Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.4, No.11, 2013
www.iiste.org

Service Oriented Grid Computing Model as a means of Cost Sharing in the Institutions of Higher Learning in Kenya

James Mwikya Reuben[1*] Alex Njoroge Kangethe[2]

1. Department of ICT, Dedan Kimathi University of Technology, PO box 657, Nyeri,Kenya
2. Department of ICT, Dedan Kimathi University of Technology, PO box 657, Nyeri,Kenya

* E-mail of the corresponding author: jamesrbn2@gmail.com

Abstract

The use of distributed systems by enterprises and academic institutions has increased exponentially in recent years, enabled by factors such as ready access to the Internet and the World-Wide Web, the maturity and ubiquity of the HTTP protocol, and the improvement in secure communication technology. In the early days, distributed applications communicated using proprietary protocols, and system administrators used adhoc (improvised) methods to manage systems that might be across town, on another continent, or anywhere in between. Numerous standards have been developed over the years to ease the costs of deployment and maintenance, with varying degrees of success. Today, the key technologies in distributed systems are service-oriented architecture (SOA), Web services, and grid computing, all of which are seeing significant investment in standardization and increasingly rapid adoption by organizations of all types and sizes. Academic organizations in Kenya have seen increase in the number of students admitted as well reduction in central government funding to these institutions to purchase more computer systems and procure management information systems .In this paper we offer a high-level description of each of the technologies, and how they can be used to develop a cost effective co-funded dynamic system that can be used by the institutions.

Keywords:service-oriented architecture, Web services, grid computing.

1. Service Oriented Architecture (SOA)

To understand the term *service-oriented architecture*, it is useful to review the key terms *(Bell et al, 2008).*

- An **Architecture** is a formal (official) description of a system, defining its purpose (use), functions (task), externally visible properties, and interfaces (boundary). It also includes the description of the system's internal components (modules) and their relationships, along with the principles (philosophy) governing its design, operation, and evolution (growth).
- A **service** is a software component (module) that can be accessed via a network to provide functionality (task, utility) to a service requester.
- The term **service-oriented architecture** refers to a style of building reliable distributed systems that deliver functionality as *services*, with the additional emphasis on loose coupling (union) between interacting services.

Technically, Service-Oriented Architecture (SOA) is a software design methodology based on structured collections of discrete software modules, known as services, that collectively provide the complete functionality of a large or complex software application as suggested by *Latha et al (2005*).Each service that makes up an SOA application is designed to provide a tightly defined set of functions. As a result, each service is built as a discrete piece of code. This makes it possible to reuse the code in different ways throughout the application by changing only the way an individual service interoperates with other services that make up the application, versus making code changes to the service itself. Therefore, we regard SOA as an *architectural style* that emphasizes implementation of components as modular *services* that can be discovered and used by clients *(Bieberstein et al., 2005).*

Services (software modules) normally have the following characteristics *(Channabasavaiah, et al., 2003):*

- Services may be individually useful, or they can be integrated—composed—to provide higher-level services. Among other benefits, this promotes re-use of existing functionality.
- Services communicate with their clients by exchanging messages: they are defined by the messages they can accept and the responses they can give.

- Services can participate in a workflow, where the order in which messages are sent and received affects the outcome of the operations performed by a service. This notion is defined as "service choreography."
- Services may be completely self-contained, or they may depend on the availability of other services, or on the existence of a resource such as a database. In the simplest case, a service might perform a calculation such as computing the cube root of a supplied number without needing to refer to any external resource, or it may have pre-loaded all the data that it needs for its lifetime. Conversely, a service that performs student tuition balances would need real-time access to balance information in order to yield correct values.
- Services advertise details such as their capabilities, interfaces, policies, and supported communications protocols. Implementation details such as programming language and hosting platform are of no concern to clients, and are not revealed.

Figure 1 illustrates a simple service interaction cycle. Service interaction begins with a service advertising itself through a well-known registry service (1). A potential client, which may or may not be another service, queries the registry (2) to search for a service that meets its needs. The registry returns a (possibly empty) list of suitable services, and the client selects one and passes a request message to it, using any mutually recognized protocol (3). In this example, the service responds (4) either with the result of the requested operation or with a fault message.

This illustration shows a simple synchronous, bi-directional message exchange pattern, a variety of patterns are possible—for example, an interaction may be one-way, or the response may come not from the service to which the client sent the request, but from some other service that completed the transaction.

1.1 Service Loose Coupling

This term implies that the interacting software components minimize their in-built knowledge of each other. They discover the information they need at the time they need it thus Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other *(Latha et al,.2005*). For example, having learned about a service's existence, a client can discover its capabilities, its policies, its location, its interfaces and its supported protocols. Once it has this knowledge, the client can access the service using any mutually acceptable protocol. The benefits of loose coupling include:

- Flexibility: A service can be located on any server, and relocated as necessary. As long as it maintains its registry entry, prospective clients will be able to find it.
- Scalability: Services can be added and removed as demand varies.
- Replaceability: Provided that the original interfaces are preserved, a new or updated implementation of a service can be introduced, and outdated implementations can be retired, without disruption to users.
- Fault tolerance: If a server, a software component, or a network segment fails, or the service becomes unavailable for any other reason, clients can query the registry for alternate services that offer the required functionality, and continue to operate without interruption.

1.2 Service Statelessness and State

Simply, the benefits of loose coupling, as listed above, are derived from the fact that a client can choose to go to any service that is capable of fulfilling its need. If its choice is restricted to a single service then a tight coupling exists between the client and the server, and the benefits of loose coupling are diminished *(Yvonne Balzer,2004).*

In the simple case of an employee pay slip from payroll service it is easy to see that once a client has requested and received information, the transaction is completed, and the client has no particular need to revisit the same service for its future needs. From this perspective, the client and service are loosely coupled.

For a more complex transaction that requires several steps, however, the design of the service might be such that the service retains in its local memory some information ("state") about the first step, expecting to make use of it when the client contacts it for the next step. In this case, the service is "stateful," and the client must return to the same service for the next step. This might result in a delay if many clients are using the same service or in a transaction failure if the node hosting the service fails between steps *(Fabio Baroncelli et al 2007).*

A better approach to the design of the service is for it not to retain state about the transaction, but to be "stateless." This implies that in a multi-step transaction *(Fabio Baroncelli et al 2007)*:

- At the end of each intermediate step the service must hand back to the client sufficient state information to enable *any* qualified service to identify and continue the transaction.
- The client must hand the state information to whichever service it selects to process the next step of the transaction.
- The selected service must be able to accept and handle the state information supplied by the client, regardless of whether it processed the earlier steps itself.

Figure 2 shows a multi-step client/service interaction. A client engaged in a three-step transaction with several services, each of which might be capable of handling any part or all of the transaction. The service that handles Step 1 stores the details of the in-progress transaction in the database, and returns requested information to the client, along with a transaction identifier. The client might request confirmation from the user before passing the transaction identifier to another service, which uses it to retrieve the state information from the database and initiates Step 2. This service then updates the database and returns additional information to the client. Finally, the client passes the transaction identifier back to a third service with a request to complete the transaction.

The approach outlined above enhances loose coupling by separating the transaction's state from the services that operate on it. In the example, both the account data and the details of the transaction can be considered to be state information, but the account data is permanent, while the transaction details only need to exist while the transaction is in progress. To minimize the amount of state that needs to be passed between the clients and the services, the critical account data and the details of the transaction are held in the database, the common requirement for all participating services is that they must be able to access the database, given a simple token such as a customer's account number, which can easily be passed between the client and the services *(Tony Shan, 2004).*

2. Web Services

Web services are distributed software components that provide information to *applications* rather than to humans, through an application-oriented interface *(M. Hadi Valipour et al 2004).* Web services can implement a service-oriented architecture. Web services make functional module *blocks* accessible over standard internet protocols independent of platforms and programming languages. These services can represent either new applications or just wrappers around existing legacy systems to make networked-enabled. The information is structured using *eXtensible Markup Language* (XML), so that it can be parsed and processed easily rather than being formatted for display. In a Web-based retail operation, for example, Web services that may be running on widely separated servers might provide students/employee registration, account management, inventory control and human resource management, all of which may be invoked multiple times in the course of a single transaction *(WebServices.org, 2013*).

Web services publish details of their functions and interfaces, but they keep their implementation details private; thus a client and a service that support common communication protocols can interact regardless of the platforms on which they run, or the programming languages in which they are written. This makes Web services particularly applicable to a distributed heterogeneous environment.

The key specifications used by Web Services Implementers are *(OGSA standards, 2012)*:

- XML (eXtensible Markup Language)—a markup language for formatting and exchanging structured data.
- SOAP (Simple Object Access Protocol)—these are web services standards that provide greater interoperability and some protection from lock-in proprietary vendor software. SOAP is an XML-based protocol that has gained broad industry acceptance after recommendation by World Wide Web Consortium (W3C). The protocol specifies envelope information, contents and processing information for a message. One can however implement SOA using any service-based technology, such as CORBA, Jini or REST.
- WSDL (Web Services Description Language)—an XML-based language used to describe the attributes, interfaces and other properties of a Web service. A WSDL document can be read by a potential client to learn about the service.

2.1 Web Services and Virtualization

Towards Service-orientation is an architectural style, while Web services are an implementation technology. The two can be used together, and they frequently are, but they are not mutually dependent. The ease with which Web services can be implemented and the ability to access them from any platform, local or remote, has led to their rapid adoption by system management bodies as *virtualization agents* that provide common manageability interfaces to disparate resources. For example, a Web service might be designed to "represent" a particular device or a legacy application, accepting control and monitoring requests through standardized interfaces, communicating with the resource through its native interface, and returning the result to the requester in a standard format *(Chatterjee et al 2009).*

Figure 3 illustrates a standards-aware management console (manager). Management console can manage set of dissimilar resources through a virtualization layer of Web services, each of which exposes standardized interfaces for the manager to use, while communicating with its associated resource through the resource's proprietary interfaces *(Distributed Management Task Force, 2012).*

3. Grid Computing

Grid computing is a distributed computing environment in which the use of disparate resources such as compute nodes, storage, data and applications, often spread across different physical locations and administrative domains, is optimized through virtualization and collective management *(Open Grid Services Architectire, 2012).*

Grid computing requires the use of software that can divide and farm out pieces of a program to as many as several thousand computers. Grid computing can be thought of as distributed and large-scale cluster computing and as a form of network-distributed parallel processing. It can be confined to the network of computer workstations within a corporation or it can be a public collaboration (in which case it is also sometimes known as a form of peer-to-peer computing) (*Latha Srinivasan et al, 2005).*

Figure 4 illustrates a grid, it shows four connected academic organizations have contributed computational and storage resources. Each organization must employ security procedures to prevent unauthorized access to its private resources by grid users.

Grid computing appears to be a promising trend for three reasons *(Global Grid Forum, 2011):*

(i) Its ability to make more cost-effective use of a given amount of computer resources, minimize cost and increased agility and collaboration.

(ii) As a way to solve problems that can't be approached without an enormous amount of computing power, and

(iii) Because it suggests that the resources of many computers can be cooperatively and perhaps synergistically harnessed and managed as a collaboration toward a common objective.

Common characteristics of grids are that they tend to be large-scale and widely distributed, to require decentralized management, to comprise numerous heterogeneous resources, and to have a transient user population. Hence grids exemplify the need for a highly scalable, reliable, platform-independent architecture that supports secure operation and standardized interfaces for common functions.

A number of corporations, professional groups, university consortiums, and other groups have developed or are developing frameworks and software for managing grid computing projects. The European Community (EU) is sponsoring a project for a grid for high-energy physics, earth observation, and biology applications. In the United States, the National Technology Grid is prototyping a computational grid for infrastructure and an access grid for people. Sun Microsystems offers Grid Engine software. Described as a distributed resource management (DRM) tool, Grid Engine allows engineers at companies like Sony and Synopsys to pool the computer cycles on up to 80 workstations at a time *(M. Hadi et al, 2009).*

4. Service-Oriented Grids

Grids are facilitated by the use of grid middleware: software components and protocols that provide the required controlled access to resources. Efforts toward the standardization of Grid Computing are based on the Open Grid Service Architecture (OGSA) , that specifies the core services and the high-level functionalities of grid

computing implementations, and the Web Service Resource Framework (WSRF), that specifies a web service infrastructure (i.e., a set of six basic web services) retaining all the Open Grid Service Infrastructure (OGSI) concepts *(Global Grid Forum, 2011)*. To date, grids have been built using, for the most part, either ad hoc public components or proprietary technologies. The rapid advances in Web services technology and standards have thus provided an evolutionary path from the architecture of current grids to the standardized, service-oriented, enterprise-class grid of the future *(Bieberstein et al., 2005).*

Figure 5 illustrate how a service-oriented grid might be constructed, it shows a simple grid in which web services are used both to virtualize resources and to provide other grid functions. The management console applications can be hosted in one of the organizations within the grid with the rest of the hardware and software resources being distributed within the other organizations on the grid.

In the figure, a single console is used to submit jobs and to manage the grid's resources. The console's management software contacts the registry service to discover the grid resources, and then contacts each resource's representing service to request periodic performance data and to be notified of significant changes in the resource's state—for example, if it becomes unavailable or heavily loaded. The user submits a job request to the job-submit service, which locates the grid's scheduling service and passes on the request. The scheduler locates and contacts the service that represents the requested application, and requests details of the resources that are required for the job. It then queries the registry to find all suitable resources, and contacts them individually via their associated services to determine availability. If sufficient resources are available the scheduler selects the best available set and passes their details to the application service with a request to begin execution; otherwise, it queues the job and runs it when the required resources become available. Once the job has been completed, the application service reports the result to the scheduler, which notifies the job-submit service. The job-submit service, in turn, notifies the user.

5. Conclusion

Web services and a service-oriented style of architecture are widely seen as the basis for a new generation of distributed applications and system management tools as suggested by. The model can easily be implemented by organizations that have similar type or are in the same line of business, this makes it easy to integrate the functionalities in the modules because they pass same type of messages to other services being used by these organizations within the grid. Through the grid organizations can share the services as well as the ability to make more cost effective use of a given amount of computing resources. In this paper we have introduced the key concepts, relationships and benefits of these two technologies, and indicated how they can be combined to develop highly scalable application systems that can span management and ownership domains, regardless of the hardware and software platforms deployed in each.

References

Bell, Michael (2008). "Introduction to Service-Oriented Modeling". *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley & Sons. p. 3.b.

Bell, Michael (2010). *SOA Modeling Patterns for Service-Oriented Discovery and Analysis*. Wiley & Sons. p. 390. ISBN 978-0-470-48197-4

Channabasavaiah, Holley and Tuggle, (16 December 2003), Migrating to a service-oriented architecture, *IBM DeveloperWorks*.

Yvonne Balzer,(16 July 2004) Improve your SOA project plans, *IBM*,

Enterprise SOA, (2009), Prentice Hall.

Bieberstein et al., (2005) Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap (The developerWorks Series) (Hardcover), IBM Press books, , 978-0131870024

Fabio Baroncelli and Barbara Martini, (2007), "A service oriented network architecture suitable for global computing", Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy

Latha Srinivasan and Jem Treadwel,(3 november 2005) "An Overview of Service-oriented Architecture, Web Services and Grid Computing", HP software global business unit.

Tony Shan,(2004), "Building a Service-Oriented eBanking Platform", scc, pp.237-244, First IEEE International

Conference on Services Computing (SCC'04).

Open Grid Services Architectire(OGSA) v1.0: http://www.gridforum.org/documents/GFD.30.pdf

Defining the Grid: A roadmap for OGSA standards v1.0: http://www.gridforum.org/documents/GFD.5.3.pdf.

Bieberstein et al,(2005)., Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap (The developerWorks Series) (Hardcover), IBM Press books, 978-0131870024.

World Wide Web Consortium (W3C): http://www.w3.org

M. Hadi Valipour, Bavar AmirZafari, Kh. Niki Maleki, Negin Daneshpour,(Aug 2009), A Brief Survey of Software Architecture Concepts and Service Oriented Architecture, in Proceedings of 2nd IEEE International Conference on Computer Science and Information Technology, ICCSIT'09, pp 34-38, China.

Organization for the Advancement of Structured Information Standards (OASIS): http://www.oasis-open.org.

Distributed Management Task Force (DMTF): http://www.dmtf.org

Global Grid Forum: http://www.ggf.org

Chatterjee and Webber, (2009): Developing Enterprise Web Services – An Architect's Guide (Prentice Hall).

WebServices.org : http://www.webservices.org

Microsoft® Corporation: Service Orientation and its role in your connected systems strategy: http://download.microsoft.com/download/d/2/5/d2513e64-0dcd-4ef6-89c4-c99ee117936f/serviceorientationwpa4.doc.

Reinventing the Wheel? CORBA vs. Web Services: http://wwwconf.ecs.soton.ac.uk/archive/00000216/01

CORBA: http://www.corba.org

DCOM:http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/dcom.asp

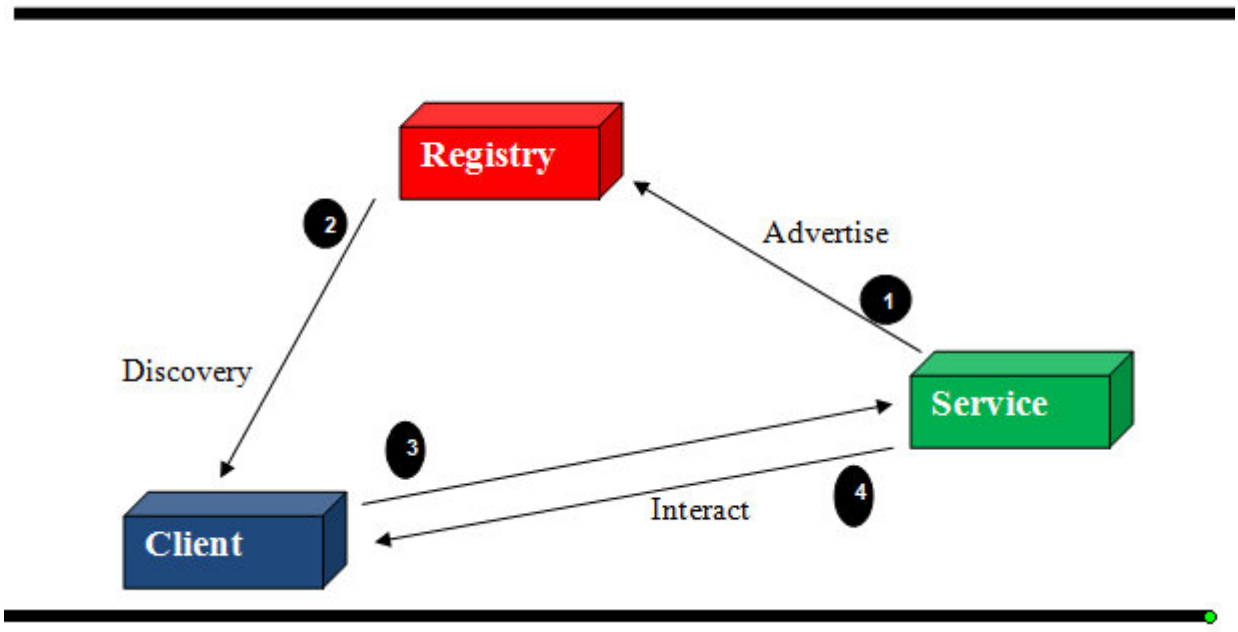

Figure 1. Service interaction in a service-oriented environment

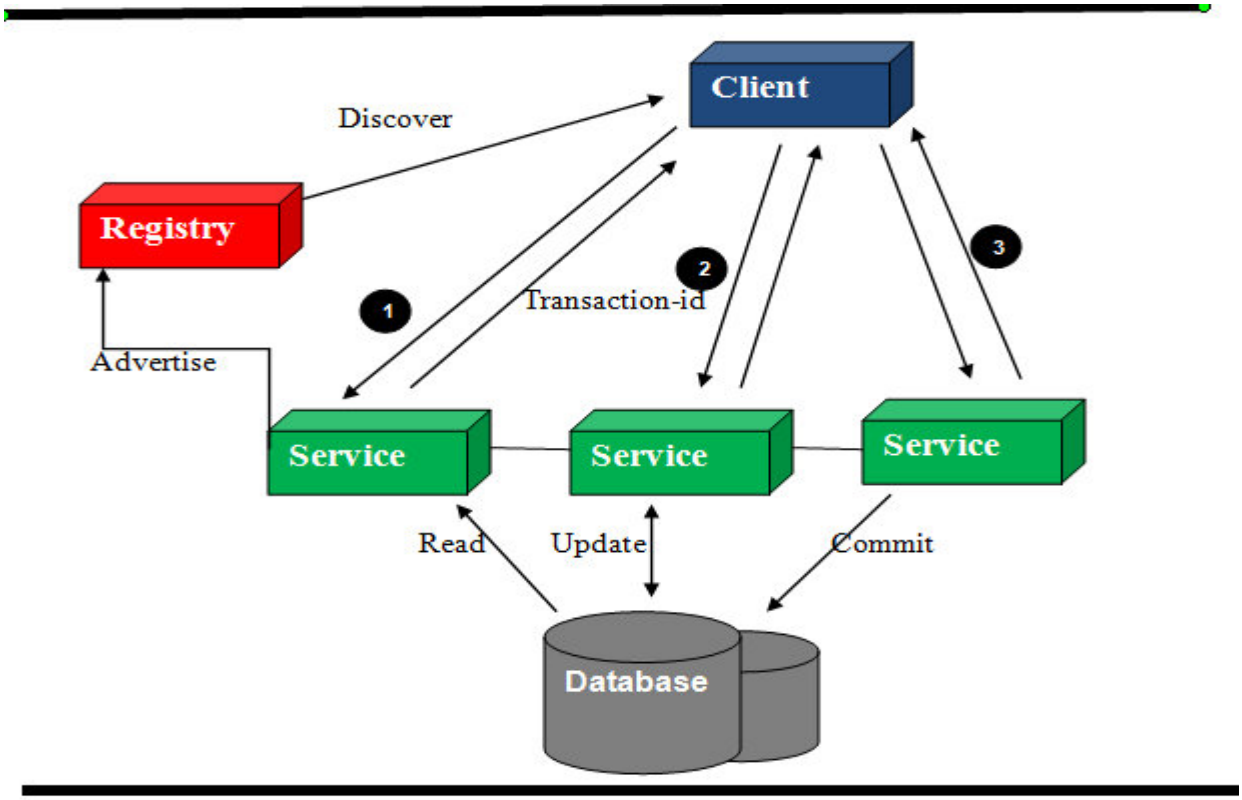

Figure 2. A multi-step client/service interaction

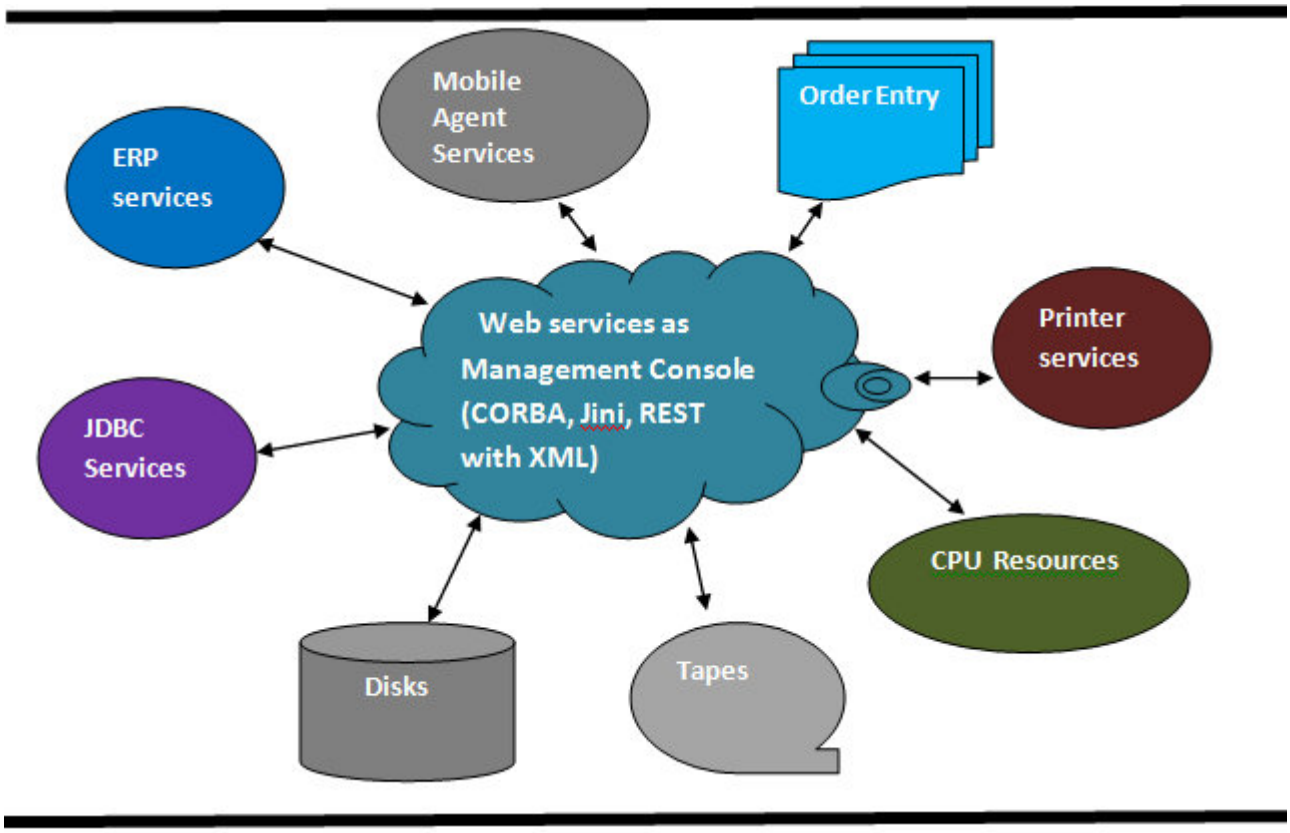

Figure 3.Management of numerous software services and hardware resources

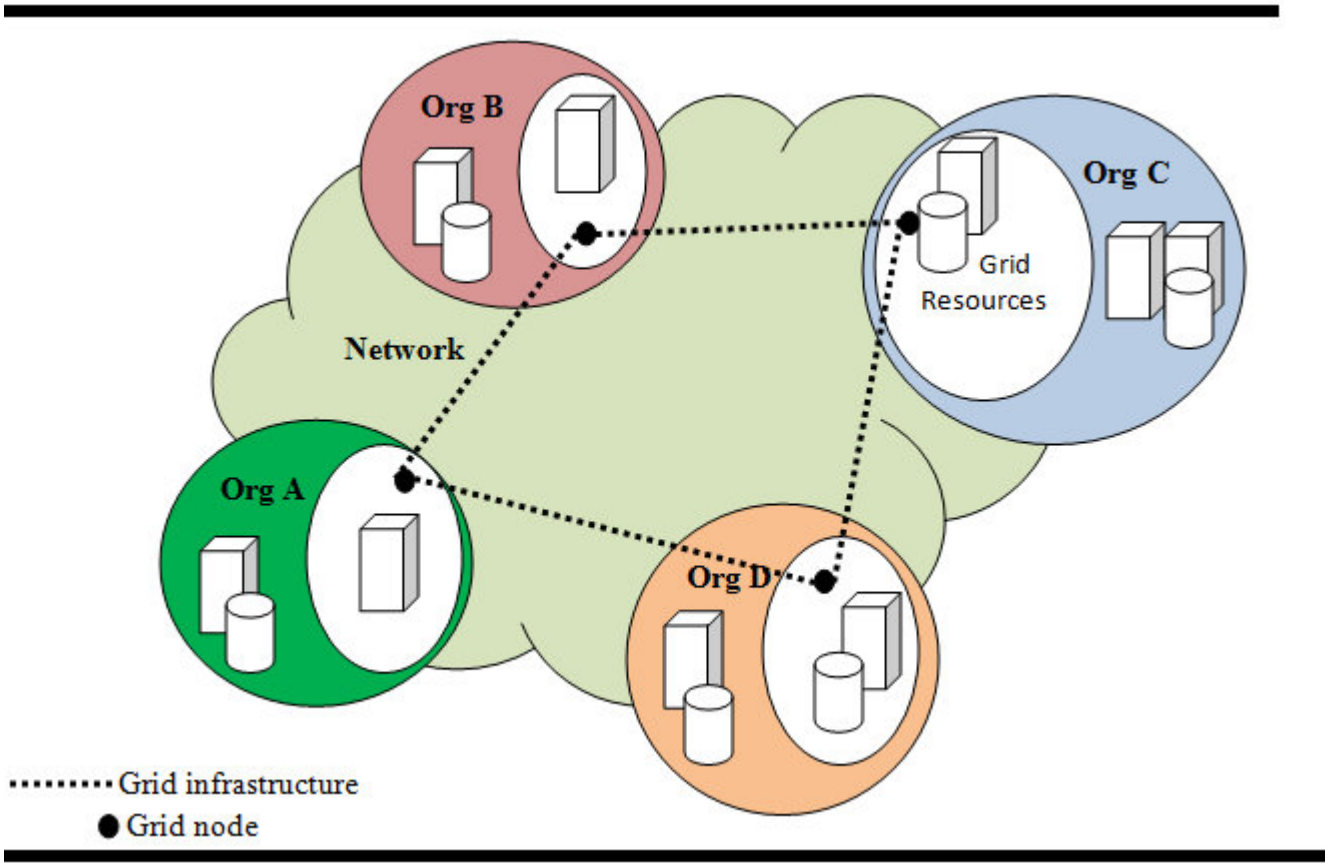

Figure 4. Grids allow resources to be shared across organizational boundaries

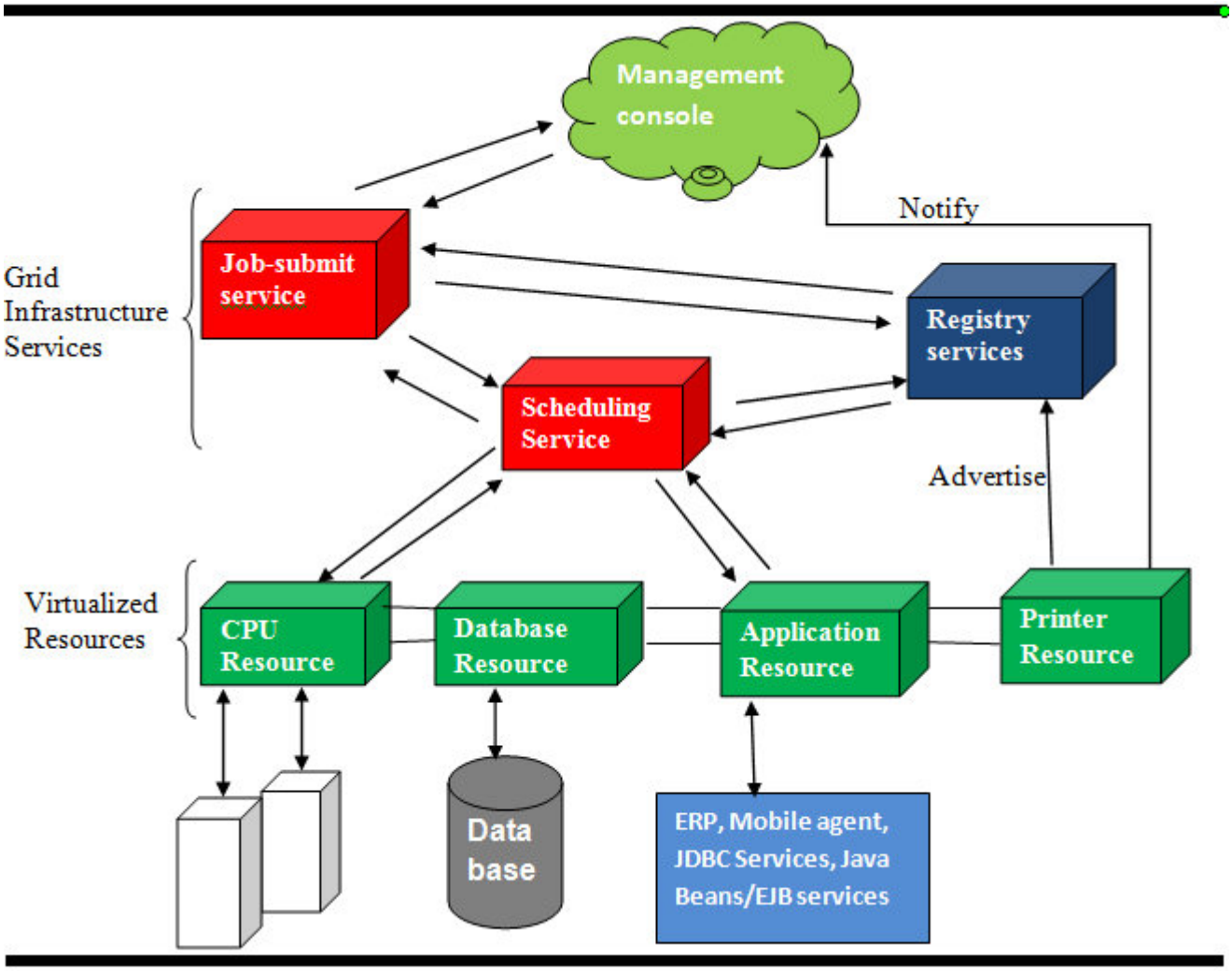

Figure 5. In service-oriented grids, web services virtualize resources and provide other functions