CASPUR

NETTAB 2006

Thursday, July 13,
(at POLARIS Science and Technology Park of Sardinia)

New bioinformatics applications based on Web Service Technologies and GRID Computing

Tiziana Castrignanò, CASPUR, Rome, Italy
tiziana.castrignano@caspur.it

Bioinformatics

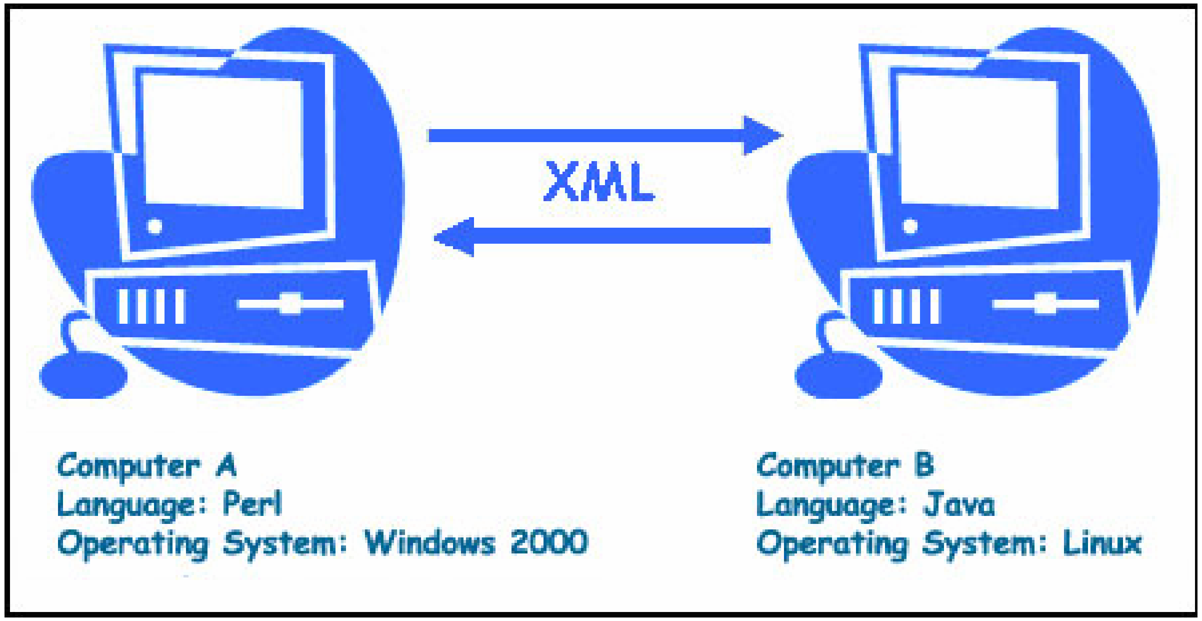
Bioinformatics is an emerging scientific discipline that uses information technology to organize, analyze, and distribute biological information in order to answer complex biological questions.

It involves the solution of complex biological problems using computational tools and systems. It also includes the collection, organization, storage and retrieval of biological information from databases.

Web services

The Web services are a type of service that can be shared by and used as components of distributed Web-based applications.

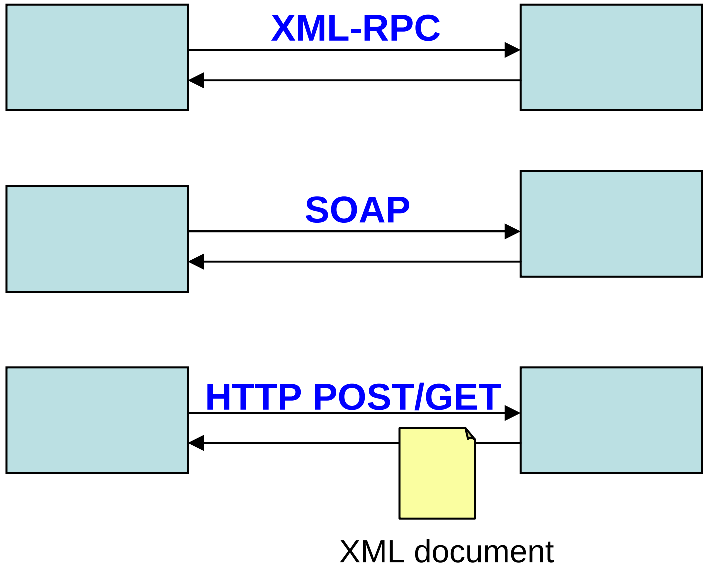
They uses a standardized XML messaging system, and they are not tied to any one operating system or programming language.



Web services

There are several alternatives for XML messaging.

For example you could use XML Remote Procedure Calls (XML-RPC) or SOAP or HTTP GET/POST passing arbitrary XML documents. Any of this option can work.



XML messaging for web services

Web service definition

1. Web services are accessed over the Web.

2. Web services describe themselves using an XML-based description language (WSDL).

3. Web services communicate with clients (both end-user applications or other Web services) through XML messages that are transmitted by standard Internet protocols, such as HTTP or FTP.

4. Web services are not “tied” to any operating system or programming language (the communication beetwen client and server is based on XML)

Web service properities requested

1- *self-describing*

if you publish a new web service, you should also publish a public interface to the service and a human-readable documentation, so that other developers can more easily integrate your service

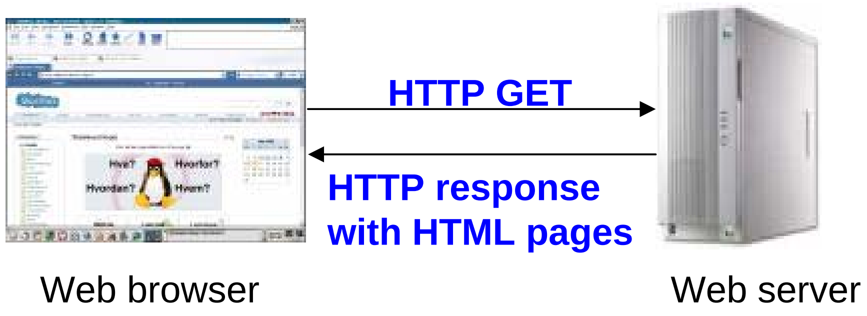
2- *discoverable*

there should be some simple mechanism for you to publish your new web service, so that interested parties can find the service and locate its public interface

CASPUR

Web service goal

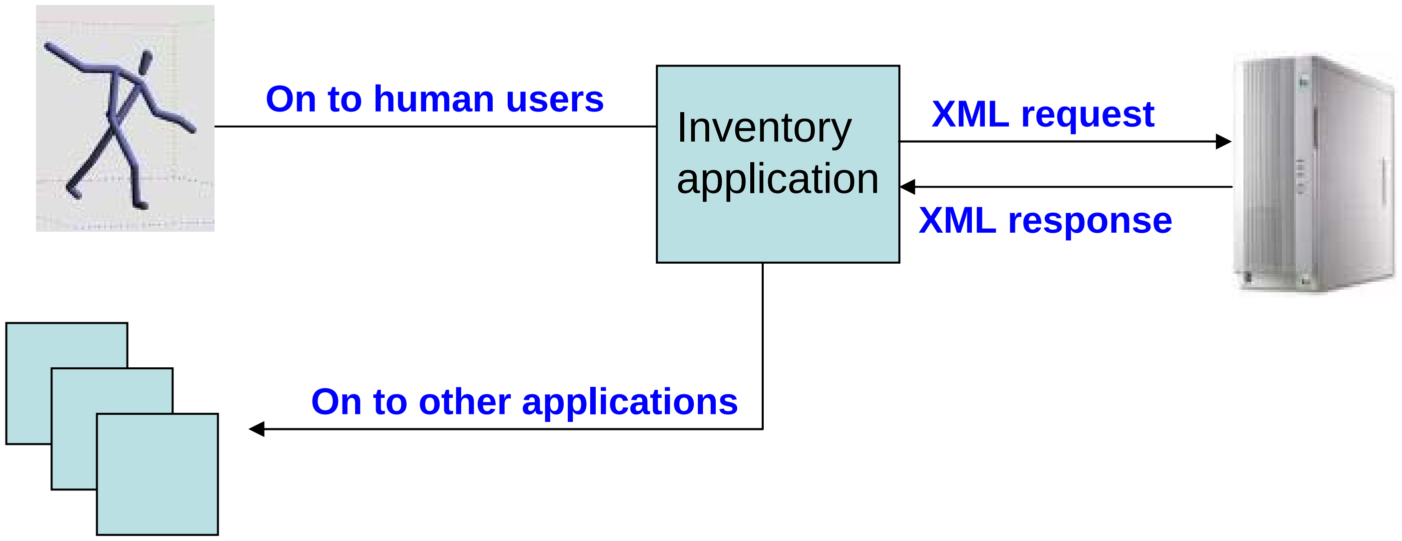
For years developers have created CGI programs and Java servlets designed primarily for use by other applications. Main limit of this technology was that most of these systems consisted of ad hoc solutions!



human-centric Web

Web service goal

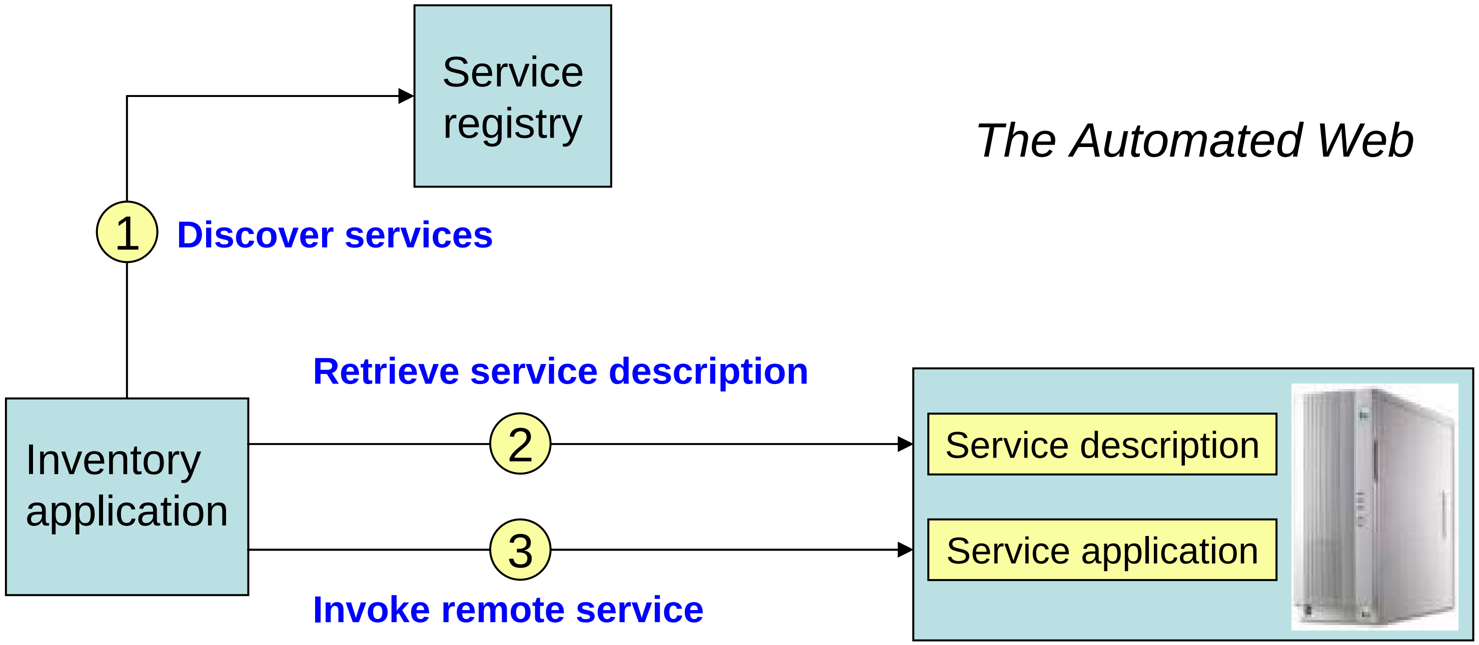
With web-service we move from a human-centric Web to a conversation that take place directly **between applications**.



With web services the promise of some standardization should hopefully lower the barrier to application integration.

The Web Services Vision: The Automated Web

Current web service technology does take us one step closer to completely automated web services and “just in time” application integration.



Web service Architecture

Let now examine first the individual roles of each web service actor (web service roles) and second the emerging web service protocol stack.

Web service Roles

1- *Service provider*

The service provider implements the service and makes it available on the internet.

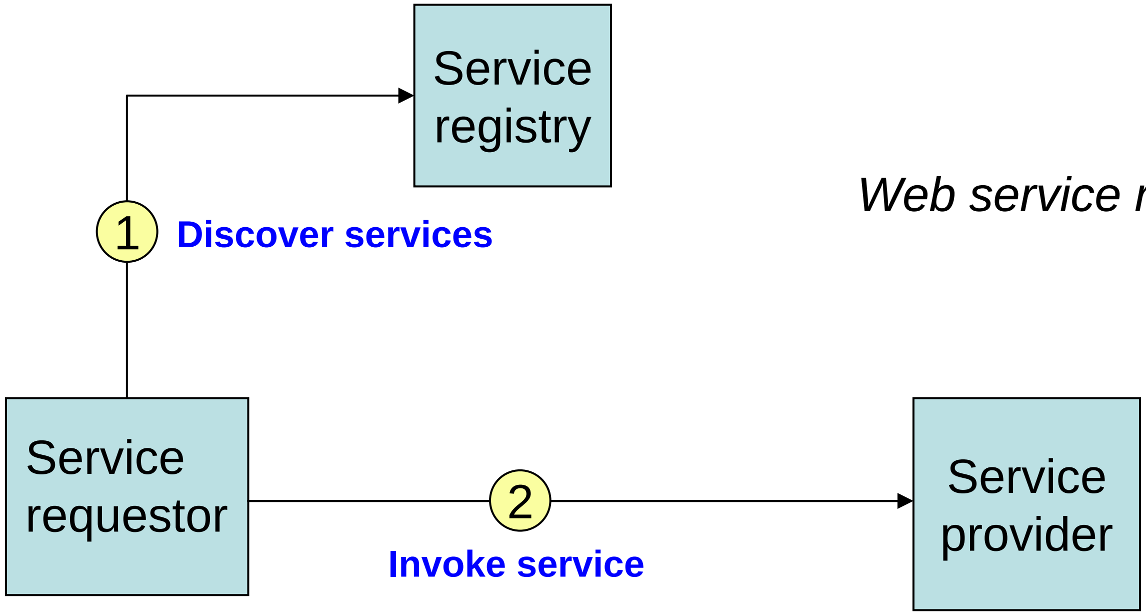
2- *Service requestor*

The requestor is any user of the web sevice who utilizes an existing web service by opening a network connection and sending an XML request.

Web service Architecture: *Web service Roles*

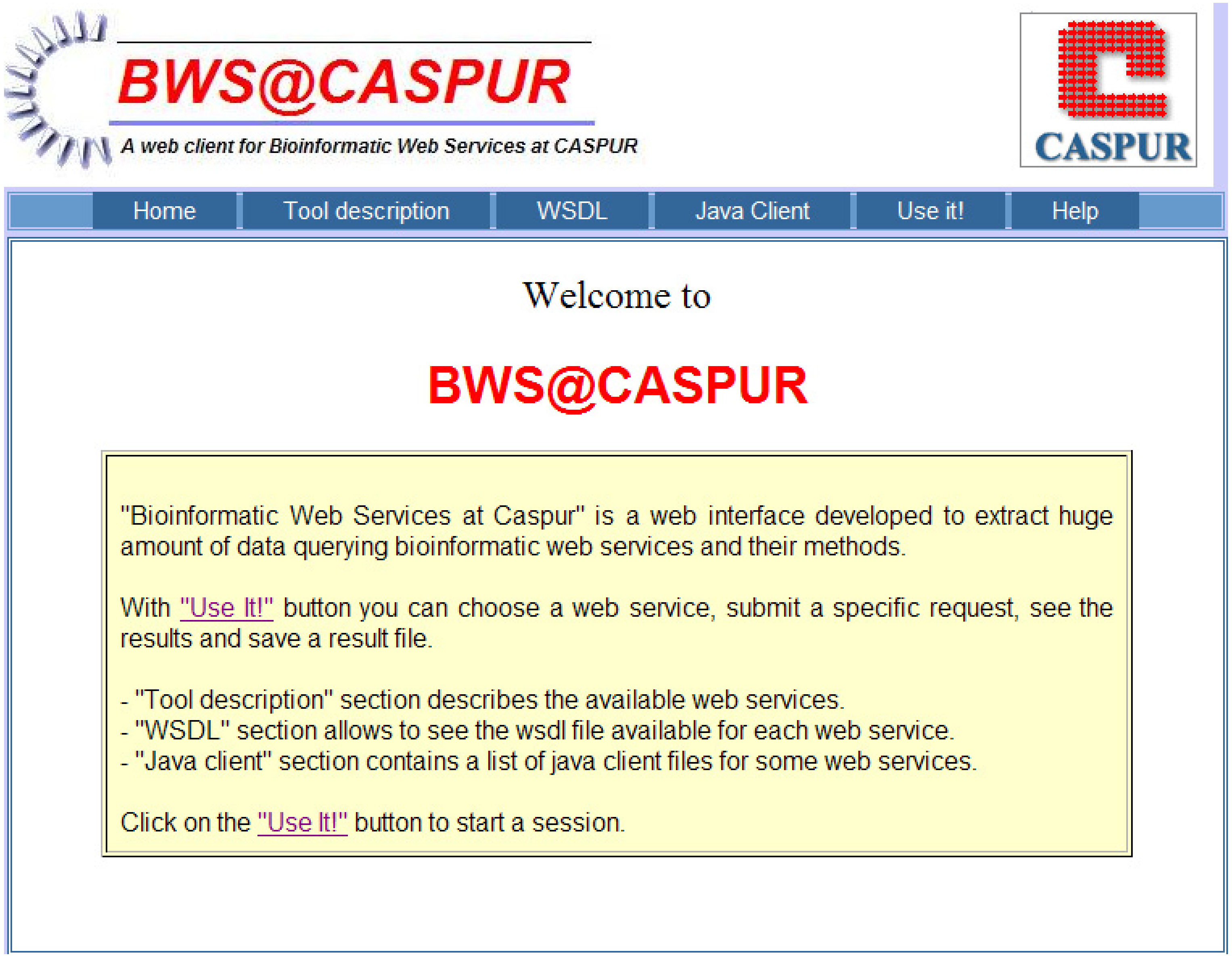
3- *Service registry*

It is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones.



Web service roles

A centralized site for CASPUR bioinformatics web services is the site:
http://t.caspur.it/webservices/home.php



Welcome to

BWS@CASPUR

"Bioinformatic Web Services at Caspur" is a web interface developed to extract huge amount of data querying bioinformatic web services and their methods.

With "Use It!" button you can choose a web service, submit a specific request, see the results and save a result file.

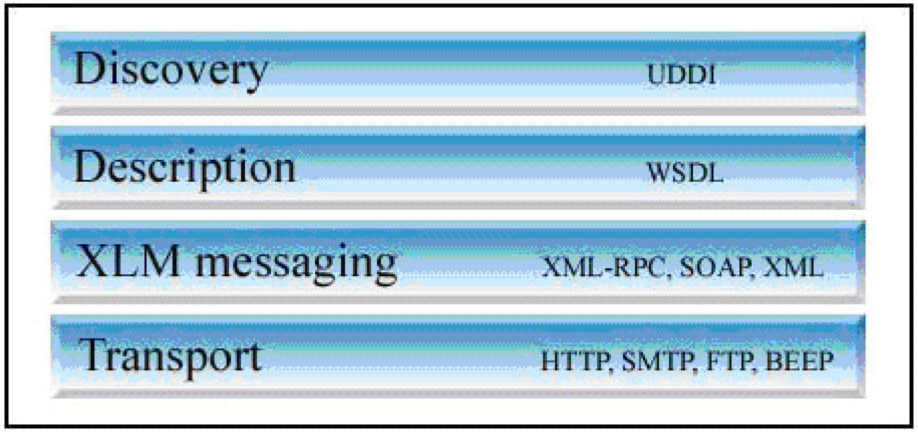
- "Tool description" section describes the available web services.
- "WSDL" section allows to see the wsdl file available for each web service.
- "Java client" section contains a list of java client files for some web services.

Click on the "Use It!" button to start a session.

Web service Architecture

Web service Protocol Stack

The web service stack is still evolving, but currently has four main layers:



Web service protocol stack

Web service Architecture: *Web service protocol stack*

CASPUR

1- *Service transport*

At this stage we simply have to transport messages between applications. This could be done by any known protocol, HTTP, FTP or even SMTP.

2- *XML Messaging*

A layer resposible for encoding messages in a common XML format, so that messages can be understood at either end. Usually the most used protocol library for various languages (e.g. Php, C, Java) to achieve this translation is SOAP.

Web service Architecture: *Web service protocol stack*

3- *Service description*

A layer necessary to describe the public interface to a specific web service. Currently, service description is handled via the ***Web Service Description Language (WSDL)***

4- *Service discovery*

Finally we centalize services into a common registry, providing easy-find functionality. Service discovery is possibly handled via UDDI (Universal Description, Discovery and Integration)

Service Request Perspective

CASPUR

A typical development plan for a service requestor is:

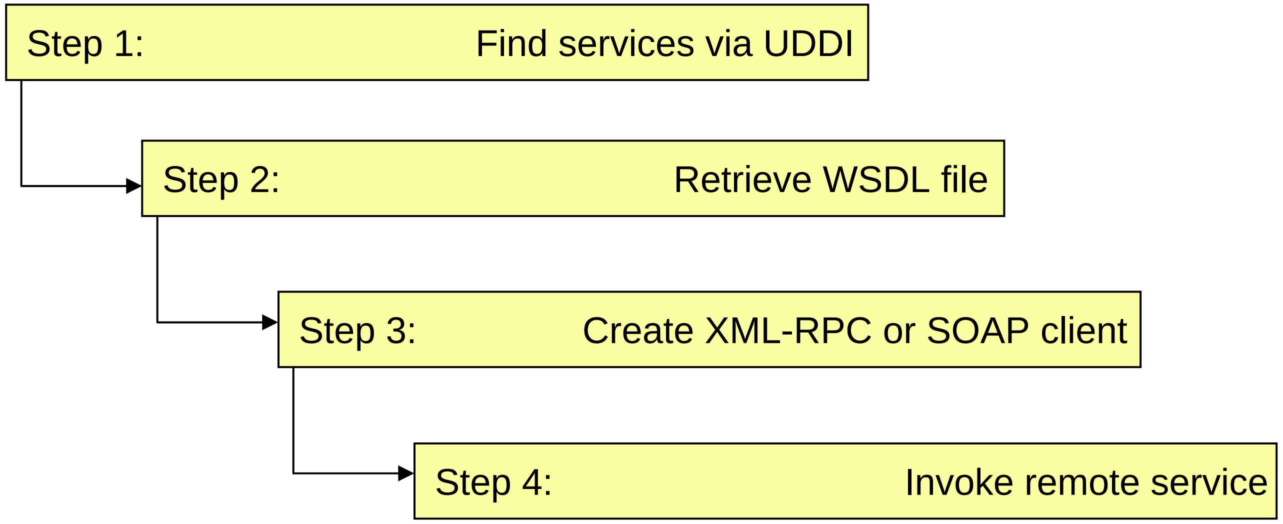
1- First, you must identify and discover thos services that are relevant to your application.

2- Once you have identified the service you want, the next step is to locate a service description.

3- You must create a client application. For example. You may create a SOAP client in the language of your choice simple analysing the WSDL file.

4- Eventually, run your client application to invoke the web service.

Service Request Perspective


Step 1: Find services via UDDI
Step 2: Retrieve WSDL file
Step 3: Create XML-RPC or SOAP client
Step 4: Invoke remote service

The service requestor perspective

Service Provider Perspective

CASPUR

A typical development plan for a service provider is:

1- First, you must develop the core functionality of your service. This is usually the hardest part, as, for eaxample, your application may connect to a database.

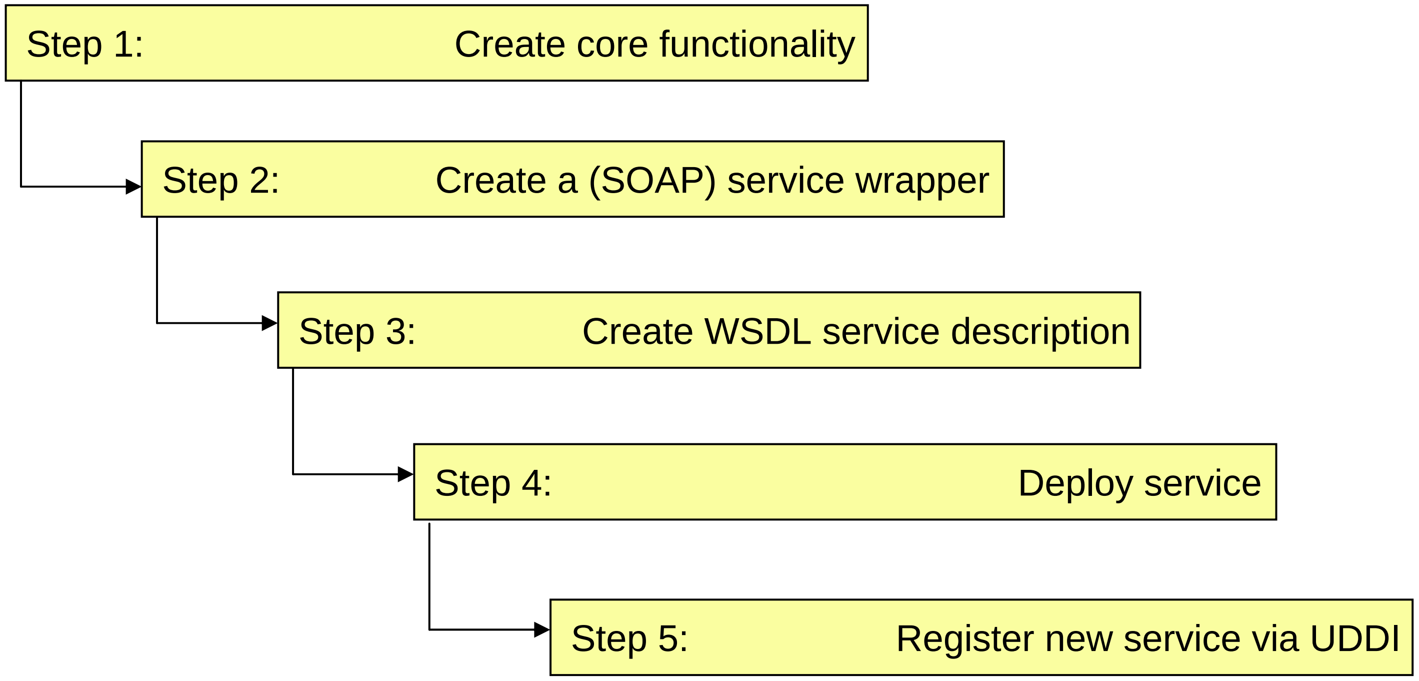
2- Second, you must develop a (XML_RPC or SOAP) service wrapper to your core functionality. This is usually a simple step.

3- Next, you should provide a service description (WSDL file for SOAP or human-readable instructions for XML-RPC).

4- You need to deploy the service: you could install or run a standalone server or integrate it with an existing one.

Service Provider Perspective

5- Fifth, you need to publish the existence and specification of your new service on a global UDDI directory or perhaps a specific UDDI directory of your istitution.


Step 1: Create core functionality
Step 2: Create a (SOAP) service wrapper
Step 3: Create WSDL service description
Step 4: Deploy service
Step 5: Register new service via UDDI

The service provider perspective

Why web service technology for Bioinformaticists ?

The online sources of biomedical data provide remarkable user interface, each different to each other. This inconvenience is disastrous for the bioinformaticists who tipically needs to aggregate data from many on-line sources to create a data set for further analysis.

When this data reside on different servers, using different data formats and access methods, the first step is to write a set of software ‘scripts’ to fetch them, reformat them and place the extract into a local database.

Why web service technology for Bioinformaticists ?

CASPUR

This is not straightforward, because most online biological databases were designed to be accessed by humans, not by machines.

Furthermore bioinformaticists often write scripts to parse HTML source to extract the data ignoring graphics links and explanatory text.

Why web service technology for Bioinformaticists ?

Problems deriving from this are several:

- database manager always change user interface adding graphics and buttons to improve user experience; each small chenge breaks dozens of scripts
- there is no documentation of what a data source's web pages are supposed to contain, so bioinformaticists must guess from few examples
- there is massive duplication of efforts

In order to facilitate universal access to bioinformatics data and analysis software, Web sevices have much to offer (see the article by Licoln Stein *Nature* 2002, **417**: 119-120).

Why web service technology for Bioinformaticists ?

CASPUR

•A number of online bioinformatic databases and services are currently available (at EBI, DNA Data Bank of Japan, Virginia Bioinformatics Institute, ecc.).

•Web services that are currently in place allow programmatic access to data.

•In a true Web services model, the data providers would register their services in a formalized service registry, and researchers' scripts would no longer need to be concerned with the interface details of the different databases.

This tutorial will guide attendees through the various components of creating Web services.

Web service code examples will be shown in Java language, because it allows the same program to be executed on multiple operating systems and it contains built-in support for using computer networks.

SOAP is an XML-based protocol for exchanging information between computers. It is an excellent technology in accessing resources from the web. This technology is the most used solution in the interoperability of bioinformatics.

By using SOAP technology, you can connect the services from programs like Java, Perl or others.

The development environment needs specific SOAP library such as Axis or SOAP::Lite, ecc., according to the language in use (e.g. Axis for Java and SOAP::Lite for Perl)

HOW TO BUILD UP YOUR OWN WEB SERVICE

We've chosen Java as our program language to develope Web Services.

Why Java?

- It uses the object-oriented programming methodology.
- It allows the same program to be executed on multiple operating systems.
- It contains built-in support for using computer-networks
- It is designed to execute code from remote sources securely.
- It should be easy to use and borrow the good parts of older object-oriented languages like C++.

Installation of Packages

By choosing Java, the most common framework to build a WS is

AXIS (together with the application server *Apache Tomcat*)

Axis is an implementation of the SOAP ("Simple Object Access Protocol") submission to W3C and is an Open Source SOAP server and client. .

Tomcat is the Java Servlet container for Implementing Java servlets and Java Server Pages.

1. Install Tomcat

You can get source from the Apache Web site (http://jakarta.apache.org/tomcat/index.html) and download the latest production version of the server (currently 5.0.16)

Also you need to install JDK 1.3 or better (currently 1.5).

2. Start Tomcat

Several environmental variables need to be set to run Tomcat

CATALINA_HOME – set to top-level directory of the Tomcat installation

JAVA_HOME – set to top-level directory of the Java installation

After Apache Tomcat installation and starting you can test it by browsing

http://localhost:8080/

3. Install AXIS

The AXIS toolkit is distributed as a collection of jar files.

To install AXIS on your server, go to http://xml.apache.org/axis/index.html and download the latest release.

AXIS most important features are:

- the implementation of SOAP 1.1/1.2
- supporting JWS (Java Web Services): gives an easy and instant deploy of Web Services implementation of WSDL, the WS descriptor
- Soap Monitor and TCP Monitor, two application written in Java to monitor SOAP net traffic

The class WebService used @ Caspur

The class WebService was built in order to manage the return type of any webservice response.

The main idea is that each webservice must extend the class WebService and use its inherited functions to return data to the client.

It is implemented in Java as an Object array and will be shown soon.

A first WebService: *a simple “hello service” class example*

```
public class HelloService extends WebService {

    public Object[] hello(String user) {
        if (user.equals(""))
            return failedCall(null,"Error: empty input");
        String results = "Hello " + user +" !!!";
        return rightCall(results);
    }

}
```

To create our WebService we save a new file **HelloService.jws**.

The name of the file must be the same of the public class defined.

public class HelloService extends WebService {..}

In our (Caspur) implementation every Web Service “extends” the WebService Class previously defined from us.

This means that our web service HelloService.jws will inherit some useful functions, such as:

-rightCall. This is the function that build the array of results in case of a successfull call on our methods of the WS.

- failedCall. A function to return an array that gives messages about tl
  error that happened in case of a failed request

To call a specific service (method) of a WS the implementation of the Java, the method must be signed as public.

In the previous example we have only one method (*hello*):

public Object[] *hello* ***(String*** *user* ***)***

The key word public means that it will be accessible from outside connections.

Object[] means that either in both cases of a failed or successfull request, this method will return an Object array.

The input of this function is expected to be a String that will be used inside with the name “user”.

```
 if (user.equals(""))
          return failedCall(null,"Error: empty input");
```

In this example, in case that the input string of a request is empty,

we force the WS to return an error message with a failedCall

```
      String results = "Hello " + user +" !!!";
      return rightCall(results);
```

Otherwise, we return a rightCall with a hello message!

This is the main idea of the data structure in return from a web service built extending our WebService class

The main array contains two arrays:

The first one (position zero) describe the status as an integer:

0 - if the request is successfull

1 – if the request has failed somewhere

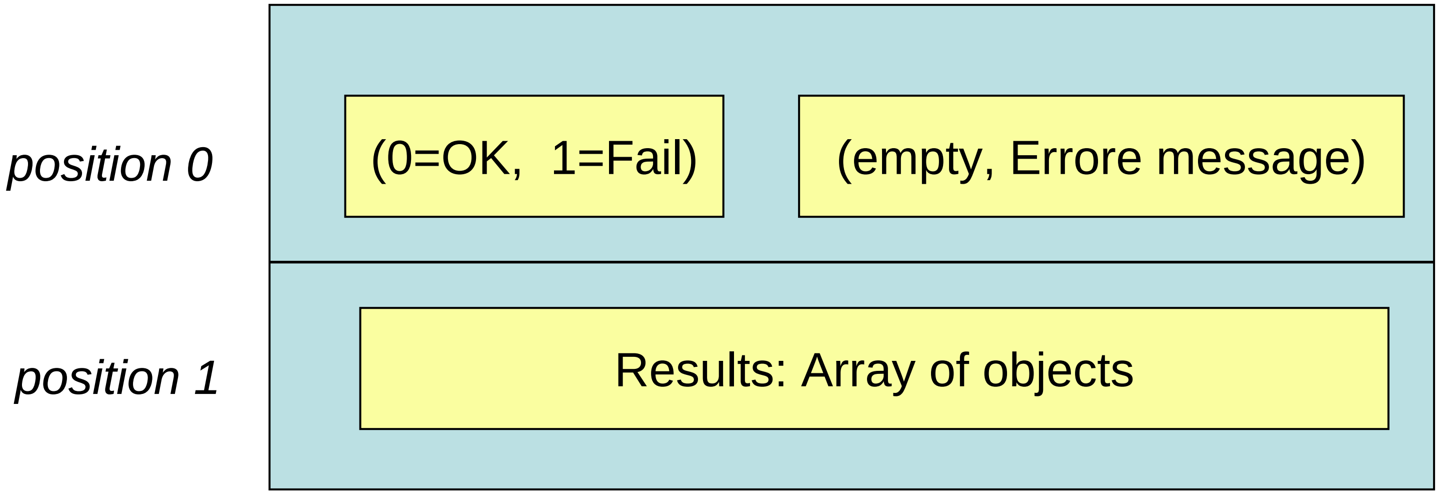
The error message, in case of a fail, is inside the same first array.

The second one (position one) contains the data from the Web Service in case of a successfull request.

Array_status (two components array, position 0)

$$\left\{ \begin{array}{l} \text{0=ok; empty} \\ \text{1=fail; error message} \end{array} \right\}$$

Array_result (array of objects, position 1)



CASPUR

For each WS which extends the Web Service class:

If the operation requested was completed succesfully the webservice returns rightCall(data);
instead if any kind of problem occurred during computation the webservice returns failedCall(data).

In this way the client, by checking the status of webservice answer, will be aware of the reliability of data returned.

This class is particularly useful when you are interested in error managing. Such problem is a foundamental task when developers work on a distributed Grid.

Example of a call of the method “hello” with input user = “Paolo”

The array returned is:

```
Array
(
    [0] => Array
        (
            [0] => 0
            [1] =>
        )
    [1] => Array
        (
            [0] => Hello Paolo !!!
        )
)
```

Example of a call of the method “hello” with input user = “”

The array returned the error as espected:

```
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => Error: empty input
        )
```

The WSDL is the file descriptor of a Web Service compiled with no errors:

The head WSDL of our HelloService would be:

```
<wsdl:definitions targetNamespace="http://t.caspur.it:8080/axis/webservices/HelloService.jws">
<!-- WSDL created by Apache Axis version: 1.2.1 -->
```

The XML is a tag-language. Every tag embed different informations.

In the WSDL “definition” field we found the correct address of the WS.

```
<wsdl:message name="helloRequest">
<wsdl:part name="user" type="xsd:string"/>
</wsdl:message>
```

this part of the WSDL specify the existence of a method “hello”

where the request takes as input “user” a string type

```
<wsdl:message name="helloResponse">
<wsdl:part name="helloReturn" type="impl:ArrayOf_xsd_anyType"/>
</wsdl:message>
```

then the response type will return an array of “any type”, an object array

EXAMPLE of a bioinformatic WS:

the usage of a BLAST code

```
public Object[] blast2sequences(String seq1, String seq2){
   try{
     String command_type=null;
     String com_type=null;
     command_type=commandType(seq1, "first sequence");
     com_type=commandType(seq2, "second sequence");
     if (!comparate(seq1,seq2))
         return failedCall(null,"Error: two sequences are different");
     String path = createFileSeq(seq1);
     String path2 = createFileSeq(seq2);
     String com = bl2seq_start+command_type+input+path+second_input+path2;
     return rightCall(exec(com));
   }catch (Exception e) {return failedCall(null, e.getMessage());}

}
```

```
public Object[] blast2sequences(String seq1, String seq2){
```

Here we define a method that will BLAST two sequences.

The BLAST is an NCBI tool that finds regions of local similarity between sequences.

The input of this public function then will be two sequences of nucleotides

or proteic atoms , seq1 and seq2.

```
command_type=commandType(seq1, "first sequence");
    com_type=commandType(seq2, "second sequence");
    if (!comparate(seq1,seq2))
        return failedCall(null,"Error: two sequences are different");
```

This part get use predefined private functions such as “commandType” and “comparate”.

The idea is that we need to be sure that the two sequences submitted in the request must be of the same biological type (nucleotides or proteine).

So if the comparation fails we send a messagge error.

```
String path = createFileSeq(seq1);
String path2 = createFileSeq(seq2);
String com = bl2seq_start+command_type+input+path+second_input+path2;
return rightCall(exec(com));
```

The BLAST tool needs data files to operate.

From the two string in input we create two files (with a private function).

Then we write the entire shell command that should usually launch BLAST from a linux command line.

Executing the command we send the string obtained as result.

Creating dynamic Clients of a Web Service using PHP.

In order to build a php client for webservices, it must be first installed the soap library for php.

Them following useWS is the function that embeds steps to communicate with a webservice:

```
function useWS ($wsdl, $method, $params) {

        $customer = new SoapClient($wsdl);
        try {
                if (!is_array($temp = $customer->_soapCall($method, $params))) {
                        display($temp);
                        return false;
                }
//check status

                if ($temp[0][0] == "0") return $temp[1];
                else {
                        print "<br>WS Error:<br><br>";
                        display($temp[0][1]); exit;
                }
        } catch (Exception $e) {
                die(""Web Services may be down, try again later");
        }
}
```

$wsdl is the variable containing the webservice's address

$method contains the name of the method

$params is the array of input parameters' names and their types

The following code use this function to perform a Web Service request:

```
$wsdl = "http://t.caspur.it:8080/axis/webservices/GeneExtract.jws?wsdl";
                    //define the address of our WebService

$method = "findGene";
                    //define the method to be called on that WS

$organism = “human";

$gene = “gata1";

$params = array("org"=>"$organism","gene"=>"$gene");
                    //build an array with paramateres related

$results = useWS($wsdl, $method, $params);
                    //Web Service call
```

If everything works, we would get for example:

```
Array
(
   [0] => 48401210
   [1] => 48408964
   [2] => f
   [3] => X
)
```

Those results are the coordinates of the chromosomal range of gene “gata1”

Possible error generated on our request, if Gene specified could not be found ($gene = “tiziana";) $results would be:

```
Array
(
   [0] => Array
      (
         [0] => 1
         [1] => Gene not found
      )
)
```

Nettab 2006

The MEPS server for identifying protein conformational epitopes

Castrignanò T (1), D'Onorio De Meo P(1), Carrabino D (1), Orsini M (2), Floris M (2) and Tramontano A (3,4)

(1) CASPUR, Consorzio Interuniversitario per le Applicazioni di Supercalcolo per Universita` e Ricerca, Roma
(2) Center for Advanced Studies, Research and Development in Sardinia (CRS4), Bioinformatics Unit, PULA (CA)
(3) Department of Biochemical Sciences, University 'La Sapienza', Roma
(4) Istituto Pasteur—Fondazione Cenci Bolognetti, University 'La Sapienza', Roma

Availability: http://www.caspur.it/meps

One of the most interesting problems in molecular immunology is epitope mapping, i.e. the identification of the regions of interaction between an antigen and an antibody.

The solution to this problem, even if approximate, would help in designing experiments to precisely map the residues involved in the interaction and could be instrumental both in designing peptides able to mimic the interacting surface of the antigen and in understanding where immunologically important regions are located in its three-dimensional structure.

We have developed a method able to find the surface region of a protein that can be effectively mimicked by a peptide, given the structure of the protein and the maximum number of side chains deemed to be required for recognition.

The method is implemented as a publicly available server.

It can also list all peptide sequences that can mimic the surface of a given protein and store them in a database.

MEPS server, available at http://www.caspur.it/meps

Web services implementation

We define here a surface ensemble as the collection of all peptides of a given length L that can position their side chains in such a way that at least m ($1<m<L$) of their side chains are able to mimic exposed regions of the protein surface.

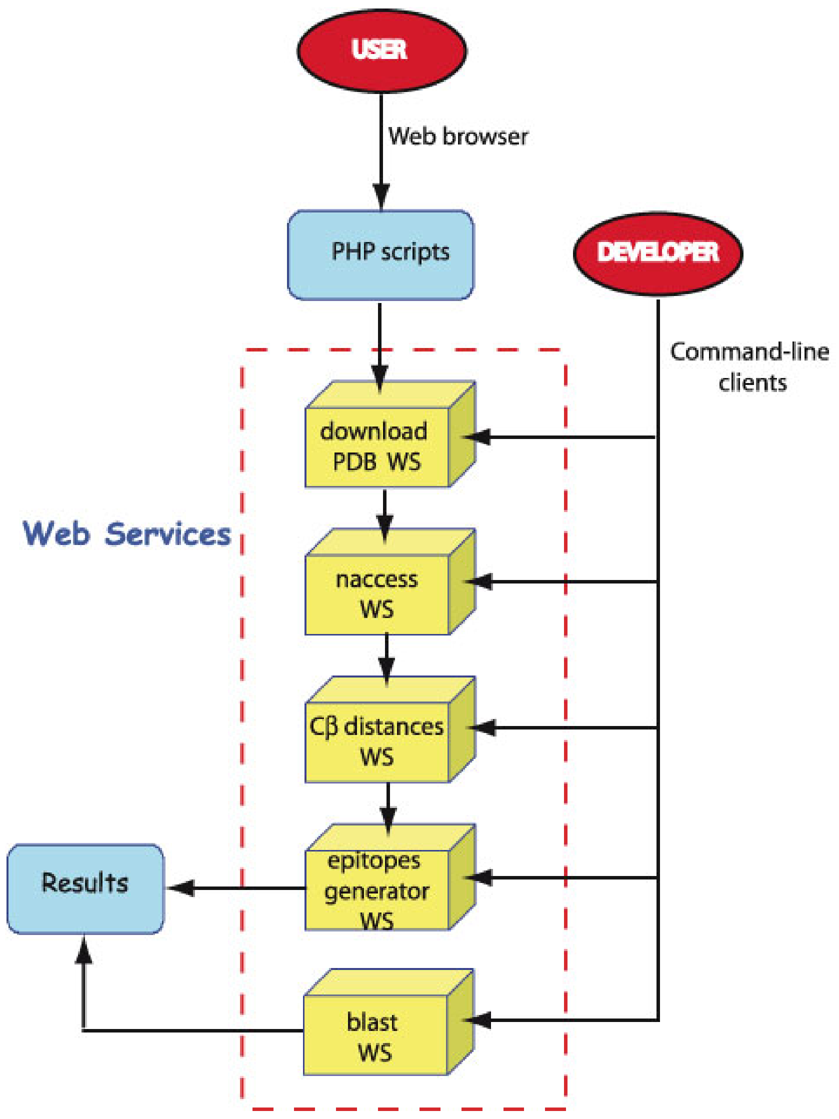
Given the structure of a target protein, we first select all solvent exposed amino acids. In the current implementation the threshold for minimum solvent accessible surface is set to 40 Å2.

Next, we compute the distance between the Cβ (Cα for glycines) of each pair of exposed amino acids and store them in a matrix.

The matrix is used to build a graph where each node represents a surface amino acid, and an edge connects two nodes if their distance is lower than a maximum distance threshold *d*.

The graph is represented as a collection of adjacency lists: there is a list for each amino acid and each list contains a pair [*neighbour*, *neigh_distance*] if *neigh_distance* is lower than *d*.

MEPS Project



CASPUR

Why Grid Computing for bioinformatics?

The explosive growth of the biological data, stimulated by genome projects, has generated a parallel development of efficient computational approaches suitable for several biological research projects. In this area the need of High Performance Computing (HPC) is growing, though usually not affordable by computational resources of a single research laboratory.

Grid computing addresses this problem by coordinating and unifying several computational resources, allowing the evaluation and mining of large amount of data in the terabyte and petabyte range.

Why Grid Computing for bioinformatics?

Unfortunately, present-day versions of Grid middleware provide only a small part of the functionality required from bioinformatics community.

On the other hand, web services are the distributed computing technology that offers powerful capabilities for scalable interoperation of heterogeneous software across a wide variety of networked platforms.

To increase individual and collective scientific productivity by making powerful information tools available to everyone, a service-oriented strategy is necessary.

Why Grid Computing for bioinformatics?

New projects on service-oriented grids have the assets of both grid and web service technology and help researchers to obtain high performance web services .

Complex applications exchanging huge amount of data, using several web services, have to be managed to gain high performance and high avalability systems, encouraging convergence of grid and web services.

A High Performance Grid Web Service framework for the identification of "Conserved Sequence Tags".

Paolo D'Onorio De Meo(1), Danilo Carrabino(1), Nico Sanna(1),
Tiziana Castrignano`(1), Giorgio Grillo(2), Flavio Licciulli(2), Sabino Liuni(2),
Matteo Re(3), Flavio Mignone(3), Graziano Pesole(2,3,*).

1) CASPUR: Supercomputing Center for University and Research,
Via dei Tizii, 6/b - 00185 Rome Italy,
2) Istituto Tecnologie Biomediche - Sezione di Bari, C.N.R., Bari, Italy,
3) University of Milan, Dipartimento di Scienze Biomolecolari e Biotecnologie,
via Celoria 26, Milan 20133, Italy

* present address: Dipartimento di Biochimica e Biologia Molecolare "E. Quagliariello",
Università di Bari, Italy

Grid Computing

CASPUR

Among service-oriented grid applications, to face the problem of identifying and assessing the coding or noncoding nature of conserved sequence tags (CSTs) through cross-species genome comparisons, we present a grid-web service framework, CSTgrid, whose core is implemented as web services.

It is composed by one grid daemon module and by seven web services, three for grid components and four for resource components.

CSTgrid web tool, available at www.caspur.it/CSTgrid.

Grid Computing: the scientific problem

The annotation of sequence features in genome tracts is a fundamental task in genome analisys. Although the complete genomes of several eukaryotic organisms have been sequenced, we are not yet able to detect their complete gene inventory, including their regulatory elements.

The identification and assessment of coding or noncoding nature of conserved sequence tags (CSTs) through cross-species genome comparisons may contribute significantly to functional annotation of whole genome sequences with the discover of novel genes or gene expression isoforms.

Grid Computing: the scientific problem

CASPUR

The computation of a coding potential score (CPS) for each CST identified in a pairwise genome comparison has been introduced, that provides a reliable classification of CSTs in coding (high CPS) and non-coding (low CPS), these latter being candidates of some regulatory activity.

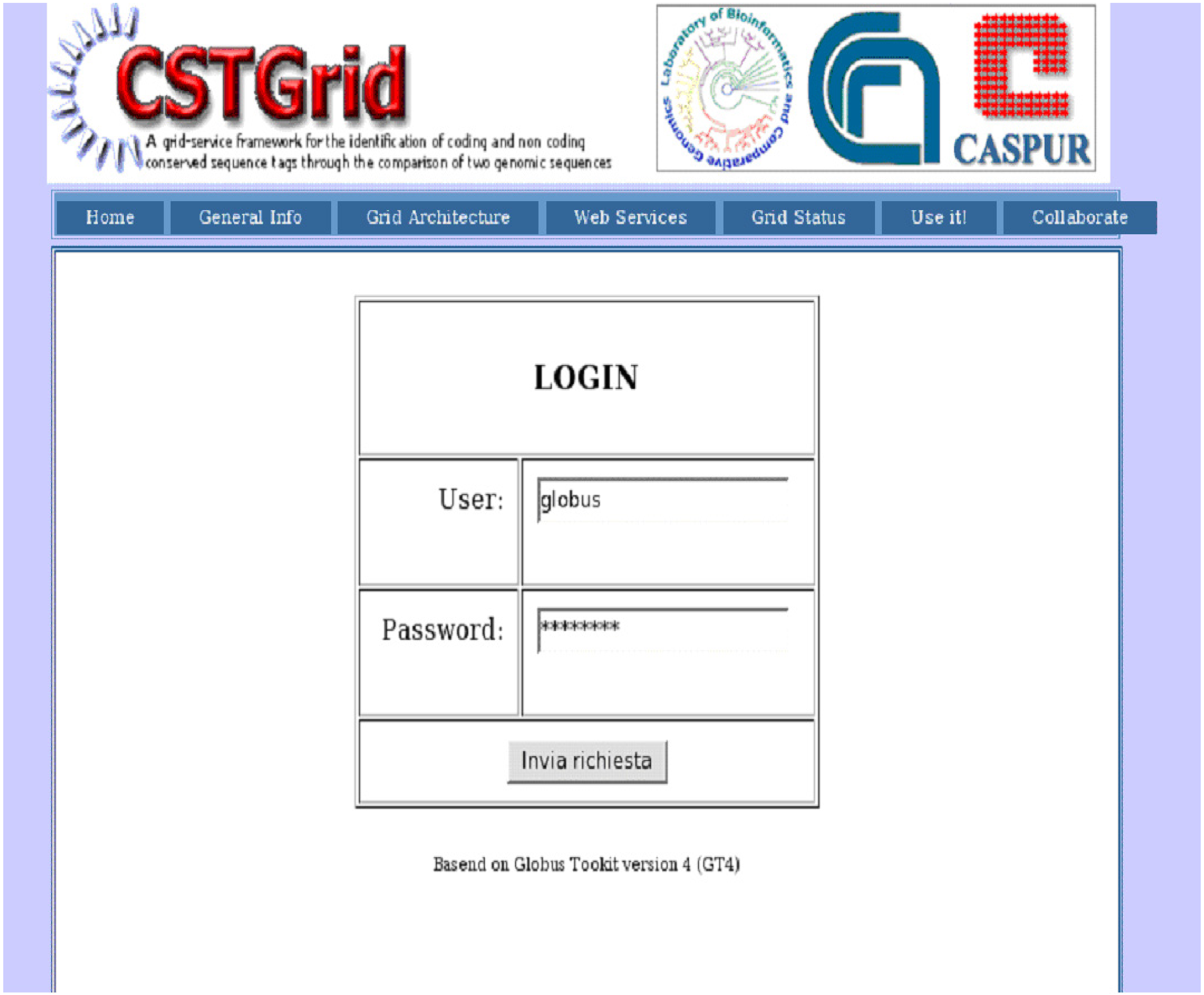
CSTgrid has been developed as an Open Grid Service Architecture, in which services act as building block of the Grid system, allowing biology community to use all services without any knowledge of the underlying infrastructure.

Grid Computing

It can provide high performance, high availability and can fairly handle hundreds of concurrent requests.

The grid infrastructure has an *ad hoc* library, implemented as a set of web services, developed meanwhile the grid community is working on a standard toolkit for service-oriented grid.

Furthermore our grid web service prototype built to minimize the overhead of standard grid toolkit (e.g. Globus toolkit), is based on grid source components developed compliant to Gtk standards, thus permitting an easy migration path to future grid service-oriented standards.


CSTGrid
A grid-service framework for the identification of coding and non coding conserved sequence tags through the comparison of two genomic sequences
Laboratory of Bioinformatics and Comparative Genomics
CASPUR
Home
General Info
Grid Architecture
Web Services
Grid Status
Use it!
Collaborate

LOGIN

User:	globus
Password:	*********

Invia richiesta

Basend on Globus Tookit version 4 (GT4)

Grid Computing

CASPUR

A set of four web services (*Gene info*, *Features*, *Seq_ret*, *CSTfinder*) has been developed allowing the user to perform a CST search in four different ways:

(i) pasting the sequences (in FASTA format),

(ii) uploading a text file containing one query sequence and one target sequence,

(iii) submitting the Ensembl gene ID and selecting the corresponding organism and

(iv) selecting the organism and choosing the chromosomal range.

The first two selection cases involve the use *CSTfinder* WS only whereas the last two involve the other three WSs needed to compose the CTSminer output.

Grid Computing

In the table we list each of four WS with a short description and the input and output streams.

Resources WS	***Description***	***Input***	***Output***
GeneInfo	Gives information about a gene (chromosome number, coordinates, strand).	An ENSEMBL identifier.	A chromosome name, a chromosomal range, a strand.
Features	Gives a list of annotated features in a chromosomal range.	An organism, a chromosome name, a chromosomal range.	A list of features.
SeqRet	Extract DNA sequence from a chromosomal range of an organism.	An organism, a chromosome name, a chromosomal range, a strand, a mask option.	A DNA sequence.
CSTfinder	Performs the search of CSTs between two DNA sequences.	Two DNA sequences.	A list of CSTs and their associated features (Coding Potential Score, Alignment, % Identity, etc.).

Both *Gene Info* and *Features* WSs query liteDB, an home-made database of features and genes annotated on genomes.

Gene Info takes a Ensembl gene name and queryies liteDB for the chromosomic coordinates of the gene.

Features takes the chromosomic coordinate and queryes liteDB for the list of annotated features. Data to populate liteDB tables are mainly extracted from UCSC and Ensembl databases, but other sources can be used.

The advantage of using liteDB is that information taken from different sources is parsed and stored with homogeneous structure. Moreover, liteDB has been designed with a very simple structure so that direct queries to the database can be performed avoiding the need for complex API.

Grid Computing

Seq_ret WS is based on a custom C program (written by F. Mignone) designed to efficiently extract genomic sequences given the organism name, the absolute genome coordinates and strand orientation (forward or reverse) of the required region. *It* has been designed keeping performance in mind; it is able to extract the selected region much faster than similar programs such as extractseq from EMBOSS package.

Grid Computing

CSTfinder represents the core of the resources and essentially implements the new version of the algorithm described in [1] with default parameters i.e. word size of 7 and maximum E-value of 10-5 for Blast analysis and minimum CST length of 60 nt. A couple of sequences is needed to run a job. CSTfinder results are displayed by scanning each detected CST with the highest-scoring triplet window (default minimum length of 60 nt).
This approach facilitates the detection of potential coding regions located in longer CSTs which might contain both coding and non-coding tracts (through the presence of untranslated mRNA or intronic regions).

[1] Castrignano T, Canali A, Grillo G, Liuni S, Mignone F, Pesole G. "CSTminer: a web tool for the identification of coding and noncoding conserved sequence tags through cross-species genome comparison". Nucleic Acids Research, 2004, vol.32 (Web Server issue):W624-7.

Data Format:

1 Ensembl gene name

2 Chromosolmal range

3 Sequence in fasta format

4 List of CSTs

5 List of annotated features

Web Service

Resources

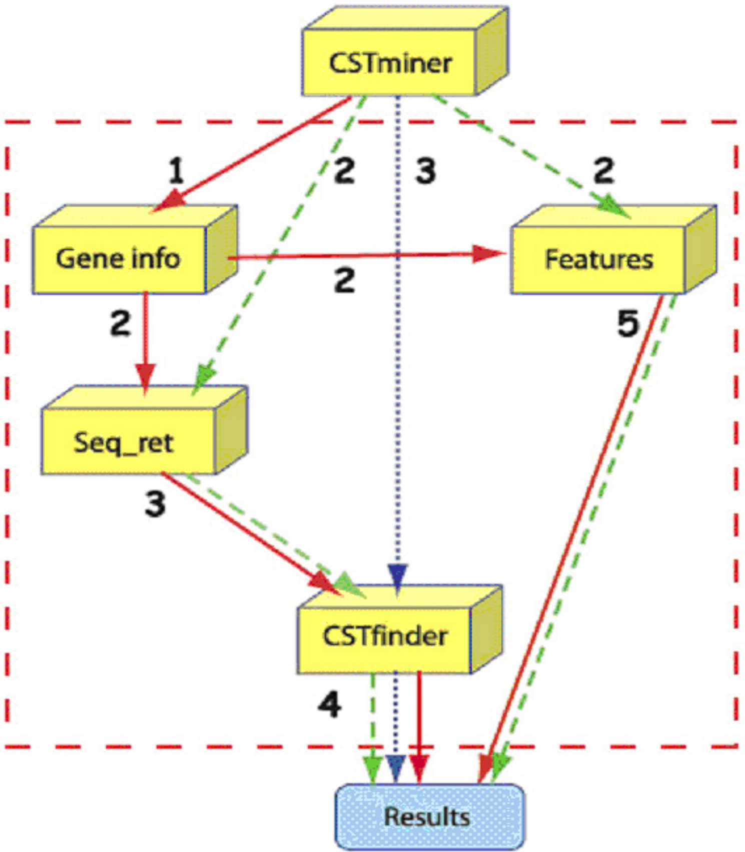
Input data selection:

Ensembl gene selection

Chromosomal parameters selection

User submitted sequence

CASPUR



The resources worklow

CASPUR

The software architecture of CSTgrid

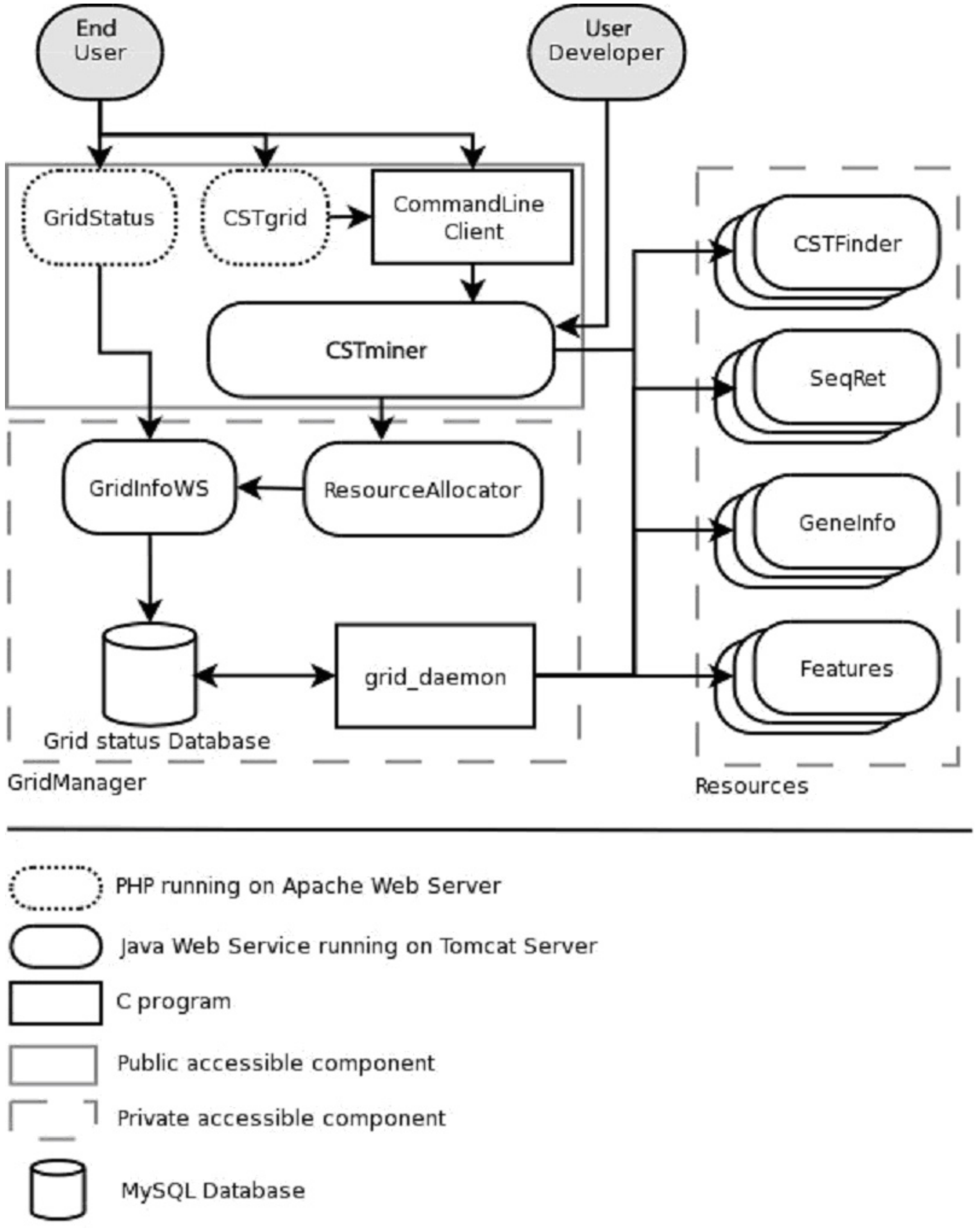
The system is developed in a multi-layered components to allow a Rapid Application Development (RAD) infrastructure and minimal administration efforts. CSTgrid is logically composed by three tiers (figure 1):

i) An *interface tier* responsible for communicating with end-user agents such as web browsers and command line clients.

ii) A generic (not oriented to search CSTs) *grid tier* composed by a grid daemon responsible for the management of the grid resources.

iii) A *resource tier* composed by a set of Resources WS, specific to search CSTs.

CASPUR


End User
User Developer
GridStatus
CSTgrid
CommandLine Client
CSTminer
CSTFinder
SeqRet
GeneInfo
Features
GridInfoWS
ResourceAllocator
grid_daemon
Grid status Database
GridManager
Resources
PHP running on Apache Web Server
Java Web Service running on Tomcat Server
C program
Public accessible component
Private accessible component
MySQL Database

The interface tier

This tier is responsible for communicating with end-user agents such as web browsers and command line clients. PHP scripts (GridStatus and CSTgrid), running under Apache, allow the user both to obtain information about the status of the grid and to launch a CST search job through a command line client. More in detail CSTgrid script inserts new requests into and fetch results from the CSTminer web service, the specific web service for managing jobs to search CSTs. CSTminer performs continuously the following steps:

* receives a request from a client;
* obtains information about free resources from the ResourceAllocator web servic
* uses several resources depending on the input request to perform CSTs search;
* sends CST results to the end-user agent.

Grid Computing

CSTminer is a public WS available to end-user developer through the standard service description layer, Web Service Description Language (WSDL), the XML grammar for specifying a public interface for a web service. Using CSTminer WSDL the end-user developer can locate the WS and invoke any of the publicy available functions from his own home-made applications.

As any WS, CSTminer can let users to create new more complex software that makes use of CSTs data through the standard web service.

The grid manager tier

The grid manager tier is based on four components: two web services (*GridInfo* and *ResourceAllocator*), one database to store information about the grid status and one grid daemon.

The database contains all the information about the hosts taking part to the grid, the services available on that hosts and the history of the availability of these services. The history data are managed by the grid daemon, a C program running in background, which periodically queries their services to know the actual status and stores this information into the database.

The detecting time interval for a given WS is calculated by the system and thus configured and stored in the database.

The grid manager tier

GridInfo is a private web service responsible of giving access to information about the grid status toward the external world via web.

GridInfo sends its data to two components:

i) the GridStatus PHP page;

ii) the ResourceAllocator web service for the managing of the resources.

ResourceAllocator is a web service responsible of taking resource requests and providing access to them according to a load-balancing failure-safe policy.

It takes up-to-date information about the grid by the *GridInfo* web service.

The grid manager tier

For CSTgrid platform, in *ResourceAllocator*, we implemented, as failure-safe policy, the Dynamic Weighted Round-Robin (DWRR) [2] for load balancing. DWRR is a variant of WRR, in which the main merit of the algorithm is to minimize the frequency of detection.

ResourceAllocator, calling the method to perform a DWRR, detects each host's load in the system at intervals and, following the detection of loads, a set of weights (the inverse ratio of host loads) is given to each host.

The system allocates new jobs to each host according to the set of weights.

[2] Li D-C, Wu C., Chang F.M. Determination of the parameters in the dynamic weighted Round-Robin method for network load balancing. Computers and Operation Research. 32 (2005) 2129-2145.

The grid enabled CSTminer

CSTminer is a web tool for the identification and characterization of genome tracts which are highly conserved across species during evolution. It is available at www.caspur.it/CSTminer.

Such a tool make use of local executables to perform CSTs search and is dynamically interconnected to Ensembl genomes.

The system was adequate for few concurrent requests, but in case of multiple concurrent requests the server performance dropped.

Furthermore, in case of a failure of some part of the distributed system, the entire application was unable to give any output.

The grid enabled CSTminer

CASPUR

These facts gave us the idea to develop a grid version of the software where each component of the system was replicated to gain better performances in case of many concurrent requests and to manage component failures.

In fact when an incoming search request is submitted according to the input selection the *ResourceAllocator* web service assigns the corresponding resources to different jobs depending on predefined policies.

The *CSTminer WS* performs the search using the *Resources WS*, located on remote machines and replicated to obtain the fault-tolerant property.

Fault tolerance

In the event of a *Resource WS* failure, searches are simply rescheduled on other available servers.

Queuing information are stored in the grid-status database possibly to preserve the trace of failure jobs. The end-user agent is also able to show the route and the history of each job.

The system also offers an interface to view the status of the grid showing a map with the distributed resources that can be selected to control their state, history, load, etc. The grid daemon is the managing component of failures.

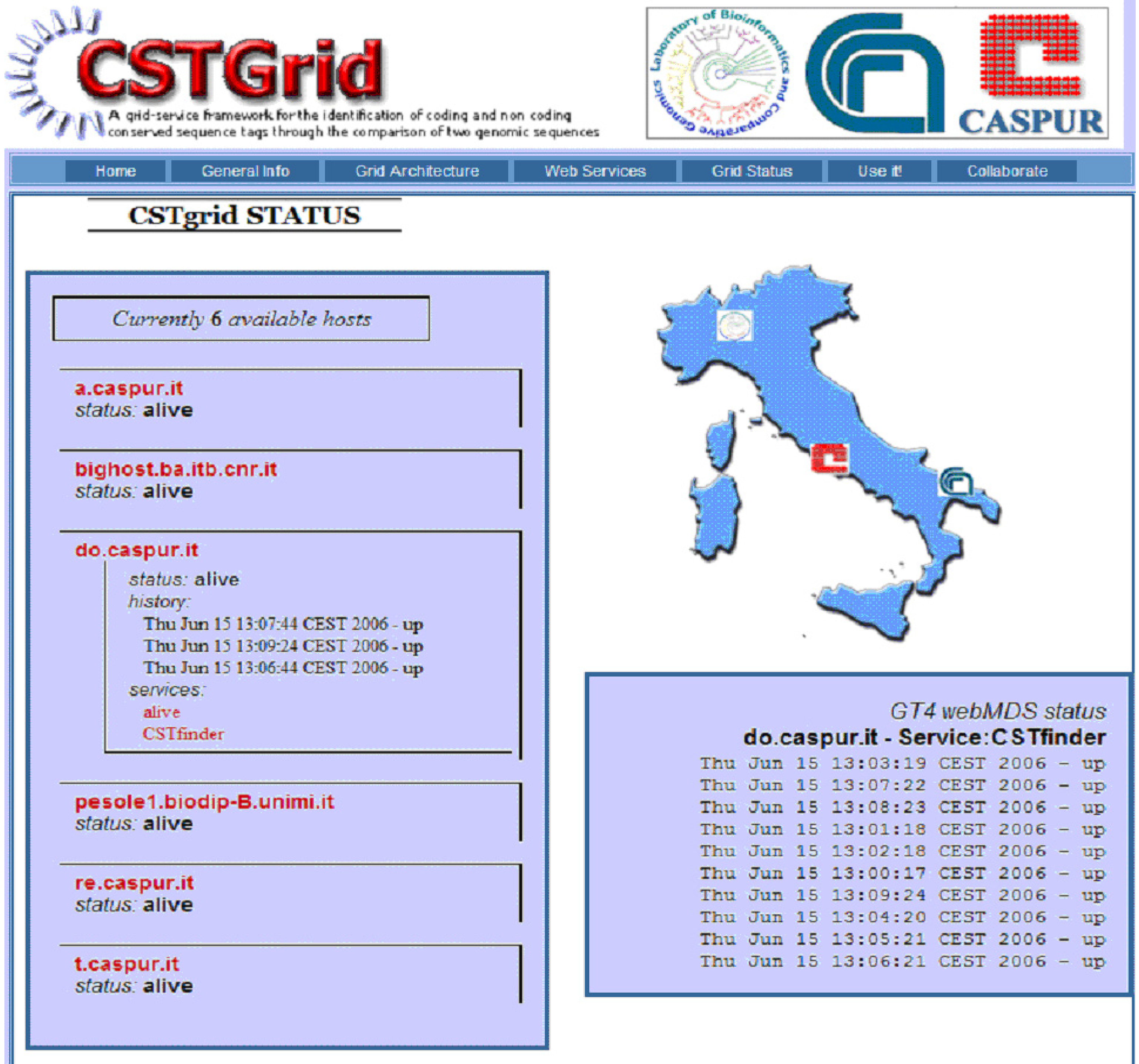
Fault tolerance

It periodically queries servers and stores information about their status in the database.

Therefore when the CSTgrid server asks for free resources the *Resource Allocator* web service, through the information stored in the database, will exclude those unavailable.

If suddenly a resource becomes unavailable while the CSTgrid server is using it, the CSTgrid server notifies the failure to the grid daemon and requests a new resource.

Nettab 2006


CSTGrid
A grid-service framework for the identification of coding and non coding conserved sequence tags through the comparison of two genomic sequences
Laboratory of Bioinformatics and Comparative Genomics
CASPUR
Home
General Info
Grid Architecture
Web Services
Grid Status
Use it!
Collaborate

CSTgrid STATUS

Currently 6 available hosts

a.caspur.it
status: **alive**

bighost.ba.itb.cnr.it
status: **alive**

do.caspur.it
status: alive
history:
Thu Jun 15 13:07:44 CEST 2006 - up
Thu Jun 15 13:09:24 CEST 2006 - up
Thu Jun 15 13:06:44 CEST 2006 - up
services:
alive
CSTfinder

pesole1.biodip-B.unimi.it
status: **alive**

re.caspur.it
status: **alive**

t.caspur.it
status: **alive**

GT4 webMDS status
do.caspur.it - Service:CSTfinder

```
Thu Jun 15 13:03:19 CEST 2006 - up
Thu Jun 15 13:07:22 CEST 2006 - up
Thu Jun 15 13:08:23 CEST 2006 - up
Thu Jun 15 13:01:18 CEST 2006 - up
Thu Jun 15 13:02:18 CEST 2006 - up
Thu Jun 15 13:00:17 CEST 2006 - up
Thu Jun 15 13:09:24 CEST 2006 - up
Thu Jun 15 13:04:20 CEST 2006 - up
Thu Jun 15 13:05:21 CEST 2006 - up
Thu Jun 15 13:06:21 CEST 2006 - up
```

Conclusion

CSTgrid architecture is highly modular allowing an easier development and debugging process.

The system has been developed as a Service-Oriented Architecture based on a collection of web services distributed over a geographical grid.

It deploys an interface layer, completely unaware of underneath grid-layer.

The system has been designed in a user-centric way providing two points of access:
the first one is for end-user to perform hight-performance CST serches;
the second one for the developer user to build new large scale WS applications.