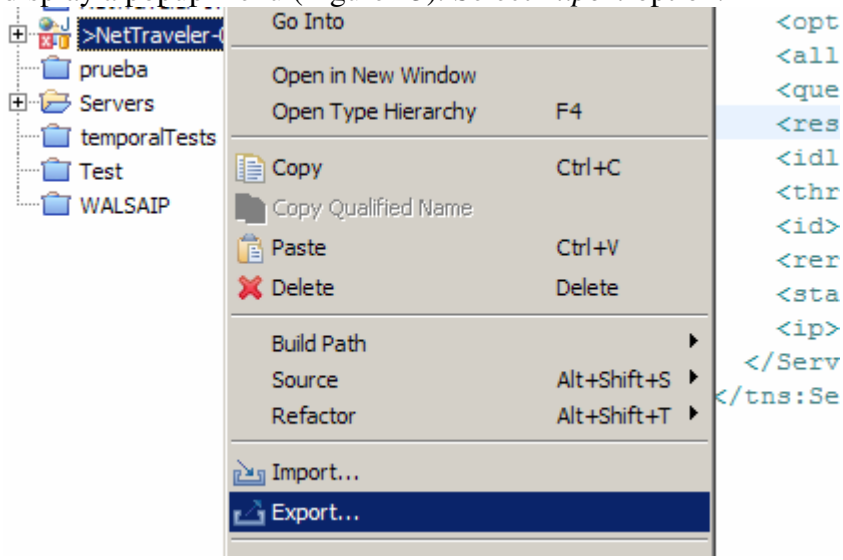


Figure 23 Exporting project

The next dialog will ask you to choose from several different options, please under web, select *WAR* file (as shown on Figure 24).

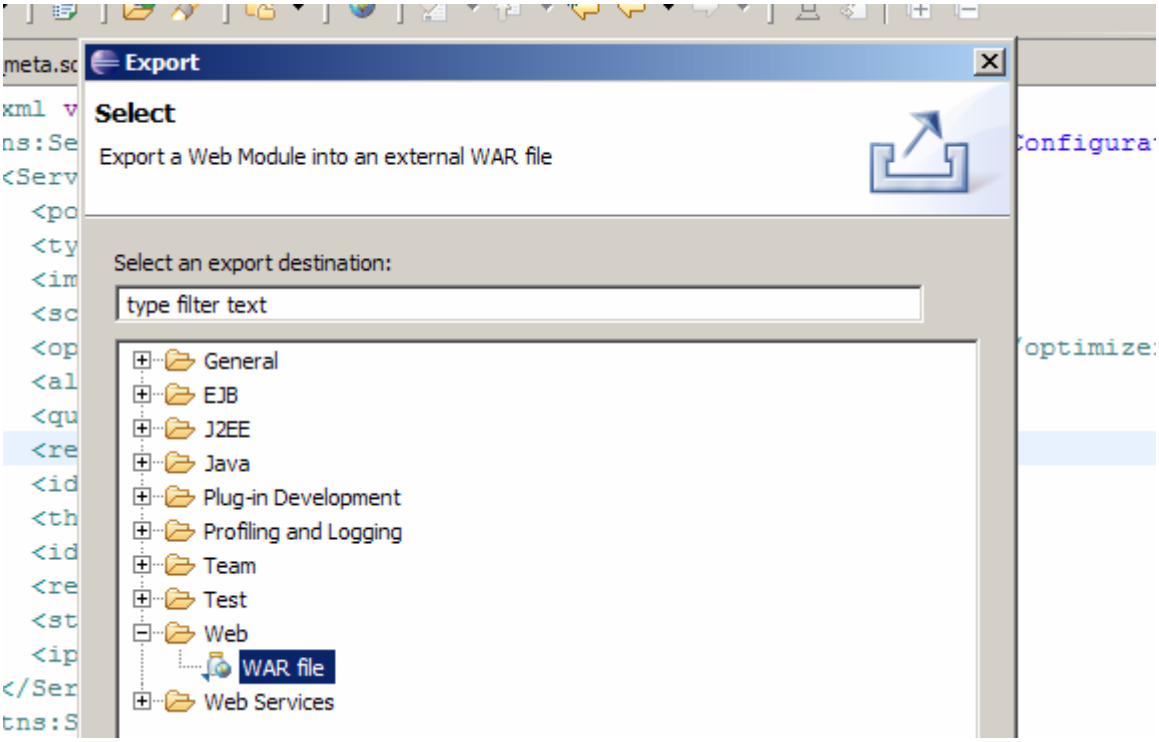


Figure 24 War File Selection

Once selected this option please, write the name of the war file, (nettraveler.war) and choose to finish the export process, see Figure 24.

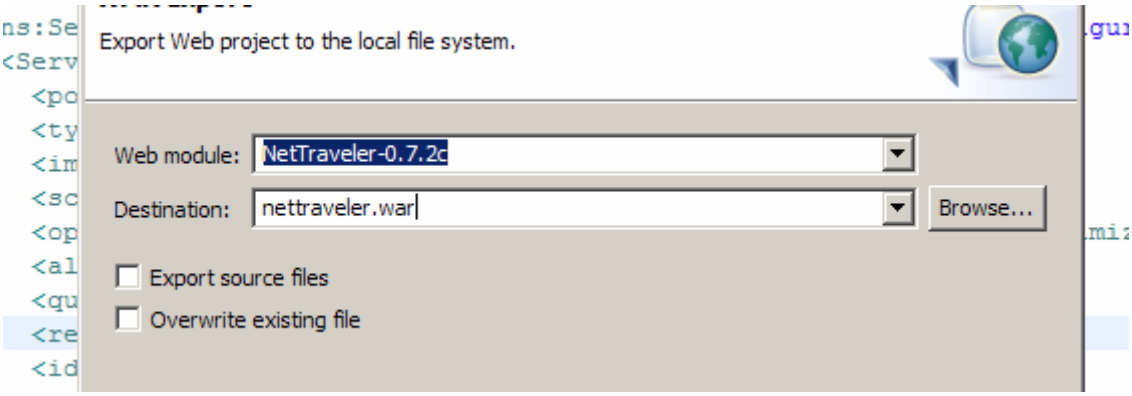


Figure 25 Finishing export process

The last step would be to deploy this generated war file on Tomcat. To deploy it on Tomcat, copy the war file to **<TomcatPath>\webapps.** Once you restart Tomcat the service would be available. The next steps would be to configure runtime configurations for each service, but that is part of the next section which explains how to run and use the demo.

Please feel free to test and run each of the programs under the *edu.uprm.admg.nettraveler.test*, to finally test your configuration.

One important aspect is that current NetTraveler implementation defines that for each given service there is an instance of Tomcat. So for each QSB you will need another Tomcat server. Current implementation relies on having a single Tomcat server for each service.

1. Running the Demo

1. *Running the Query Browser*

- In order to run the system please, press right click on the QueryBrowser.java and select Run As → Java Application, (see Figure 26).

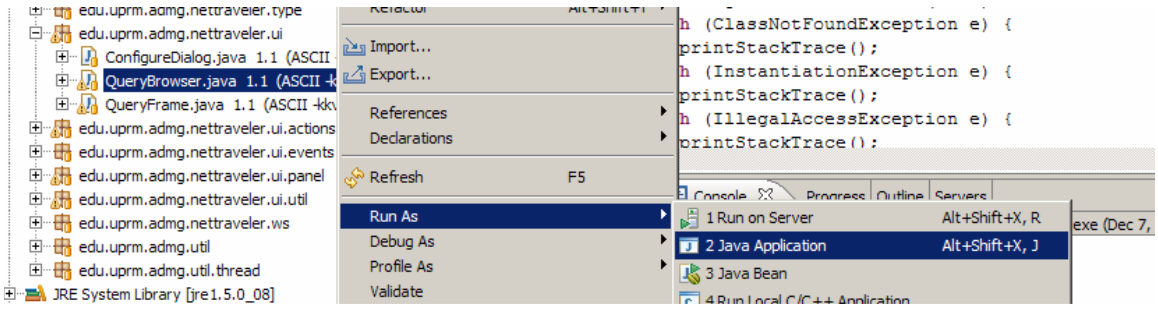


Figure 26 Running Query Browser Client

- The next screen that will appear is the Figure 27. On window you will have to specify the parameters to connect to where the catalog is running in order to have access to see the schema and to select which of the QSB you are running on your system would be the entrance to the client.

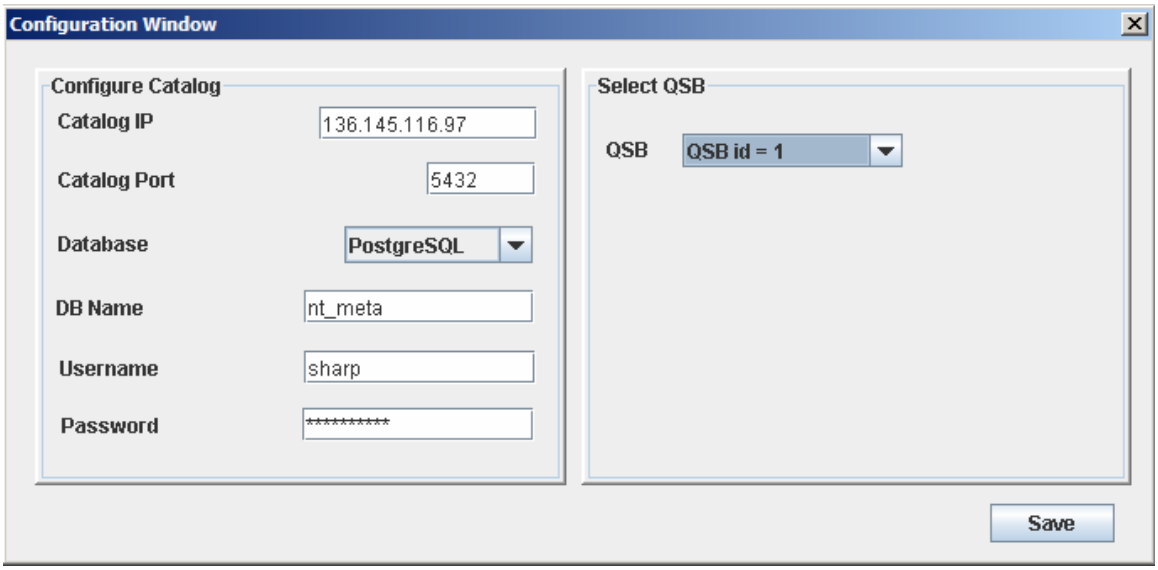


Figure 27 Query Browser Configuration Windows

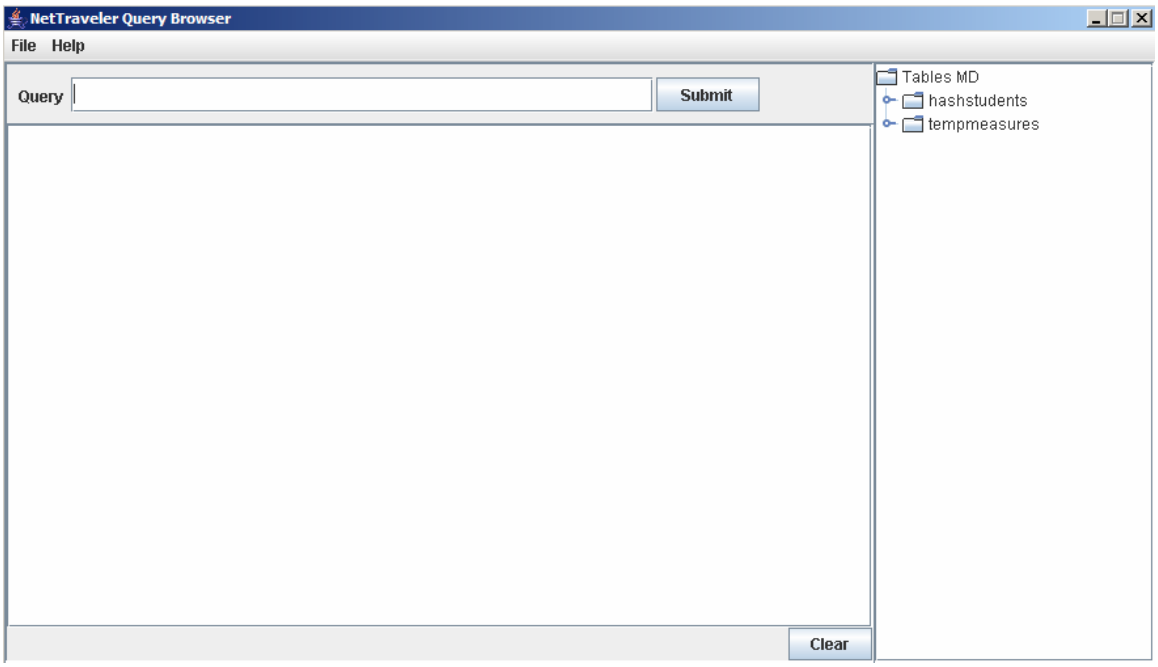
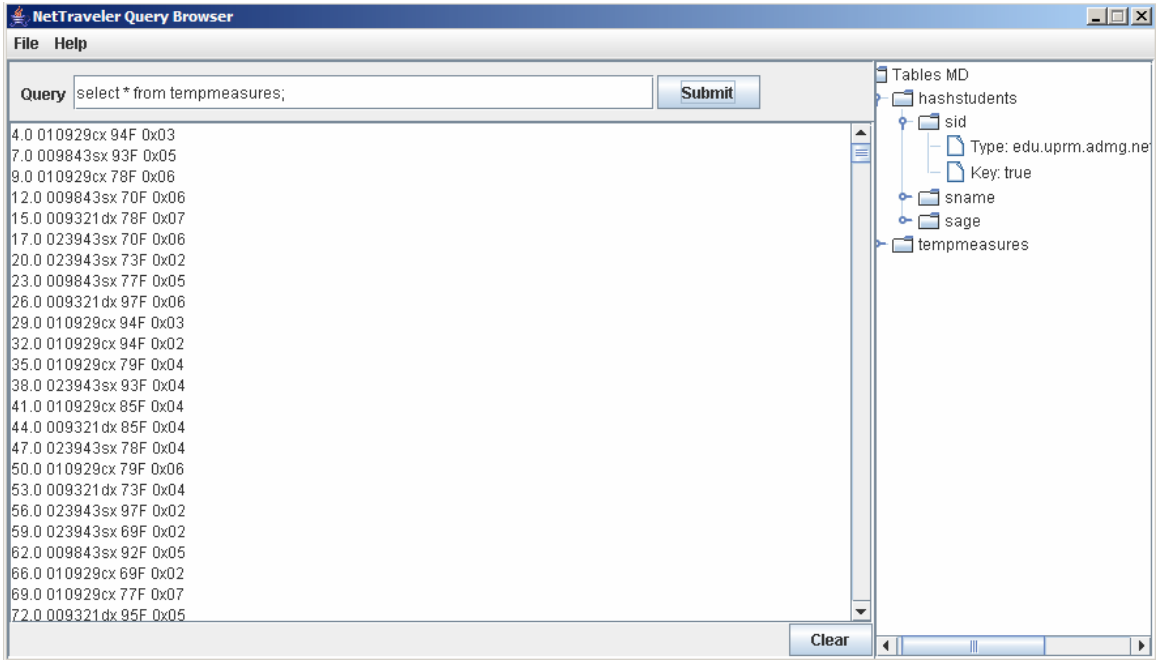


Figure 28 Query Browser

- Once you have the configuration correctly the next window that will appear will be the main window (Figure 28 and 29) you will see several areas. To your right, you will find the metadata concerning to your relations. On the upper side you will see a text area where you can submit your queries to the system, and on the center of the screen you will see the results.



2. Using NetTraveler Webapp

- Another important aspect is web based. This interface is useful for setting up some runtime configurations and to set up some actions that would have effects on the data sources, for example managing replicated data sets (partitioning data). Figure 29 shows the login screen. To manage the users that can admin the systems please modify the file *users.xml* that could be found on **WebContent\WEB-INF\sec** (Figure 30).

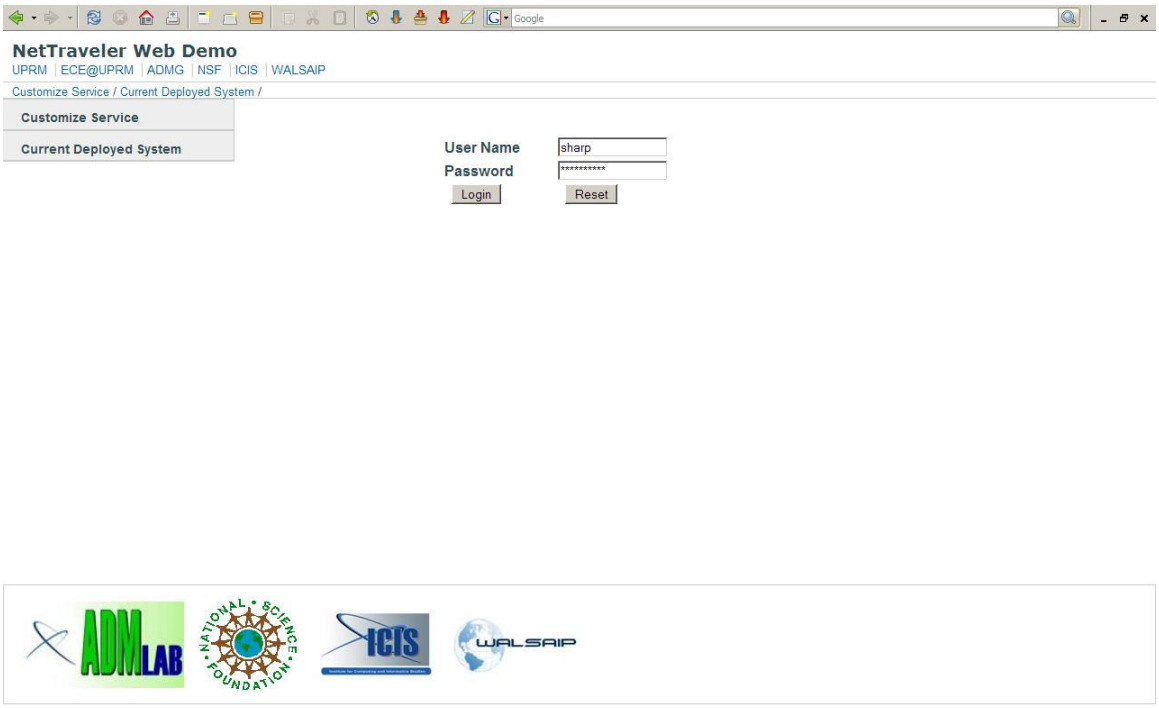


Figure 29 Login Screen

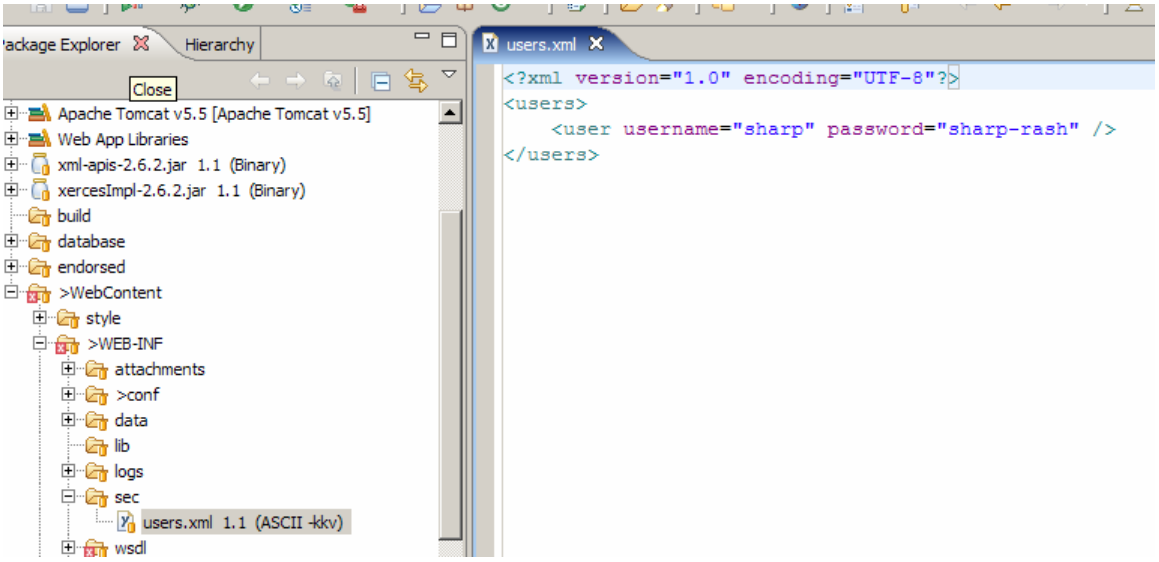


Figure 30 users.xml

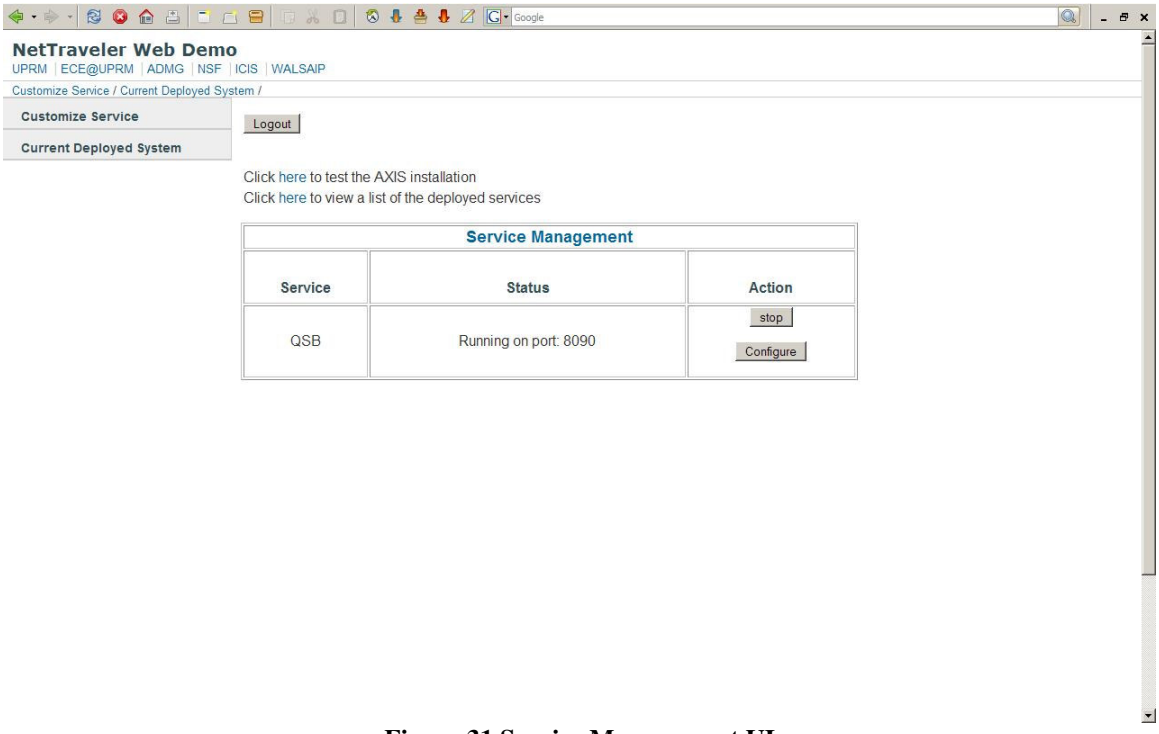


Figure 31 Service Management UI

- Once you have successfully log in, you will face the service management interface. In this interface you will have to decide to possible options 1) starting/stopping the service and 2) configure it.

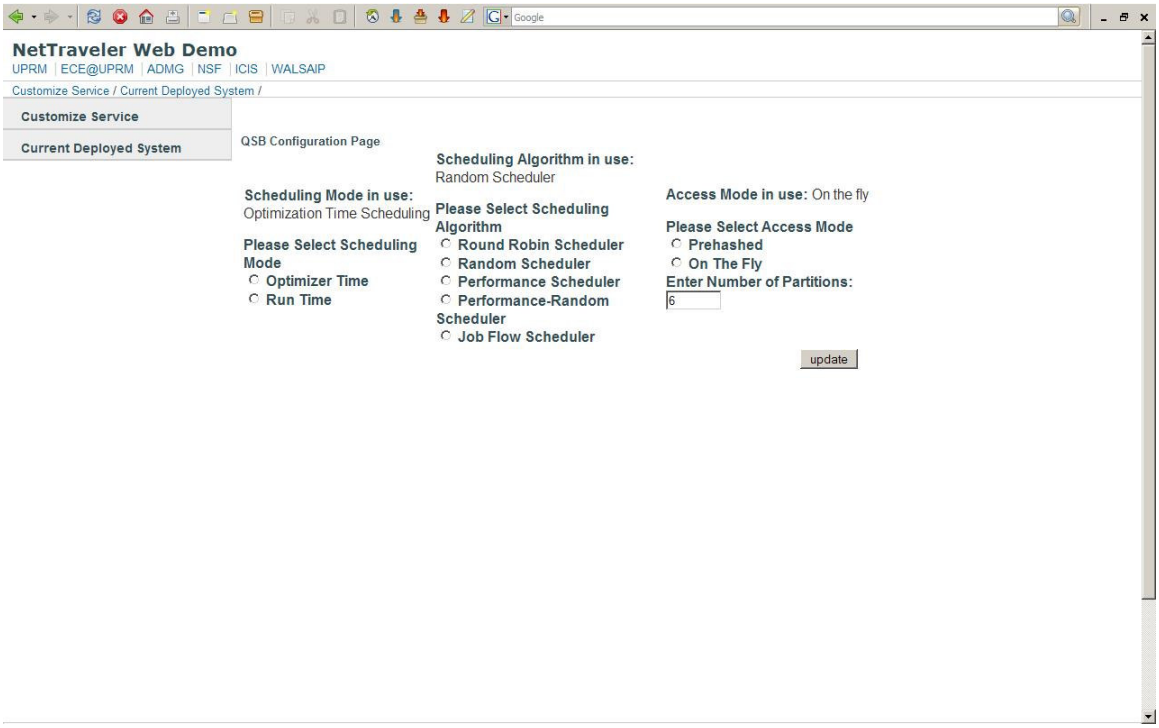


Figure 32 QSB Configuration UI

- Depending on the service you are connected it will display different configurations. Figure 32 shows the different configuration selection for a selected QSB. For an IG the configuration would be different and it is only for configuration of the number of buckets for a specific relation.

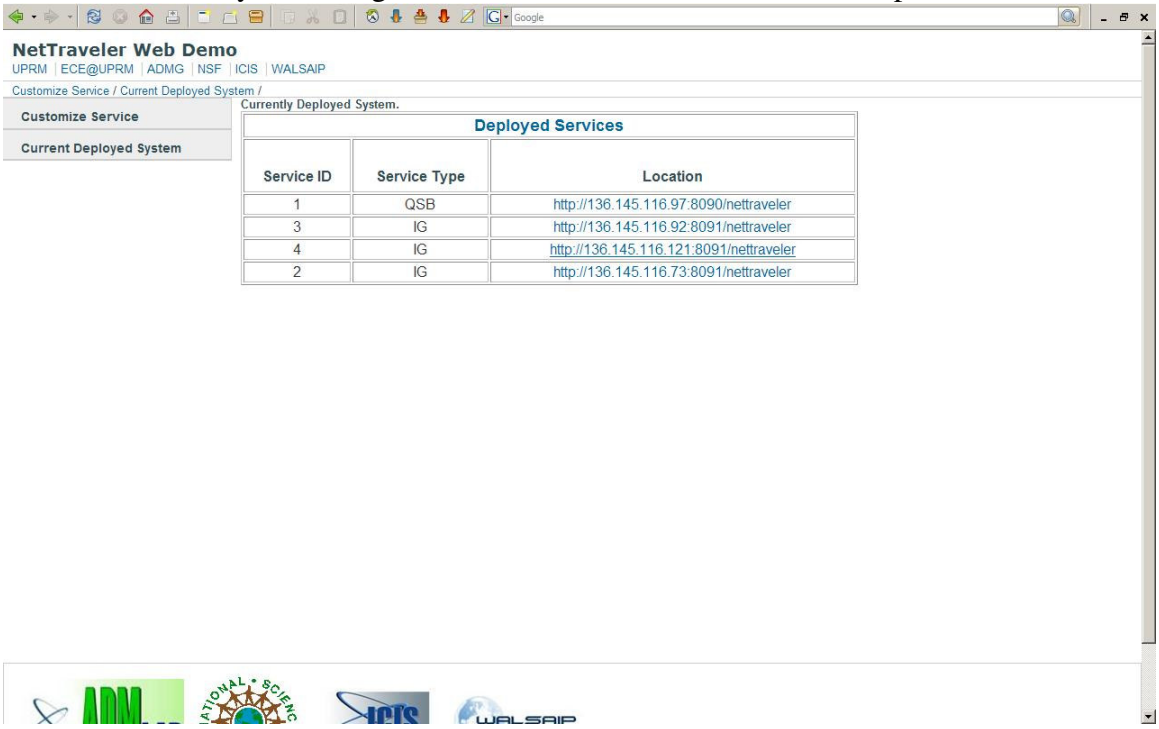


Figure 33 Deployed Service List

- On the left side of the webpage there is a menu, which shows you two options. The first one "Customize Service" it takes you to the configuration screen for the current service. If you select the second option "Current Deployed System" will provide you with a mechanism for knowing which services are available and access to each one of the service for configuration purposes.
- As you can see this is a mere proof of concepts. This is only a UI for testing, and tutorial purposes. See source code if you are interested in interfacing your own applications with NetTraveler.

And that's the general information that you need to understand in order to run NetTraveler. For any doubts or comments, please feel free to contact me at angel.villalain@ece.uprm.edu

Appendix

```
*****************************************************************
*                              nt_meta.sql                      *
*****************************************************************
-- Site table. Info about the data sources sites.
-- id = ID of the service
-- type = Type of service (IG=, DSS=, QSB=)
-- ip = IP address of the service
-- port = Port where the service is listening
-- impl = Implementation type of the service. (Web service, sockets)
create table site(
       id varchar(15) not null CHECK (id > 0),
       type int not null,
       ip varchar(40) not null,
       port int not null default 8080,
       impl int  not null,
       primary key (id)
);

-- Table of peers.
-- States a relation between two services that are peers.
-- Peerism is bidirectional. Hence if a service A is peer of
-- a service B then B is also a peer of A. To maintain this
-- relation, there will be two records peer each peer
-- relation.

create table peer(
       s1 varchar(15),
       s2 varchar(15) ,
       primary key(s1, s2),
       foreign key (s1) references site(id) on delete cascade,
       foreign key (s2) references site(id) on delete cascade
);
-- This table tells which QSBs know which IGs
create table knows(
       qsb varchar(15) ,
       ig  varchar(15) ,
       primary key(qsb, ig),
       foreign key (qsb) references site(id) on delete cascade,
       foreign key (ig) references site(id) on delete cascade
);

-- Catalogs metadata
--create table catalog(
  --     id int not null CHECK (id > 0),
    --   name varchar(100) not null,
      -- primary key(id)
--);

--  Metadata of relations
create table relation(
       id int not null CHECK (id > 0),
       name varchar(100) not null,
       primary key(id)
```

```
);

-- Holds the attributes of every relation
create table attribute(
       id int not null CHECK (id > 0),
       rid int,
       name varchar(100) not null,
       type int not null,
       pos int not null CHECK (pos >= 0),
       primary key (id),
       foreign key (rid) references relation (id) on delete cascade
);

-- Key table
-- Tells the attribute that is unique for a relation in
-- a site
create table keys(
    id    int     not null CHECK (id > 0),
      aid   int   not null,
    cid   int     not null,
    isKey boolean not null,
    primary key(id),
    foreign key(aid) references attribute(id) on delete cascade,
    foreign key(cid) references catalog(id) on delete cascade
);

create table replication_meta(
      rid   int   not null,
      numbuckets int not null,
      foreign key(rid) references relation (id) on delete cascade
);

create table site_stats(
      minvalue    double      not null,
      maxvalue    double      not null,
      perprocess  double      not null,
      rtype       integer     not null,
      id                varchar(15) not null,
      foreign key(id) references site(id) on delete cascade
);

create table store(
       rid int not null,
       sid varchar(15) not null,
       primary key (rid, sid),
       foreign key (rid) references relation(id) on delete cascade,
       foreign key(sid) references site(id) on delete cascade
);
```

```
*****************************************************************
*                              igsite1.xml                      *
*****************************************************************
<?xml version="1.0" encoding="UTF-8"?>
<tns:ServiceDef xmlns:db="http://nettraveler.admg.uprm.edu/DatabaseDef"
xmlns:pln="http://nettraveler.admg.uprm.edu/PlanOpsSchema"
xmlns:serv="http://nettraveler.admg.uprm.edu/ServiceSchema"
xmlns:table="http://nettraveler.admg.uprm.edu/TableDefSchema"
xmlns:tns="http://nettraveler.admg.uprm.edu/NettravelerSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://nettraveler.admg.uprm.edu/NettravelerSchema
nettraveler.xsd http://nettraveler.admg.uprm.edu/TableDefSchema
TableDefSchema.xsd http://nettraveler.admg.uprm.edu/ServiceSchema
ServiceSchema.xsd http://nettraveler.admg.uprm.edu/DatabaseDef
DatabaseDef.xsd http://nettraveler.admg.uprm.edu/PlanOpsSchema
PlanOpsSchema.xsd ">
  <ServiceIG>
    <SiteProf>
      <ID>1</ID>
      <URI>http://localhost:8091/nettraveler/qs</URI>
      <ip>136.145.116.97</ip>
      <port>8091</port>
      <type>IG</type>
      <impl>AXIS_1_4</impl>
      <siteprofile>
        <Type>MEMORY</Type>
        <MinValue>128</MinValue>
        <MaxValue>1024</MaxValue>
        <PerProcess>0.0</PerProcess>
      </siteprofile>
      <siteprofile>
        <Type>CPU</Type>
        <MinValue>0.4</MinValue>
        <MaxValue>0.8</MaxValue>
        <PerProcess>0.0</PerProcess>
      </siteprofile>
      <siteprofile>
        <Type>PROCESS</Type>
        <MinValue>0.2</MinValue>
        <MaxValue>0.6</MaxValue>
        <PerProcess>0.0</PerProcess>
      </siteprofile>
      <siteprofile>
        <Type>DISKACCESS</Type>
        <MinValue>0.1</MinValue>
        <MaxValue>0.3</MaxValue>
        <PerProcess>0.0</PerProcess>
      </siteprofile>
      <recoveryprofile/>
    </SiteProf>

<optimizer>edu.uprm.admg.nettraveler.optimizer.TestOptimizer</optimizer
>
    <allocation>1.3</allocation>
    <query>20</query>
    <result>10</result>
```

```
    <idle>60000</idle>
    <thread>70000</thread>
    <reroute>5</reroute>
    <stats>false</stats>
    <Database>
      <HostName>ADMWS03</HostName>
        <ip>136.145.116.97</ip>
        <port>5432</port>
        <connections>1</connections>
        <DSType>RDBMS</DSType>
        <Driver>org.postgresql.Driver</Driver>
        <URI>jdbc\:postgresql\://127.0.0.1:5432/nettraveler</URI>
        <databaseName>nettraveler</databaseName>
        <userlogin>sharp</userlogin>
        <password>sharp-rash</password>
        <TransactionLevel>8</TransactionLevel>
        <autocommit>true</autocommit>
        <IG>IG</IG>
        <Table>
          <TableName>hashstudents</TableName>
          <Columns>
            <ColumnName>sid</ColumnName>
            <Type>
              <SourceType>int8</SourceType>
              <Size>0</Size>

<ImplType>edu.uprm.admg.nettraveler.type.MIInteger</ImplType>
            </Type>
            <isKey>true</isKey>
            <TableName>hashstudents</TableName>
            <Position>0</Position>
            <Schema>nont</Schema>
          </Columns>
          <Columns>
            <ColumnName>sname</ColumnName>
            <Type>
              <SourceType>varchar</SourceType>
              <Size>30</Size>

<ImplType>edu.uprm.admg.nettraveler.type.MIString</ImplType>
            </Type>
            <isKey>false</isKey>
            <TableName>hashstudents</TableName>
            <Position>1</Position>
            <Schema>nont</Schema>
          </Columns>
          <Columns>
            <ColumnName>sage</ColumnName>
            <Type>
              <SourceType>int8</SourceType>
              <Size>0</Size>

<ImplType>edu.uprm.admg.nettraveler.type.MIInteger</ImplType>
            </Type>
            <isKey>false</isKey>
            <TableName>hashstudents;</TableName>
            <Position>2</Position>
```

```
            <Schema>nont</Schema>
          </Columns>
          <databaseName>nettraveler</databaseName>
          <Cardinality>0</Cardinality>
          <Owner>sharp</Owner>
          <ReplicationInfo>
            <TableName>hashstudents</TableName>
            <AllowReplication>true</AllowReplication>
            <NumBuckets>4</NumBuckets>
          </ReplicationInfo>
        </Table>
    </Database>
    <Operators>
      <Name>HashConstraints</Name>
      <Plan>
        <PlanName>HashAccessPlan</PlanName>

<ClassName>edu.uprm.admg.nettraveler.plan.HashAccessPlan</ClassName>
      </Plan>
      <Constraints>
       <Type>MEMORY</Type>
        <MinValue>128</MinValue>
        <MaxValue>1024</MaxValue>
        <PerProcess>0.0</PerProcess>
      </Constraints>
      <Constraints>
       <Type>CPU</Type>
        <MinValue>128</MinValue>
        <MaxValue>1024</MaxValue>
        <PerProcess>0.0</PerProcess>
      </Constraints>
    </Operators>
  </ServiceIG>
</tns:ServiceDef>
```