SERVICE GRID FOR BUSINESS COMPUTING

Zongwei Luo[1], Jia Zhang[2a] and Rosa M. Badia[3]

E-business Technology Institute[1]

The University of Hong Kong, Pokfulam

Pokfulam Road, Hong Kong

Department of Computer Science[2]

Northern Illinois University

DeKalb, Illinois 60115, USA

CEPBA-IBM Research Institute[3]

Dept. Computer Architecture

Universitat Politecnica Catalunya

Campus Nord - Modul D6

c/Jordi Girona 1-3

08034 – Barcelona, Spain

1 Introduction

In this chapter, we will introduce an advanced topic of grid computing – Web services-oriented grid for business computing. Web services represent a new concept of computing that enables diverse and distributed resources to communicate with each other based upon a set of standards. We will discuss what values the marriage of Web services and grid computing can bring to business computing and the current state of the art of the field. The layout of the chapter is as follows:

- Grid computing overview
- Web services orientation
- Service oriented grid
- Integrated service platform for integration

[a] The second author is also a Guest Researcher at National Institute of Standards and Technology (NIST).

- Challenges and considerations

2 Grid computing overview

Maybe one of the most essential reasons why Grid computing is considered as the next wave of standardization of business computing is that it offers a promising framework to support universal resource sharing and reuse. It should be noted that the term *resources* here refers to any broad-sense computing resources or so-called IT resources that are geographically distributed and owned by different organizations, which include data sources, supercomputers, storage systems, databases, and practically all applications and peripherals. By establishing a Grid network in which contained resources are universally accessible, Grid computing enables applications to share computing resources, allocate resources as needed, and dynamically re-pool resources when the supply and demand change.

With the support of resource sharing, the grid-based computing can be constructed incrementally. An application can be composed of small, standard, distributed, and interchangeable components in the grid network. Organizations can start small, gain experience, and as they are successful, continue to build larger-scale applications. When a component needs to be updated, the original system does not need to be ripped; instead, a new component will be located from the grid network and replace the corresponding old component without interfering with the whole system.

The key strategy that grid computing realizes resource sharing is the separation of the association between resources from the application systems. For example, considering adopting grid computing to construct environmental data simulation software that contains numerous amount of modeling data, the hard-coded association of data resources in the traditional system is managed by a Geographical Information System (GIS) like application. Grid shields off the data source and breaks the hard links between the data source and the GIS application system. Moreover, instead of duplicating GIS data to each computer with GIS application installed, GIS data is “virtualized” into a central pool and allocated or provisioned to GIS applications when they need it.

Thinking in large, resources can be further consolidated into distributed larger resource pools managed by associated processing centers. Dynamic provisioning of computing servers, software applications, and data reserves can be effectively supported and scaled up in stages.

Further, although some scientific and industrial applications are important enough to warrant the use of dedicated high-end computers, a much larger body of applications can benefit from the enhanced distributed computing capabilities powered by the managed heterogeneous resources through Grids. Grid computing is thus becoming a critical infrastructure for science, business, and industry by providing a resource virtualization layer. This technology will certainly result in a new programming model eventually reshaping the business applications development process.

2.1 Grid technology – resource sharing perspective

In order to enable resource sharing under Grid environment, dedicated Grid resource brokers are commonly adopted. These Grid resource brokers are responsible of managing and scheduling resources and application execution depending upon resource consumers' and owners' requirements, When the availability of resources changes, the Grid resource brokers will dynamically relocate and reschedule resources accordingly. In this section, we will discuss several well-known Grid resource brokers.

2.1.1 The Nimrod-G Grid resource broker

(http://www.globus.org/research/applications/nimrod.html)

The Nimrod-G Grid resource broker is a computational economy-based global Grid resource management and scheduling system. It supports deadline- and budget-constrained algorithms for scheduling applications on distributed resources. In detail, the Nimrod-G Grid resource broker performs resource discovery and scheduling, dispatches jobs to appropriate remote Grid nodes, starts and manages job execution, and gathers results back to the home node. The Nimrod-G broker leverages services provided by different lower-level Grid middleware solutions to perform resource discovery, trading, and deployment of jobs on

Grid resources. The broker system consists of four main components: (1) a Task Farming Engine (TFE) that supports job management protocols and APIs., (2) a Scheduler that performs resource discovery, trading, and scheduling, (3) a Dispatcher and Actuator that deploy agents on Grid resources, and (4) agents for managing the execution of jobs on resources.

2.1.2 Condor-G resource broker

(http://www.globus.org/retreat00/presentations/miron-08-00)

Condor-G resource broker combines the inter-domain resource management protocols of the Globus Toolkit (an open source software toolkit used for building Grid, available at: http://www-unix.globus.org/toolkit) and the intra-domain resource management methods of Condor (a highly distributed batch system for job scheduling and resource management, available at http://www.cs.wisc.edu/pkilab/condor) to allow a user to harness multi-domain resources as if they all belong to her personal domain. Condor uses Globus to provide underlying software needed to utilize Grid resources, such as authentication, remote program execution, data transfer through remote resource access, computation management, and remote execution environments:

- Remote resource access issues are addressed by requiring standard protocols for resource discovery and management. These protocols defined by Globus toolkit support secure discovery of remote resource configuration and state, secure allocation of remote computational resources, and management of computation on those resources.
- Computation management issues are addressed via the introduction of a robust and multi-functional user computation management agent that is responsible for resource discovery, job submission, job management, and error recovery.
- Remote execution environment issues are addressed via the use of mobile sandboxing technology that allows a user to create a tailored execution environment on a remote node.

2.1.3 Platform LSF (Load Sharing Facility)

(http://www.platform.com/products/LSF)

Platform LSF (Load Sharing Facility) is a workload management solution and software that optimizes the use of enterprise-wide resources for compute- and data-intensive applications. Users are granted transparent and on-demand access to valuable computing resources. Platform LSF fully utilizes IT resources to ensure policy-driven and prioritized service levels for always-on access to resources. Based on the production-proven, open, and grid-enabling Virtual Execution Machine (VEM) architecture that sets the benchmark for performance and scalability across heterogeneous environments, Platform LSF is the leading commercial solution for production-quality workload management in cluster computing environment.

2.1.4 Adaptive resource broker

(http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/096.pdf)

The Adaptive Resource Broker targets to automate job migration when the status of a resource changes and enable user to change a job attribute during run-time. The user will be able to view application status, resources used by the application, predicted completion time, and so on. The aim of the Adaptive Grid Resource Brokering is to address the concerns relating to dynamic adaptability without affecting existing user applications. The Adaptive Grid Resource Brokering leverages the reflective technique to separate concerns for functional and non-functional behavior of a program, resulting in some code that is highly reusable and extensible.

The broker comprises basic components that implement resource discovery and selection, dispatching, adapter management, and monitoring. An Adapter Manager controls migration, which is supported by job monitoring and enabled by rescheduling and check-pointing. The broker gathers dynamic information about the resources (e.g., accessibility, system workload, performance) during the runtime. Dynamic information (e.g., performance slowdown, target system failure, job cancellation) is reported to a monitor. The monitor provides predictive information to the Adapter Manager, which uses this information to make a decision as to whether job migration is required. The main task of the Adapter Manager is to ensure that the job requirements are fulfilled.

2.1.5 EZ-Grid

(http://www2.cs.uh.edu/~ezgrid)

EZ-Grid is a resource brokerage system coupled with user interfaces and robust information objects for multi-site Grid computing. The on-going EZ-Grid project aims at promoting efficient job execution and controlled resource sharing across sites. To realize the goal, EZ-Grid constructs a resource brokerage system composed of policy frameworks and information subsystems. The complexity in Grids is eliminated by providing services such as automatic resource discovery, assistance of preparing users' jobs by profiling the application, efficient resource usage through brokering, and transparent file transfer between Grid resources. For administrators, EZ-Grid provides a policy framework to help them enforce fine-grained usage of policy-based authorization for the users of their resources.

2.2 Grid technology – parallel exploitation

The concept of parallel programming has been extensively used by the scientific community as a means for speeding up scientific applications. By running these applications in large High Performance Computing (HPC) facilities, the scientific community is able to overcome scientific challenges that were impossible to imagine some years ago. The most well-known paradigms for parallel programming are message passing and shared memory. The Message Passing Interface (MPI) standard [1] has been extensively used by the scientific community in the recent years. Similarly, OpenMP [2] can be used as an API for shared memory architectures.

In the last years, the Grid has entered the scene as a new computing alternative. It is characterized for having its resources geographically distributed in a large distant area. The set of problems tackled by the Grid scientific community is very large, and most of them are not directly related with computation intensive applications. However, the possibility of using all the computing elements connected by the Grid

as a unique and very large supercomputer is highly attractive. Similar to parallel programming, programming models for the Grid can become crucial to ensure a successful history. Between the Grid-enabled MPI approaches we find PACX-MPI [3] and MPICH-G2 [4], among others.

Other known approaches that enable programming parallel applications for computational Grids are GAF4J, HOCs, Satin, the GriPhyN Virtual Data System, and GRID superscalar. GAF4J [5] is a programming environment where applications are described in Java. This framework aids in writing Java applications with multi-threaded logic, so that the threaded tasks are distributed for execution over a Grid instead of having multiple threads started on the same node. However, the user has to explicitly program with threads. Higher-Order Components (HOCs) [6] is a component-oriented approach based on a master-worker schema. It expresses recurring patterns of parallelism that are provided to users as program building blocks, pre-packaged with distributed implementations.

Satin [7] is a Java-based programming model for the Grid, which allows explicit expression of divide-and-conquer parallelism. Satin uses marker interfaces to indicate that certain invocations method need to be considered for potentially parallel (spawned) execution. Moreover, synchronization is explicitly marked whenever it is required to wait for the results of parallel method invocations.

The GriPhyN Virtual Data System [8] based on workflows. The input to the system is an abstract workflow, which is automatically mapped to a concrete workflow. It is based on Chimera and Pegasus tools. In this approach, a graph describing the workflow has to be completely specified using Virtual Data Language (VDL) that is a textual input language.

GRID superscalar [9] is a programming model for the Grid, with objectives of providing a simple programming interface to general scientists and providing a powerful runtime capable of parallelizing the application and concurrently running different parts of the application in different computing elements of

the Grid. The types of applications that can benefit from using GRID superscalar are those composed of several calls to one or more coarse-grain functions that can be executed independently. Examples of these applications are Montecarlo simulations, parametric studies, and complex workflows where each element of the workflow is a computational intensive task. Such kinds of applications exist in different scientific fields such as bioinformatics, computational chemistry, astrophysics and others.

The similarity of Satin, the GriPhyN Virtual Data System, and GRID superscalar is that they exploit the applications' parallelism. The rest of this section will focus on discussing these three programming models.

2.2.1 Satin

Satin [7, 10, 11] is a system for running divide-and-conquer programs on distributed memory systems and on wide-area metacomputing systems. In Satin, single-threaded Java programs are parallelized by annotating methods that can run in parallel. Satin uses so-called marker interfaces to indicate that a certain method invocation needs to be considered for potentially parallel (so-called spawned) execution. Parallelism is achieved in Satin by running different spawned method invocations on different machines.

Programs in Satin express a potential parallelism with a special interface, *satin.Spawnable*, and a class *satin.SatinObject*. All methods defined in a spawnable interface are marked to be spawned rather than executed normally. The *sync* method from the class *satin.SatinObject* allows to explicitly declare synchronization points wherever required.

Figure 1 shows a sample application programmed in Satin. This code fragment calculates Fibonacci numbers, and is a typical example of a divide-and-conquer program. The work is split up into two pieces, *fib(n-1)* and *fib(n-2)*, which are then recursively solved. Splitting the problem into smaller subproblems will continue until it can no longer be split up. At that point, the answer for the subproblem is returned. Then, two solved subproblems are combined, in this case, by adding the results.

In the code segment as shown in Figure 1, an interface *FibInter* is implemented which extends *satin.Spawnable*. The fib method defined in this interface is marked to be spawned. Also, the class Fib extends *satin.SatinObject* and implements *FibInter*. From *satin.SatinObject* it inherits the *sync* method; from *FibInter* the spawned *fib* method. Finally, the invoking method (in this case main) simply calls *Fib* and uses *sync* to wait for the result of the parallel computation.

Satin's byte code rewriter generates the necessary code. A spawned method invocation is put into a local work queue. From the queue, the method might be transferred to a different CPU where it may run concurrently with the method that executes the spawned method. A new thread is started for running each of these methods. The *sync* method waits until all spawned calls in the current method invocation are finished; the return values of spawned method invocations are undefined until a *sync* is reached.

Spawned method invocations are distributed across the processors by work stealing from the work queues mentioned above. Cluster-aware Random Stealing algorithm, a specifically designed algorithm for cluster-based and wide-area (Grid computing) systems is used in the runtime. Experiments in real grid environments demonstrate that this algorithm is more efficient than traditional random steal algorithms.

```
interface FibInter extends satin.Spawnable {
        public long fib (long n);
}
class Fib extends satin.SatinObject {
        implements FibInter {
        public long fib (long n) {
                if (n<2) return n;

                long x = fib (n-1); // spawned
                long y = fib (n-2); // spawned
                sync;

                return x+y;
        }

        public static void main (String[] args) {
                Fib f = new  Fib();
                long res  = f.fib(10);
                f.sync();
```

```
            System.out.println ("Fib 10 = " + res);
    }
```

Figure 1: Fibonacci numbers programmed in Satin

2.2.2 GriPhyN Virtual Data System (VDS)

The GriPhyN Virtual Data System (VDS) [8] consists of Chimera [12], Pegasus [13] and DAGMan [14]. It allows explicitly specifying data workflows with textual data flow graphs. The workflows are described using Virtual Data Language (VDL). Figure 2 shows an example of workflow described in Chimera's VDL.

```
TR preprocess( output b[], input a ) {
        argument = "-a top -T60"; argument = " -i "${input:a}; argument = " -o "${output:b}; }
TR findrange( output b, input a2, input a1, none name="findrange", none p="0.0" ) {
        argument arg = "-a "${none:name} " -T60"; argument = " -i "${input:a1}" "${input:a2};
        argument = " -o "${output:b}; argument = " -p "${none:p}; }
TR analyze( output b, input a[] ) {
        argument arg = "-a bottom -T60"; argument = " -i "${input:a}; argument = " -o "${output:b};}
DV top->preprocess( b=[ @{output:"f.b1":"true"}, @{output:"f.b2":"true"} ], a=@{input:"f.a"} );
DV left->findrange(b=@{output:"f.c1":"true"}, a2=@{input:"f.b2":"true"}, a1=@{input:"f.b1":"true"},
        name="left", p="0.5" );
DV right->findrange( b=@{output:"f.c2":"true"}, a2=@{input:"f.b2":"true"}, a1=@{input:"f.b1":"true"},
        name="right", p="1.0" );
DV bottom->analyze( b=@{output:"f.d"}, a=[ @{input:"f.c1"}, @{input:"f.c2"} ] );
```

Figure 2: Sample workflow description in VDL

Two types of instances are possible in VDL: transformations (TR) and derivations (DV). The transformations are interface definitions of functions (nodes of the workflow) including the definition of the expected parameters and default arguments to call the functions. The derivations are instantiations of the derivations that include the logical names of the parameters. From VDL descriptions, Chimera generates an abstract workflow representation in DAX format (Data Acyclic Graph described in XML). Chimera adds a vertex in the DAX for each derivation and an edge between the vertex producing an output files and each consumer vertex.

Pegasus maps the abstract workflow to a concrete form, which is finally executed by DAGMan. To produce the concrete DAG, Pegasus relies on different Grid information services. From the logical filenames referenced in the abstract workflow, the Globus Replica Location (RLS) service is used to obtain

physical file locations. To find the location of the binaries that implement the transformations, Pegasus queries the Transformation Catalog (TC). Finally, Pegasus queries the Globus Monitoring and Discovery Service (MDS) to find the available resources. The output of the TC and the MDS are combined to make scheduling decisions. Additionally, Pegasus queries the MDS to find information about the *gridftp* servers available for data movement, job managers that can schedule jobs, and storage locations, Further, the DAGMan engine acts as a metascheduler that is able to exploit the parallelism of the graph when possible.

2.2.3 GRID superscalar

GRID superscalar [9, 15] is a Grid programming environment that allows to simply program applications that will be efficiently run on a computational Grid. GRID superscalar is able to parallelize, at runtime and at task level, a sequential application and execute the application in a computational Grid. The used approach is able to take benefit from those applications that are composed of coarse-grained tasks. These tasks can be of the size of a simulation, a program, a solver, etc. These kinds of applications are very common in bioinformatics, computational chemistry, and other scientific fields.

The process of generating and executing an application with GRID superscalar is composed of a static phase and a dynamic phase. In the static phase, a set of tools for automatic code generation, deployment and configuration checking are used. In the dynamic phase, the GRID superscalar runtime library and basic Globus Toolkit 2.4 services are used.

The application developer provides a sequential program, composed of a main program and application tasks, together with an Interface Definition Language (IDL) file that identifies the coarse-grain tasks. The syntax of the IDL file, which is borrowed from CORBA IDL, is shown in Figure 3. For each task, the list of parameters is specified, indicating the type and if it is an input, output or input/output parameter.

```
interface OPT {
        void filter(in File referenceCFG, in double latency, in double bandwidth,
                out File newCFG);
```

```
        void dimemas_funct (in File cfgFile, in File traceFile, out File DimemasOUT);
        void extract(in File cfgFile, in File DimemasOUT, inout File resultFile);
};
```

Figure 3: Sample GRID superscalar IDL file

Additionally to this IDL file, the programmer should introduce minor changes in the main program and application tasks. In the main program, GRID superscalar primitives *GS_On* and *GS_Off* are called for initialization and termination. Calls to the C library for opening or closing files are substituted by calls to the GRID superscalar primitives: *GS_FOpen*, *GS_FClose*, *GS_Open* and *GS_Close*. In the application tasks code the only change required is the use of the GRID superscalar primitive *GS_system* instead of the C system primitive, in case it is used.

In the static phase, two sets of files are automatically generated by the GRID superscalar tools (*gsstubgen*) that can be used to build an application ready to run in the Grid (see Figure 4). With these two sets of files a client-server based application is built, which has the same functional behavior of the initial user application. The client binary is run in the localhost and submits calls to the server binaries in remote hosts of the computational Grid. The server workers will only execute the functionality of the tasks listed in the IDL file. The main program (client) executed in the local host will execute the rest of the application code.

In the example of Figure 4, the files provided by the user are *app.idl*, *app.c* (the main program code) and *app-functions.c* (the application tasks code). From the IDL file (*app.idl*) the rest of files shown in the figure are generated. From these files, the most relevant are *app-stubs.c* and *app-worker.c*. The *app-stubs.c* file contains wrapper functions for the application tasks, which will be called in the client-side instead of the original tasks. The *app-worker.c* contains the main program of the server-side.

Still in the static phase, a graphical interface is provided to help users set the Grid configuration and deploy the application (Deployer Center). After this step the application is ready to be run in a computational Grid.

In the dynamic phase, the application is started in the localhost the same way it would have been started originally. While the functional behavior of the application will be the same, the basic difference is the fact that the GRID superscalar library will exploit the inherent parallelism of the application at the coarse-grain task level and execute these tasks independently in remote hosts of the computational Grid.

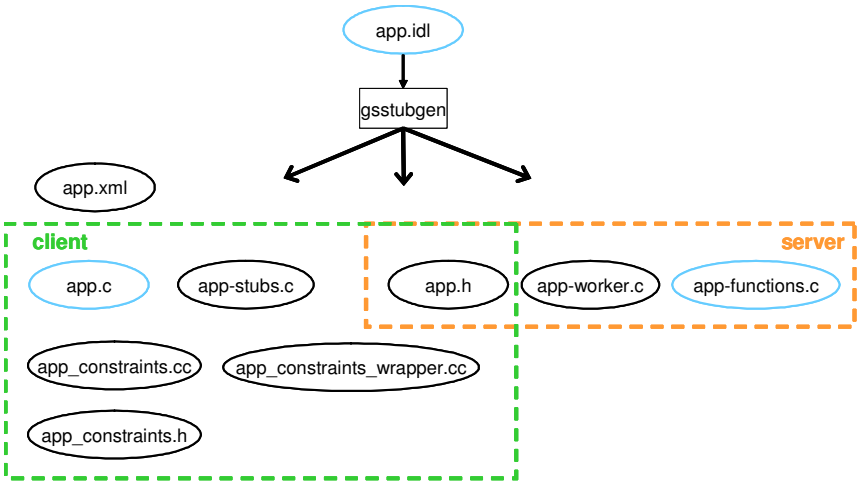

Figure 4: Code generation in GRID superscalar

To exploit the parallelism of the application, the GRID superscalar runtime builds a data dependence graph where each node of the graph represents one of the coarse-grain tasks of the application. The edges between the nodes of the graph represent file dependences between those tasks. By file dependences it refers to data dependences due to files that are read or written by the tasks. In this sense, a task that writes a given file should be executed before another that reads this file. Therefore, and edge from the first task to the second will exist in the graph. These edges will define a relative execution order that should be respected. From this task graph, the GRID superscalar runtime is able to exploit the parallelism, by sending tasks that do not have any dependence between them to remote hosts in the Grid (see Figure 5).

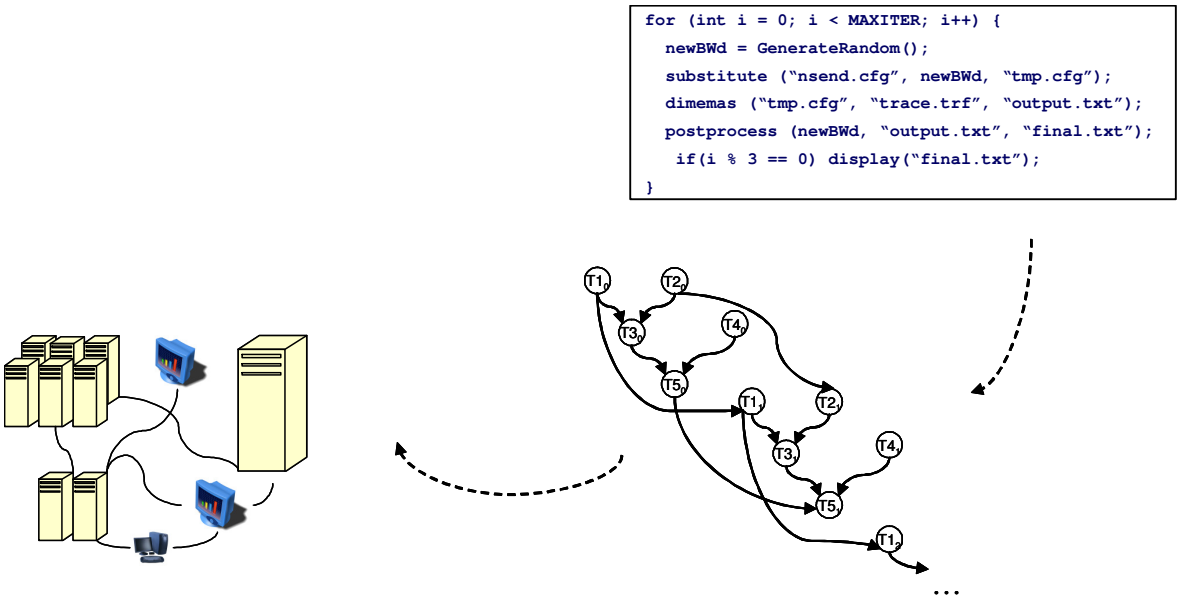

Figure5: Overview of GRID superscalar behavior

In each case, GRID superscalar broker will select among the set of available hosts, which is the best suited. This selection favors reducing the total execution time, which is computed not only as the estimated execution time of the task in the host but also the time that will be spent to transfer all files required by the task to the host. This allocation policy exploits the file locality, reducing the total number of file transfers.

All Grid related actions (file transfer, job submission, end of task detection, results collection) is done totally transparent to the user. For each task, the GRID superscalar runtime transfers the required files from their current locations to the selected host, submits the task for execution, and detects when the task has finished. At the end of the execution of one task, the data dependence graph is updated. As a result, a resource is now free and any ready task may be submitted for execution in this resource and the dependant tasks on the finished task may become ready for execution.

When the application has finished, all working directories in the remote hosts are cleaned, application output files are collected and copied to their final locations in the localhost and all is left as if the original application has been run.

Summarizing, from the user point of view, an execution of a GRID superscalar application looks as an application that has run locally, with the exception that the execution has been hopefully much faster by using the Grid resources.

2.3 Challenges to Grid Computing

Although the grid computing allows the entire collection of resources to be viewed as a seamless and synergistic information processing system that is accessible globally, it is not without penalty since the value added by employing the grid network is defeated by the following facts: (1) The heterogeneous nature of the underlying resources remains a noteworthy barrier to integrate resources together for a specific business need; (2) The interoperability between resources is largely prohibited due to the limitations from individual platforms and languages; (3) There lacks open software standards to enable organizations to develop collaborative business projects without understanding each individual system; and (4) The reusability of resources is questionable.

In recent years, with the rapid emerging of Web services concept, there appears a new branch in Grid computing towards a Web services-oriented grid, or so-called Service Grid. In the next section, we will first introduce the basic concept of Web services as well as its core techniques and standards.

3 Web Services orientation

3.1 Concept of Web Services

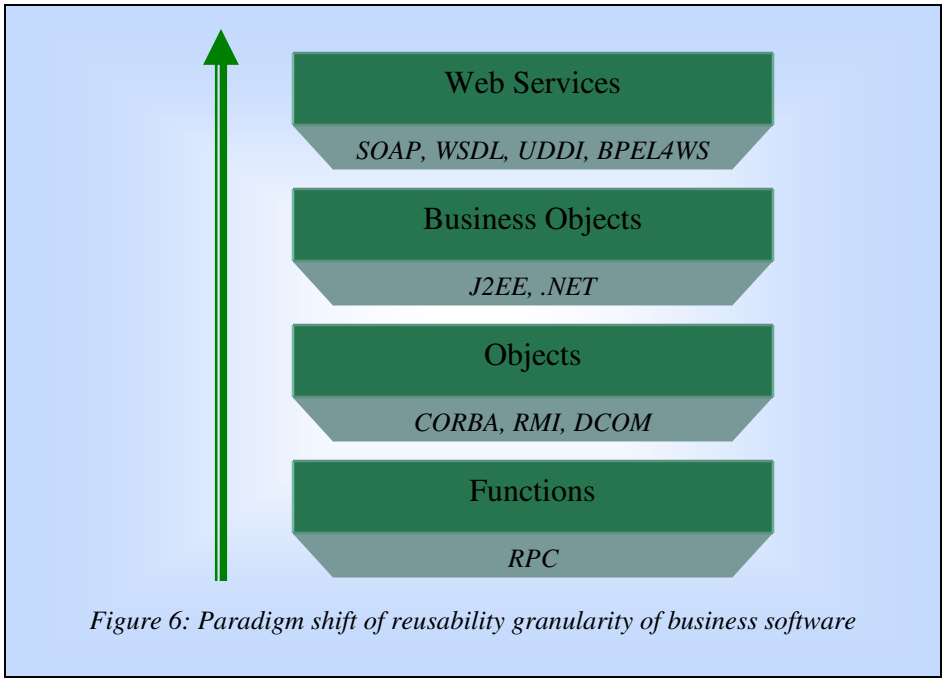

Figure 6: Paradigm shift of reusability granularity of business software

Reusability is one of the central elements that decide the success of business software development. Without reusing previous work, developers must design and code every application from scratch, which fact wastes valuable human resources and may result in schedule delays. The higher reusability, the more efficient software development. With the advancement of network and Web technologies, the reusability can be further facilitated by adopting distributed components. The last several decades have witnessed the paradigm shift of the granularity of software reusability, as shown in Figure 6. The lowest level of software reusability is limited to the function level, as Remote Process Call (RPC) provides facility for a software to invoke remote functions residing at another machine. The second level of software reusability is at the level of object that encapsulates data and associated operations (i.e., functions). Object Management Group (OMG)'s Common Object Request Broker Architecture (CORBA) [16], Sun's Java Remote Method Invocation (RMI) [17], and Microsoft's Distributed Common Object Model (DCOM) [18] are three major platforms that provide a collection of system-level services to support object-level reusability, such as

transaction, persistence, naming, security, etc. The third level is at business object level, or so-called business component level. Business components encapsulating complex business logic can be invoked in an efficient platform that hosts Web applications. Most outstanding examples, Java Platform 2 Enterprise Edition (J2EE) [19] and Microsoft .NET [20], provide a programming model based upon Web and business components. The highest level, which started just a few years ago, is to reuse complete existing business services residing anywhere in the Web to conduct new business. These business services are called Web services. The emerging paradigm of Web services has been promoting software reusability to an unprecedented level.

Simply put, a Web service is a programmable Web application that is universally accessible through standard Internet protocols [21]. In more detail, the Stencil Group, a leading industry analyst firm, defines Web services as [22]:

"…loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols."

The paradigm of Web services has been changing the Internet from a repository of data into a repository of services along the following three dimensions: (1) By means of each organization exposing its business applications as services on the Internet and making them accessible via standard programmatic interfaces, this model of Web services offers a promising way to facilitate Business-to-Business (B2B) collaboration. Second, Web services technology provides a uniform and loosely coupled integration framework to increase cross-language and cross-platform interoperability for distributed computing and resource sharing over the Internet. Third, this paradigm of Web services opens a new cost-effective way of engineering software to quickly develop and deploy Web applications by dynamically integrating other independently published Web services components to conduct new business transactions. With these promising features, the paradigm of Web services is considered to be the model of Internet computing for the future; and the Web service market is expected to grow up to $28 billion in sales in these several years [23].

3.1.1 Operational model of Web Services

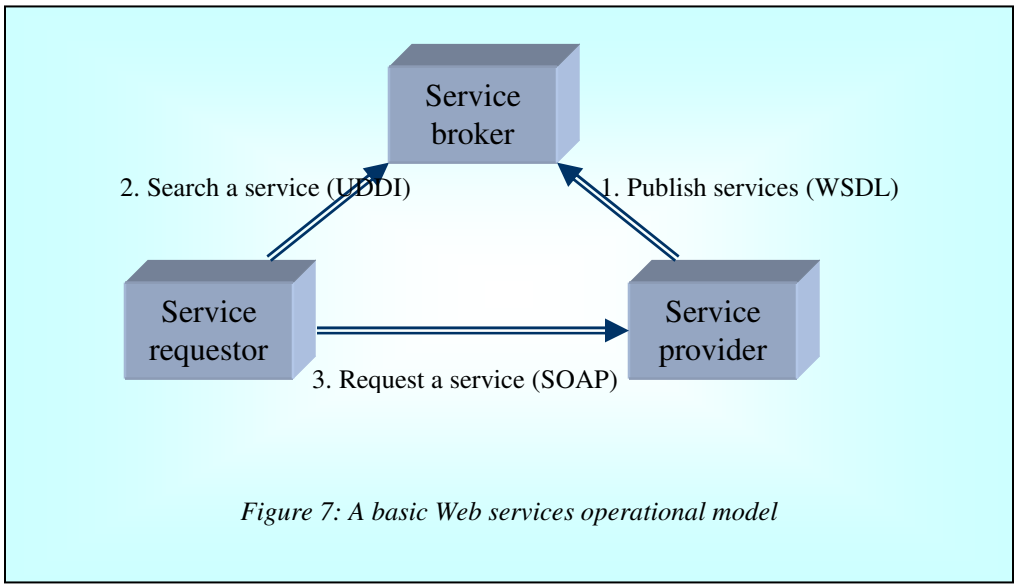

Figure 7: A basic Web services operational model

Regarding to any Web service, the entity that offers and hosts the service is called a service provider; and the entity that demands the service is called a service requester. Since the paradigm of Web services envisions that numerous business services will be published on the Internet as time goes on, instead of using an end-to-end client/server (or subscriber/publisher) operational model, a third role is identified as a dedicated service broker that acts as yellow pages to phone callers/receivers. Thus Web services basically adopt a triangular provider/broker/requestor operational model [24], as shown in Figure 7.

- Service providers publish Web services at service brokers. Service providers develop, deploy, and host Web services on their own sites. Then the service providers register with public service brokers (or so-called service registries) the Web services of their metadata description, e.g., their service types, capabilities, and network addresses. The services brokers subsequently publish registered Web services to the world.
- Service requestors search for Web services from the service brokers. Service requestors describe to public service brokers the demanded kinds of services; then the service brokers search the registered service metadata and deliver back the results that match the requests.
- Service requestors invoke Web services from service providers. Using the network addresses retrieved from

the service brokers, the service requestors bind to the located service providers. After negotiating with the service providers as appropriate, the service requestors access the required Web services and execute the services from the sites of the service providers.

The essential aspect of this broker-centered model is the concept of dynamic discovery and invocation. Web services are hosted by their own service providers. Service requestors dynamically search and locate services from service brokers, and invoke the Web services from the service providers over the Internet upon an on-demand basis.

3.2 Core Web Services technologies and standards

The underpinning that enables Web services is eXtensible Markup Language (XML) [25]. Endorsed by Worldwide Web Consortium (W3C) in February 1998, XML is a meta-language of markup language that makes data portable by defining a universally standard format for structured documents and data definitions on the Web. Detailed information about XML can be found at the official W3C Web site at http://www.w3c.org/XML.

Recall that the essential goal of Web services is to enable interoperability of distributed business services within any platform and language. Just as in the real world a common language is indispensable for effective and efficient communication among a group of people coming from different countries and cultures, XML becomes the "common language" for various business services to exchange information and interact with each other without human interventions. Based upon pure text, XML provides a system-independent way of describing data. Like static HTML (Hptertext Markup Language), XML encloses data in tags. Unlike HTML that only accepts its predefined set of tags, XML allows users to define application-specific tags that describe the content of any specific type of document. In addition, unlike HTML tags that describe how to display the enclosed content; XML tags represent semantic meanings of enclosed content. Moreover, XML adopts schema languages (e.g., XML schema and DTD (Document Type Definition)) to represent the rules that compilers can use to interpret the structure of a specific type of XML document. Therefore, XML provides a way of representing portable data. In other words, together with the associated

schemas, XML data can be exchanged by different applications.

Finally, it should be noted that XML is the basis for all other Web services standards.

3.2.1 Standards oriented to standalone Web services

The paradigm of Web services originally embraces three core categories of supporting techniques: (1) communication protocols, (2) service descriptions, and (3) service discovery. Each category possesses its own *ad hoc* standard: the Simple Object Access Protocol (SOAP) acts as a simple and lightweight protocol for exchanging structured and typed information among Web services; the Web Service Description Language (WSDL) is a description language that is used to describe the programmatic interfaces of Web services; and the Universal Description, Discovery, and Integration (UDDI) standard provides a mechanism to publish, register, and locate Web services in a systematic way. All of the three standards are derivative standards spawned from XML. These three categories constitute the backbone of Web services that enables the provider/broker/requestor operational model as shown in Figure 7: the specifications allow applications to find each other and interact following a loosely coupled and platform-independent model.

Simple Object Access protocol (SOAP)

Simple Object Access Protocol (SOAP) is a simple and lightweight protocol for exchanging structured and typed information among Web services. In the core of the Web services model, SOAP acts as the messaging protocol for transport with binding to existing Internet protocols such as Hypertext Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP) as the underlying communication protocol. Defining a uniform way of passing XML-encoded data, SOAP also enables a way of conducting Remote Procedure Calls (RPCs) binding with HTTP.

A SOAP message is specified as an XML information set. In detail, SOAP consists of three major parts for

defining a message framework and providing a message construct that can be exchanged between two Web services:

- An ***envelope*** that defines a framework for describing the content of the message and how to process it.
- A set of ***encoding rules*** that define a serialization mechanism for exchanging application-defined data types.
- A ***remote procedure calls (RPCs) convention*** that enables basic request/response interactions.

SOAP is endorsed by W3C and key industry vendors such as Sun Microsystems, IBM, Microsoft, etc. Detailed information about SOAP can be found at the W3C Web site at http://www.w3c.org/TR/SOAP.

Web Services Description Language (WSDL)

Web Services Description Language (WSDL) is an XML format for describing network services and their access information. W3C specifies WSDL as:

"…an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services)."

In the core of Web services model, WSDL is used to describe the metadata information of a Web service as what functionalities it has, where it is located, and how to invoke it. In more detail, a service provider uses WSDL specifications to publish Web services; a service broker uses WSDL specifications to find published services; and a service requestor uses WSDL specifications to bind to the services dynamically.

Using WSDL, a Web service is defined as a set of ports, each publishing a collection of port types that bind to network addresses using common binding mechanism. Every port type is a published operation that is accessible through messages. Messages are in turn categorized into input messages containing incoming data arguments and output messages containing results. Each message consists of data elements; and every data element must belong to a data type, either a simple type (e.g., XML Schema Definition (XSD)) or a complex type. Detailed information about WSDL can be found at the official W3C Web site at http://www.w3c.org/TR/WSDL.

Universal Description, Discovery, and Integration (UDDI)

Universal Description, Discovery, and Integration (UDDI) provides a "meta service" for locating Web services by enabling robust queries against rich metadata. In the core Web services model, UDDI defines a standard mechanism for service brokers to store descriptions of registered Web services in term of XML messages. In addition, by querying the UDDI registries, service requestors locate Web services so that they can invoke the services from the corresponding service providers.

UDDI registries can be either private services or public registries. The former type serves services within an enterprise or a community; while the latter type publishes services to the global business community on the Internet. The UDDI specifications record several types of information about a Web service that help service requestors determine the answers to the questions as "who, what, where and how": (1) Who: simple information about a business, such as its name, business identifiers, and contact information; (2) What: classification information that includes industry codes and product classifications, as well as descriptive information about the registered Web services; (3) Where: registration information (e.g., the URL (Uniform Resource Locator) or email address) through which each type of service can be accessed; and (4) How: registration references (i.e., tModels in the UDDI Documentation) about interfaces and other properties of a given service.

The UDDI working group includes leading technology vendors such as Sun Microsystems, IBM, HP, Microsoft, etc. Detailed information about UDDI can be found at its official Web site at http://www.uddi.org/.

3.2.2 Standards for business process using Web Services

However, depicting a narrow definition of Web services that refers to an implicit definition of SOAP+WSDL+UDDI, these three categories only guide service requestors to locate one Web service. Business interaction models typically require synchronous/asynchronous sequences of peer-to-peer message exchanges within stateful interactions involving multiple parties; thus, multiple Web services may be required to collaborate with each other for the common business interactions. For example, a travel planning process may include collaborative Web services such as flight scheduling, hotel reservation, and can rentals. The Business Process Execution Language for Web Services (BPEL4WS) is thus proposed for formal specification of synergistically coordinating and organizing Web services into business processes. By extending the Web services interaction model and enabling it to support business transactions, BPEL4WS defines an interoperable integration model that facilitates the expansion of automated process integration in both intra-corporate and inter-corporate environments.

BPEL4WS focuses on describing the behaviors of a business process based on the interactions between the process and its partners. The interactions occur through Web service interfaces, and the structure of the relationship at the interface level is encapsulated in a *partner link*. A BPEL4WS process defines how multiple business interactions are coordinated to achieve a common business goal, as well as the state and the logic necessary for this coordination. A rich process description notation is defined in BPEL4WS to precisely define essential service behaviors for cross-enterprise business protocols, such as (1) data-dependent behaviors (e.g., the delivery deadline), (2) exceptional conditions and their consequences, including recovery sequences, and (3) long-running interactions at various levels of granularity.

BPEL4WS separates the public behaviors of a business process from their internal implementations. A business process can be modeled in two ways, either an executable model or an abstract model. An executable process models actual behaviors of the participants in a business interaction; and an abstract model specifies the mutually visible message exchange behaviors of parties involved without revealing their internal behaviors.

BPEL4WS is the achievement of joint efforts from a group of technology vendors leading by IBM and Microsoft. Detailed information can be found at the site of BPEL4WS: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

3.2.3 Standards oriented to quality of Web services

Even considering an isolated Web service, the SOAP+WSDL+UDDI definition only focuses on the functional perspective of Web services. However, the business requirements of a Web service typically include non-functional features, such as reliability, security, safety, testability, fault tolerance, etc [26]. The Web services community has been starting to put significant efforts on addressing these non-functional perspectives of Web services.

To date most of the efforts emphasize on the security issue related to Web services. Two software industry giants, IBM and Microsoft, combine their efforts and propose the WS-Security standard [27], which is a family of protocols that enhances the messaging technique to solve three basic problems about the quality of protection of Web services: authentication and authorization of users, message integrity, and message encryption. Focusing on secure communication, these mechanisms can be used to accommodate a wide range of security models and encryption technologies.

Based upon the WS-Security, six enhanced models, as shown in Figure 8, are proposed to help establish secure interoperable Web services [27]: (1) WS-Policy provides a syntax-wired model to specify Web

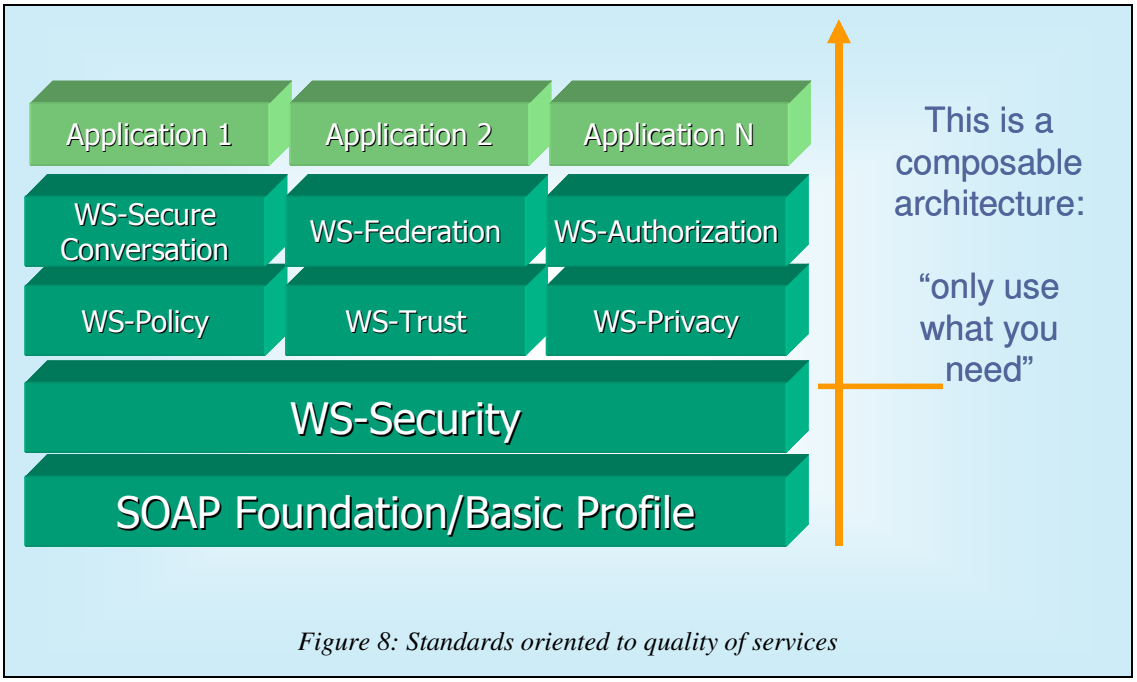

Figure 8: Standards oriented to quality of services

services endpoint policies; (2) WS-Trust defines methods to request and issue security tokens for establishing trust relationships; (3) WS-Privacy specification describes a model for expressing privacy claims inside of WS-Policy descriptions and associating privacy claims with messages; (4) WS-Authorization defines how Web services manage authorization data and policies; (5) WS-SecureConversation defines a security context based upon security tokens for secure communication; and (6) WS-Federation defines mechanisms to enable identity, account, attribute, authentication, and authorization federation across different trust realms.

4 Services oriented Grid

To sum up, Web services define a standard means for business applications to publish, discover, and interoperate with each other. Recall that a grid is defined as a layer of networked services that grant users single sign-on access to a distributed collection of compute, data, and application resources. The convergence of Web services of Grid computing opens an unprecedented and powerful technique to support business computing.

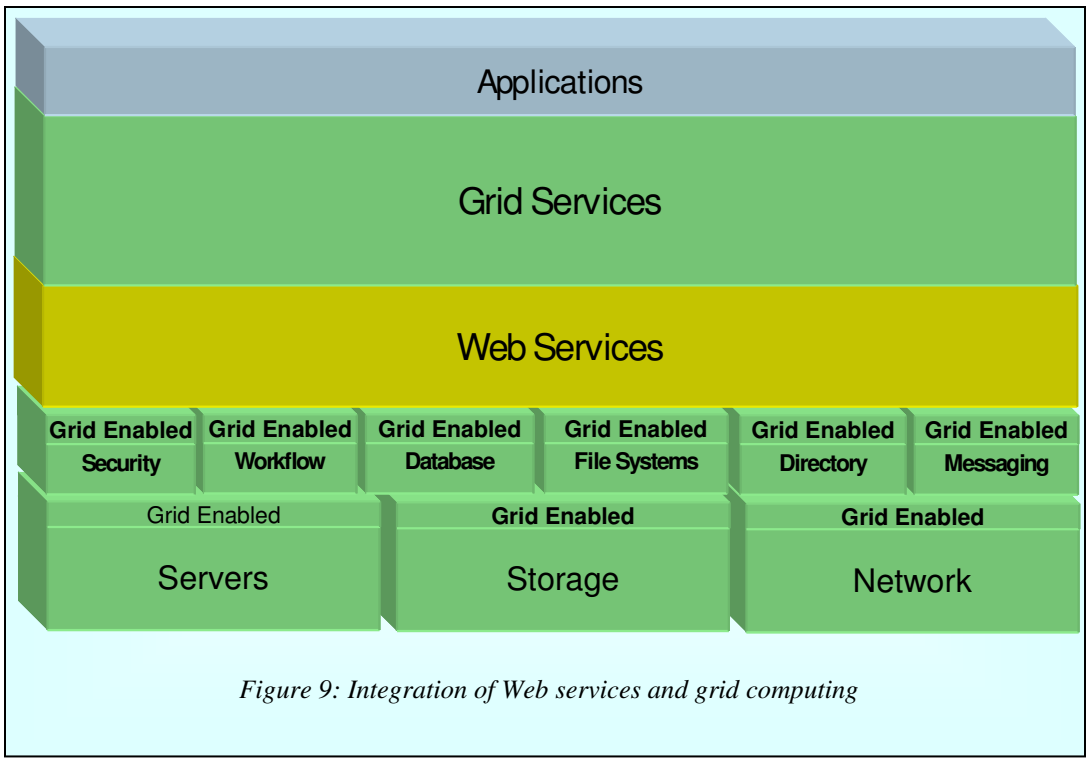

Figure 9: Integration of Web services and grid computing

The marriage of Web services and Grid computing is called services-oriented grid, or service grid in short. Service grid assembles a diverse set of specialized enabling services required to support the connections across applications enabled by Web services technology.

As shown in Figure 9, a vision presented by IBM in Globus World 2004 (http://www.globusworld.org/program/slides/k1a.pps), the essential goal of business computing is to share, access, and manage diverse resources, widely ranging from raw data to business applications or services. The concept of Grid computing provides a systematic approach to organize resources in the grid network, thus enables resources on demand, global accessibility, and vast resource scalability:

- Resource on demand. Grid protocols facilitate resource locating based upon on-demand basis in the grid network.
- Global accessibility. The entire collection of resources is viewed as a seamless information processing system that can be accessed from any location.

- Vast resource scalability. Organized in a grid network, resources can be easily grouped together to achieve higher scalability.

Meanwhile, the concept of Web services facilitates applications on demand, secure and universal access, and business integration:

- Applications on demand. Using the model of Web services, each business service is published on the Web, and users can dynamically locate and invocate the service at any time from anywhere.
- Secure and universal access. With the Web services standards, heterogeneous business applications can communicate with each other without human interventions. The publishing and subscribing of business services are both using uniform and universal standards. On-going Web services security protection protocols typical as WS-Security solidify and secure the business communications over the Internet.
- Business integration. The model of Web services depicts a promising vision for business computing by dynamically composing smaller-scale business services into new and larger-scale business applications to address specific business needs at run time.

Therefore, as shown in Figure 9, Web services and Grid computing benefit from each other: Grid computing provides scalability for Web services implementations; and Web services provide a standards-based, loosely coupled, and lower-cost foundation for Grid computing implementations. In summary, service grid - the marriage of Web services and Grid computing - establishes an ideal platform to support sharing, access, and managing diverse IT resources for business usages.

4.1 OGSA and WSRF

(http://www.globus.org/ogsa, http://www.globus.org/wsrf)

OGSA (Open Grid Service Architecture) is based on a service-oriented architecture. In this architecture, a service is an entity that provides some capability to its clients by exchanging messages. A service is defined by identifying sequences of specific message exchanges that cause the service to perform some operation. By encapsulating service operations behind a common message-oriented service interface, OGSA encourages service virtualization and isolates users from the details of service implantation and location.

The three principal elements of OGSA are: (1) Open Grid Services Infrastructure (OGSI), (2) OGSA services, and (3) OGSA schemas. OGSA builds on Web services that provide the basic mechanisms used to describe and invoke grid services. In OGSA environment, each entity is encapsulated into a service. A grid service refers to a Web service that provides a set of well-defined interfaces and conventions defined by OGSA. In other words, a grid service is a WSDL-defined service that conforms to a set of conventions relating to its interface definitions and behaviors. The concept of grid service facilitates the integration of and access to heterogeneous grid resources.

A grid service description consists of extended WSDL specifications that define interfaces and associated semantics of grid services. A grid service instance is an addressable, potentially stateful, and potentially transient instantiation of such a description. The Factory interfaces' *createService* operation creates a requested grid service with a specified interface and returns the Grid Service Handle (GSH) and initial Grid Service Reference (GSR) for the new service instance. The operation also registers the new service instance with a handle resolution service.

OGSA services may be hosted in a variety of different environments and may communicate via different protocols. OGSI defines the mechanisms for creating, naming, managing lifetime, monitoring, grouping, and exchanging information among entities (i.e., grid services). These conventions provide support for the controlled, fault-resilient, and secure management of the distributed and often long-lived state commonly required in distributed applications. In addition, OGSI also introduces standard factory and group registration interfaces for creating and discovering Grid services.

WSRF (WS-Resource Framework), announced by the Globus Alliance and IBM in conjunction with HP on 20 January 2004, defines a generic and open framework for modeling and accessing stateful resources using Web Services. This framework includes mechanisms to describe different views on a state, to support the management of the state through properties associated with Web Services, and to describe how these mechanisms are extensible to groups of Web Services.

Web service implementations typically do not maintain state information during their interactions; however, their interfaces must frequently allow for the manipulation of state as a result of Web service interactions. Web services, therefore, are required to provide their users with the ability to access and manipulate states. WSRF defines conventions for managing state so that applications discover, inspect, and interact with stateful resources in standard and interoperable ways. Particularly, WSRF defines these conventions within the context of established Web services standards.

4.2 Service grid infrastructure implementation

(*http://www.globus.org/toolkit*)

The Globus Toolkit is an open source software toolkit being developed by the Globus Alliance and many others all over the world. Version 1.0 of the toolkit came out in 1998, while version 2.0 was released in 2002. When the latest 3.0 version was released, it signaled a revolutionary paradigm shift for grid computing, the adoption of Web services by the grid computing community as the 3.0 version was based on the new open-standard Grid services (OGSA).

Currently Globus Toolkit version 4.0 (GT4) is making its way toward the direction of Web services-oriented grid. In its blueprint, it contains core components including security management, data management, execution management, information services, and common runtime components. One of the core components for execution management is Web Services Grid Resource Allocation and Management

(WS-GRAM). It comprises a set of WSRF-compliant Web services to locate, submit, monitor, and cancel jobs on Grid computing resources. WS-GRAM has a set of services and clients for communicating with a range of different local job schedulers using a common protocol. WS-GRAM is meant to address a range of jobs where reliable operation, stateful monitoring, credential management, and file staging are important.

The data management in GT4 is handled through OGSA Data Access and Integration (OGSA-DAI: *http://www.ogsadai.org.uk*), a technology aiming to manage data in the grid area. Several classes of components are dealt with in OGSA-DAI: Grid Data Services (GDS), Grid Data Service Factories (GDSF), and a Grid Data Service Registry (GDSR). Data is moved between these components by a collection of mechanisms, from the simplest use of the OGSA infrastructure to transferring XML documents to sophisticated mechanisms such as Grid-FTP and MQ Series.

4.3 Challenges to service grid

Current service grid standard and implementation development still leave behind the promises to support universal resource sharing and reuse. While currently the service grid employs WS-GRAM and OGSA-DAI to manage job schedulers for computing and data sharing, respectively, there are many problems to be solved. It does not have a solution yet for managing application sharing, particularly for the legacy business applications sharing under certain operation systems. Current Web services-based integration technologies do not solve the application sharing problem simply because the application sharing involves more than application integration. In the real business environment, it is often necessary to have a solution that offers on-demand computing power without losing investment over the legacy business application installed. This motivates us to move to an integrated service platform that could manage data, computing, and application in an integrated fashion based on service grid technology, as an application platform demonstration for service grid.

5 Integrated service platform for integration

Integrated services platform (ISP), being developed at E-business Technology Institute, The University of Hong Kong (ETI), provides functions to manage a network of resources as a grid application support platform for resource sharing. The core component of ISP is a set of brokers that aggregate and connect distributed IT resources. These brokers manage a heterogeneous group of resource suppliers, provide resource brokering services, and keep control of resource activities for multiple groups of consumers. For example, three types of brokers are identified based upon the individual roles they act: (1) computation broker, (2) data broker, and (3) application broker. Each broker type is defined by its specific standard interface. A computation broker acts as an entry point for job submissions; a data broker handles data and storage sharing, and an application broker deals with software sharing.

5.1 Platform architecture

Distributed resources in ISP – servers, storage, databases, and practically all other applications and peripherals – are connected through a grid network and aggregated by a Service Broker. As shown in Figure 10, the Service Broker in ISP adopts a hierarchical structure to provide Internet service requestors a wide range of resource brokering services. With a common standard interface (service portal) as an entry point, service requestors can obtain services such as job submission (computation brokering), data and storage sharing (data brokering), and software license sharing (application brokering). The nature of the resources is then made transparent to service requestors.

In detail, ISP is constituted as a three-layer structure, as shown in Figure 10. The first layer is the Grid Portal layer that interacts with Internet service requestors. The middle layer is the Service Grid layer that manages generic resource brokering facilities. As support to the Service Grid layer, a number of specialized brokers (application, data, computation, and license) are developed. The third layer is composed of components that handle different resource brokering services (WS-GRAM, Grid SuperScalar, GridFTP,

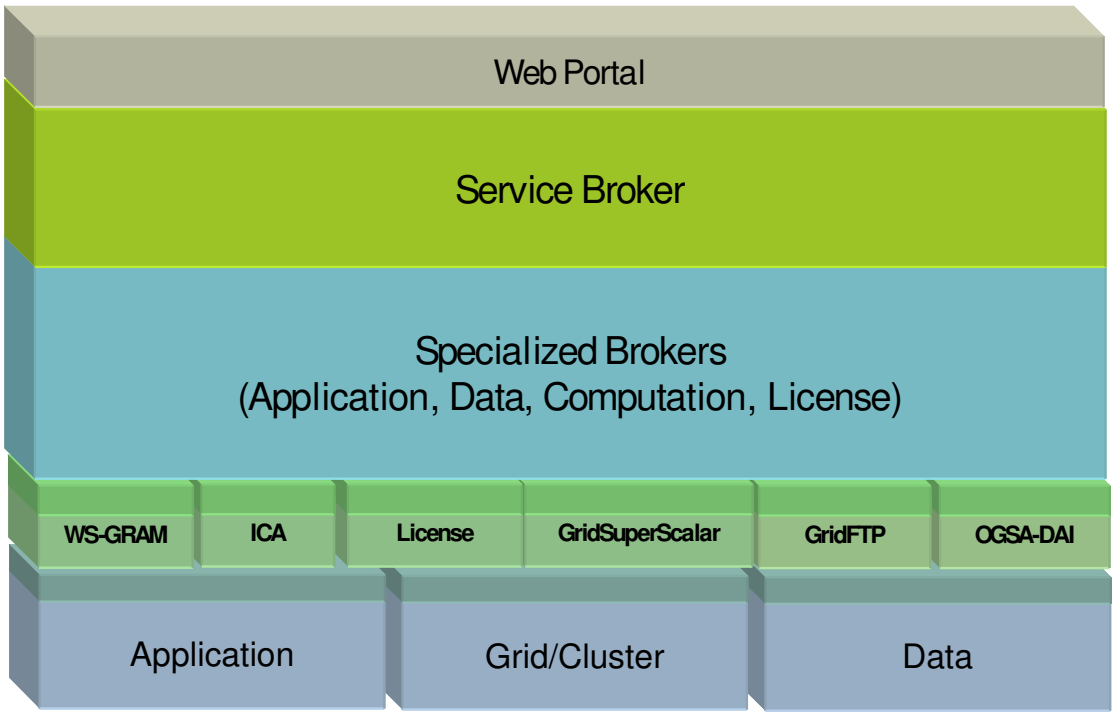

Figure 10: Infrastructure of the integrated service platform

OGSA-DAI, and license management tools) based upon the specific nature of different types of resources: data, application, computation and license.

5.1.1 Service Broker

The Service Broker handles the process of making decisions involving resources over multiple administrative domains. Depending upon different nature of the resources (i.e., data, application, and computation), the Service Broker in turn invokes one of the specific brokers to manage resources in the grid such as data, application, and computation power. The Service Broker contains a License Broker, a Publication Broker, a Discovery Broker, and a Notification Broker:

- The License Broker manages licenses based on license usage analysis and prediction, as well as considerations on scheduling policies. It works with application license management software to provide organizations transparently shared software application licenses across multiple

geographic locations and organizational boundaries. It also ensures license availability for sharing to enable organizations to maximize and optimize the utilization of the software licenses.

- The Publication Broker provides capabilities for invoking service registration services to describe services and register services in the Service Broker. Necessary security is enforced in order to publish and modify the description of services.
- The Discovery Broker provides standard interfaces to locate service discovery facility and help find the services needed.
- The Notification Broker allows subscription for the notifications of resources. If service requestors subscribe to a resource and specify the notification contract including an initial lifetime for the subscription, a stream of notification messages will flow from the resource to all the subscribers when corresponding events occur at the resource. The message may trigger an action on the service requestors if an action receiver is specified in the notification contract.

5.1.2 Computation and Data Broker

The Computation Broker enables users to quickly access distributed remote computation resources in a secure, easy, and transparent way. For example, it interfaces with WS-GRAM and then invokes job schedulers to schedule job computations. The process may involve searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites. Three modules are included in the computation broker: a scheduling broker, a transfer broker, and a job broker. The scheduling broker accepts requests from end users, then selects and routes the requests to a job scheduler. The job scheduler locates a job broker. The job broker then submits jobs through a job manager to an execution site, starts and ends job execution, monitors the running status, and recovers from remote failures output and error. The transfer broker exchanges data, such as the job's executable and standard input files, between a remote HTTP, FTP, or Globus Access to Secondary Storage (GASS) server, through the functions of the data services located by the data broker.

5.1.3 Application Broker

The Application Broker is responsible for accessing distributed remote application resources on behalf of grid users. It mainly employs a user interface (UI) sharing, application distribution, and license brokering technologies.

- UI sharing: it uses thin client (such as Citrix ICA (Independent Computing Architecture) and terminal service) and portlet technology (allowing a portal page to be customized more quickly either internally by its development team or by end users) to enable remote sharing of application UI. The thin client portlet identifies the platform where a user resides on, and offers an opportunity to automatically download the appropriate thin client. This option saves administrators and Application Service Providers (ASPs) valuable time because they on longer have to touch every desktop to deploy thin client software.
- Application distribution: it provides standard interfaces to invoke an application distribution implementation to resolve and submit an application request, schedule an application process, and monitor resource usage among the computation nodes. One of such implementations is Citrix Metaframe server farm. It can automatically assign and reassign processes among the computation nodes in order to continuously take advantage of the best available resources. It also allows the distribution of application tasks to provide failure resilience.

6 Challenges and considerations

Although the concept of service grid paints futuristic visions of business computing equipped with universal resource sharing and scalable integration, this technology is still at its infancy and far from the level of maturity when most business organizations will adopt the technology to serve core activities. As we discussed earlier, the foundation of services-oriented grid is its new set of standards and protocols in defining this new generation of technology; however, these new standards and protocols are either still under debate or insufficient so that new complementary standards and protocols are keeping on emerging [28].

The paradigm of Web services is considered to be the model of Internet computing for the future; and the Web service market is expected to grow up to $28 billion in sales in these several years [23]. However, the actual adoption of Web services in industry is quite slow. It is not clear that this new model of Web services provides any measurable increase in software quality; therefore, many companies are reluctant to employ Web services to conduct business. The Web services community has been putting significant efforts on offering some promise to address the security challenges related to Web services. Nevertheless, WS-Security and related techniques and languages only address the security issue of Web services-centered computing, while software quality is a holistic property that encompasses many more attributes beyond security, such as reliability, safety, survivability, interoperability, availability, fault tolerance, performance, etc [26].

In spite of the immaturity of Web services standards on software quality, the core standards need to be enhanced. As the standard transport protocol between Web services, SOAP suffers from its significant consumption of network bandwidth [29]. This drawback derives from the fact that SOAP is built on the basis of XML. As any XML message, most parts of a SOAP message is XML markup and protocol overhead; while only a small part represents the real payload. Although the rapid advancement of present network techniques has been dramatically increasing the bandwidth to support even highly demanding fields such as multimedia streaming, there still exist domains where network bandwidth is costly (e.g., wireless network and dial-up connections).

WSDL is the *ad hoc* standard for Web services publication. However, WSDL can only specify limited static information of a Web service, such as its abstract interface, bindings to specific message formats and protocols, and the location the service. In recognition of this problem, researchers from both academia and industry have been developing other description languages to extend the power of WSDL to depict Web service architecture.

Furthermore, as a *de facto* standard to describe the interactions between Web services, BPEL4WS also has

a limitation. The current BPEL4WS specifications do not allow definitions of operations in addition to Web services components. In other words, for a system to be defined by BPEL4WS, it can only include Web services components. If a system intends to define some local operations between two Web services components, e.g., change data format and verify data obtained, these operations cannot be defined by BPEL4WS. As a result, for some applications (e.g., a travel scheduling system that contains Web services components of air ticket scheduling, hotel reservation, and car rental) where the output of one Web service component can be directly used as an input to another Web service component, BPEL4WS is sufficient to define various interactions and relationships between the components in the system. For other systems where Web services components are just part of the systems or where Web services components cannot be simply integrated with each other without further data transformation, however, BPEL4WS is inadequate and cannot be used to define the system.

In summary, since Web services are very much "work in progress", the long-term wisdom of adopting Web services as the basis to construct universally interoperable grid is inevitably discounted by the short-term difficulties. Fortunately, since the paradigm of Web services has been widely recognized as the solution for the new generation of business computing, well-known international standards organizations such as W3C, OASIS, and WS-I, have been devoting tremendous efforts on enhancing standards; industrial giants such as IBM and Microsoft have set up their goals towards Web services; and numerous researchers and practitioners have been conducting hard work on this new technology. Therefore, it is reasonable for us to believe that in the very near future, Web services technology will be ready to support grid computing.

Finally, let us take a look at potential challenges to grid computing itself. Grid computing represents a concept that affects the entire technology industry. As the focus of the grid research and development is shifting from high performance computing to meeting commercial business computing requirements, we see a bright future for the grid. The grid vision of resource virtualization is being well accepted. However, it will still take years before commercial grid computing becomes mature enough to be the popular enabling means for business applications. Challenges lie on both business and technical sides with issues

such as grid standards maturation, software readiness, and market cultivation beyond traditional high performance computing.

In addition, bringing together the virtualized computing resources, grid computing is likely to move one step further to open the door to utility computing to provide electric utility like services. Grid resource virtualization couples naturally with utility computing, which provides IT functionality on demand. Certainly, the key to the success of utility computing is dependent on delivering the IT functionality at users' needs. As a result, quality of services becomes an increasingly important metric that matters.

7 Bibliography

[1] The Message Passing Interface (MPI) standard, http://www-unix.mcs.anl.gov/mpi.
[2] OpenMP webpage, http://www.openmp.org/drupal.
[3] PACX-MPI webpage, http://www.hlrs.de/organization/pds/projects/pacxmpi.
[4] N. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface", *Journal of Parallel and Distributed Computing (JPDC)*, May, 2003, 63(5): pp. 551-563.
[5] S. Venkatakrishnan, M. Kuchhal, and S. Kumar, "Grid Application Framework for Java", 2003, http://alphaworks.ibm.com/tech/GAF4J.
[6] M. Alt, J. Dünnweber, J. Müller, and S. Gorlatch, "HOCs: Higher-Order Components for Grids", *Proceedings of Workshop on Component Models and Systems for Grid Applications*, 2004, Springer Verlag.
[7] R.V. van Nieuwpoort, J. Maassen, G. Wrzesinska, T. Kielmann, and H.E. Bal, "Satin: Simple and Efficient Java-based Grid Programming", *Accepted for publication in Journal of Parallel and Distributed Computing Practices*, 2004.
[8] Griphyn project website, "Griphyn project website", http://www.griphyn.org.
[9] R.M. Badia, J. Labarta, R. Sirvent, J.M. Pérez, J.M. Cela, and R. Grima, "Programming Grid Applications with GRID Superscalar", *Journal of Grid Computing*, 2003, 1(2), pp. 151-17.
[10] The Ibis webpage, http://www.cs.vu.nl/ibis.
[11] R.V. van Nieuwpoort, J. Maassen, G. Wrzesinska, T. Kielmann, and H.E. Bal, "Adaptive Load Balancing for Divide-and-Conquer Grid Applications", *Accepted for publication in Journal of Supercomputing*, 2004.
[12] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation", *Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, 2002, Edinburgh, Scotland, pp. 37-46.
[13] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments", *Journal of Grid Computing*, 2003, 1(1): pp. 25-39.
[14] Condor Team, "The Directed Acyclic Graph Manager", 2002, http://www.cs.wisc.edu/condor/dagman.
[15] GRID superscalar website, http://www.cepba.upc.es/Grid.
[16] CORBA, http://www.corba.org.
[17] RMI, http://java.sun.com/products/jdk/rmi.
[18] DCOM, http://www.microsoft.com/com/default.mspx.
[19] J2EE, http://java.sun.com/j2ee.
[20].NET, http://www.microsoft.com/net.

[21] C. Ferris and J. Farrell, "What Are Web Services?" *Communications of the ACM*, Jun., 2003, 46(6): pp. 31.
[22] Stencil Group, 2003, http://www.stencilgroup.com/ideas_scope_200106wsdefined.html.
[23] P. Holland, "Building Web Services from Existing Application", *eAI Journal*, Sep., 2002, pp. 45-47.
[24] J. Roy and A. Ramanujan, "Understanding Web Services", *IEEE IT Professional*, Nov., 2001, pp. 69-73.
[25] XML, http://www.w3.org/XML.
[26] P. Neumann, "Principled Assuredly Trustworthy Composable Architectures", emerging draft of the final report for DARPA's Composable High-Assurance Trustworthy Systems (CHATS) program, 2004, http://www.csl.sri.com/users/neumann/chats4.pdf.
[27] WS-Security, http://www-106.ibm.com/developerworks/webservices/library/ws-secure.
[28] Z. Luo and J. Zhang, *Web Services Oriented Grid for Business Computing, Technical Report*. 2004, ETI, The University of Hong Kong, Pokfulam, Hong Kong.
[29] C. Werner, C. Buschmann, and S. Fischer, "WSDL-Driven SOAP Compression", *International Journal of Web Services Research*, 2005, 2(1): pp. 14-35.