GUISET: A CONCEPTUAL DESIGN OF A GRID-ENABLED PORTAL FOR E-COMMERCE ON-DEMAND SERVICES

BY

ODUSOTE BABAFEMI OLUBUNMI
(CU033020070)
B.SC COMPUTER SCIENCE

A MASTERS DISSERTATION SUBMITTED TO THE DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES, COLLEGE OF SCIENCE AND TECHNOLOGY COVENANT UNIVERSITY, OTA OGUN STATE, NIGERIA.

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF SCIENCE (M.SC) DEGREE IN COMPUTER SCIENCE.

June 2011

DEDICATION

This dissertation is dedicated to God Almighty, the Omnipotent and Omniscient, the giver of life and the ingenious architect of my destiny for His faithfulness, tender-mercies and graciousness towards me.

I also dedicate this dissertation to my dear parents, Mr. and Mrs. David Odusote, whom God, my source has used as the resources for the pursuit of our academic career in covenant university.

ACKNOWLEDGEMENT

I am deeply indebted to God, my father and friend, the author of wisdom and understanding for His faithfulness and generous endowments of grace that saw me through my master's studies. My deep and sincere appreciation goes to the Chancellor, Dr. David Oyedepo and the Board of Reagents of Covenant University for the vision and mission of the University. Also special thanks to the management staff of the University: the Vice Chancellor (VC), Prof. Aize Obayan, the Deputy VC, the Registrar, the Deans of the Colleges, the Heads of Departments for their unwavering commitments to the pursuit of excellence and sound academic scholarship.

My earnest appreciation goes to my supervisors, Prof. Matthew Adigun, H.O.D, Department of Computer Science, University of Zululand, South Africa, in his capacity as my main supervisor who provided the research direction and qualitative guidance for the work. I am deeply appreciative for the research assistantship he offered me to visit his research center in South Africa, an adventure that gave focus to my research work, and gave me opportunities for immense capacity development. With all sense of gratitude, I also thank Dr Justine Daramola, my co-supervisor, for his invaluable contribution to this work. Your fatherly disposition and undeterred willingness and support to ensure academic quality is never undermined are greatly appreciated. But for your critical reviews and instructions, this work would not have been an adventure in futility. Thank you Dr Daramola. Thank you so much. You are an admirable role model and you give me reason to always aim for the stars in my pursuit. Thanks again.

Great thanks to the H.O.D, Department of Computer and Information Science, Covenant University, Prof. Ezekiel Adebiyi. Sir, you took amazing interest in my unhindered progress and advancement as a young academic. This is still astonishment to me. I appreciate your leadership, tutorship, counsels, instructions, encouragements, supports and good wills. I also thank the Director, Academic Planning, Covenant University, Prof. C.K Ayo for his fatherly disposition and timely encouragements, chastisement and tutorship. I salute your person sir. I thank Dr Nicholas Omoregbe for his guidance and supervision and good wills. To all the staff and faculty of the Department of Computer and Information Science whose values and worth is far beyond description, both my teachers and my highly esteemed colleagues, I appreciate you all and God bless you all.

Lastly, but deliberately I want to acknowledge a special category of people in my life. I call them 'My Heroes'. These are the people that God has used to nurture and make me who I am. Firstly, I specially appreciate my mother, Mrs. M.O Odusote, who has sacrificed all her life to get me to where I am today; I pray that God will grant you long life in health and vitality to enjoy all the fruits of your labor. Great thanks also to my world dearest brothers, Mr. Olumide Odusote (UK) and Mr. Abayomi Odusote, for giving your yesterday to establish my today and your today to secure my tomorrow; and to my entire siblings, Mr. and Mrs. Adewale Wahab, Mr. and Mrs. Olusesi, Mr. and Mrs. Ibukunoluwa Odusote, Mr. and Mrs. Opeyemi Odusote, Miss Moyosore Odusote and my nieces and nephews, thank you very much for your encouragements and supports both spiritually and physically.

I am also indebted to the Deputy Registrar, Covenant University, Mr. E.O Ojo. He has been my '*school father'* and his fatherly support and counsel have made my stay here a most memorable one. God bless you sir. I must also not forget to appreciate Mr. Segun Awonusi (Jay-Tee) for helping me early in life to appreciate and embrace the virtues of discipline, diligence and dedication as a student in his tutorial college, Jay-Tee Tutorial College. Sir, the positive impacts of your words and chastisements are undeniably evident today. God bless you sir!

I also want to appreciate the unmatched efforts of my post graduate lecturers, Dr. P.B Shola of University of Ilorin, and Dr. Adewole of University of Agriculture, Abeokuta. God shall reward both of you greatly. God bless and keep you all.

TABLE OF CONTENTS

CHAPTER ONE

CHAPTER TWO

CHAPTER THREE

CHAPTER FOUR

CHAPTER FIVE

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Conventional grid-enabled portal designs have been largely influenced by the usual functional requirements such as security requirements, grid resource requirements and job management requirements. However, the pay-as-you-use service provisioning model of utility computing platforms mean that additional requirements must be considered in order to realize effective grid-enabled portals design for such platforms. This work investigates those relevant additional requirements that must be considered for the design of grid-enabled portals for utility computing contexts.

Based on a thorough review of literature, we identified a number of those relevant additional requirements, and developed a grid-enabled portal prototype for the Grid-based Utility Infrastructure for SMME-enabling Technology (GUISET) initiative – a utility computing platform. The GUISET portal was designed to cater for both the traditional grid requirements and some of the relevant additional requirements for utility computing contexts. The result of the evaluation of the GUISET portal prototype using a set of benchmark requirements (standards) revealed that it fulfilled the minimum requirements to be suitable for the utility context.

CHAPTER ONE

1.0 INTRODUCTION

1.1 BACKGROUND INFORMATION

The use of computer systems for personal and corporate purposes has increased since the early 1990's and many individuals and corporate organizations have benefited tremendously from its evolution. Notwithstanding, the personal computer (PC) technology model was adjudged a failure for reasons of non-affordability and lack of extensive shareability [1].

Resource-constrained enterprises such as the Small, Medium and Micro Enterprises (SMMEs) especially in the rural areas could not easily afford the PC technology amongst other requirements to enable their business processes. Hence many of the SMMEs have to form various business clusters with the aim to engendering shareability of relevant technologies, operational equipment, facilities, etc through a "co-operative society" approach solely for socio-economic benefits [1]. With this approach, every registered member of the co-operative society: business owners, service providers, service consumers, etc can easily harness any facility (jointly owned) and some other membership benefits to enhance their business processes.

In most developing countries, e-Commerce is being adopted in large organizations, but a large number of these resource-constrained enterprises such as the SMMEs are yet to enjoy the maximum benefit e-Commerce offers [2]. Some of the reasons as identified in [3] include inability to afford the Total Cost of Ownership (TCO) of e-Commerce tools and applications, lack of Information Technology (IT) expertise, amongst others.

Moreso, in order to reduce TCO, SMMEs could pay for just the services and resources used per time, without being burdened with high operational costs and overheads. This mode of resource delivery and utilization is often referred to as On-Demand Computing (ODC) [4, 5]. On-demand computing is a paradigm that facilitates the availability and utilization of computing resources solely on per user request basis [4]. Utility computing is a form of ODC that enables resource provisioning to users through a payment model such as subscription or pay-as-you-use [4]. The term *Utility* is derived from real world provision of utility services such as electrical power, water and gas, where consumers pay for the services used, based on usage rather than on a flat-rate basis [5]. This paradigm has gained adoption in enterprise computing, where software resources and

services are accessible by users over a network, based on the user's request. This approach to software delivery can be termed software on demand [4].

The software-on-demand and utility computing paradigm of software delivery would provide SMMEs with several benefits such as the reduction of IT-related operational costs. They no longer need to invest heavily in building, owning or maintaining applications for e-Commerce services such as Customer Relationship Management (CRM), On-line Payment Processing (OPP), Report Generation and Analysis (RGA), Order Management Systems (OMS), Inventory Management Systems (IMS), etc, as they can access these services through a network and charged accordingly based on resource usage [5].

This whole scenario however, is one of the motivations for the notion of a Grid-based Utility Infrastructure for SMME-Enabling Technologies, GUISET [1], that was proposed by the Computer Science Department of the University of Zululand, South Africa. GUISET is a research agenda based on adopting the utility approach of service-oriented architecture (SOA) for service delivery. It leverages the success of handheld mobile devices whose technology is more affordable and shareable with mobile mode of utility computing [6].

GUISET therefore aims at technologically enabling the business activities of SMMEs by facilitating an affordable access to relevant technologies on a *pay-as-you go* basis. The technology is conceptualized as a package of mobile e-Services; therefore *web presence* is the starting point for the enterprise being enabled. This research agenda envisages a future in which service providers will competitively provide computing services at a varying cost compared to the currently fixed cost to clients based on their quality of service (QoS) requirements [1].

GUISET is not an application but, a SOA-based utility infrastructure that is conceptualized to accommodate services as a suite of service-oriented on-Demand Applications such as applications developed elsewhere by different service providers across various domains such as: e-Commerce, e-Tourism, e-Health, e-Business, e-Government, etc [7]. Moreso, as a grid-based utility infrastructure, GUISET is meant to provide an enabling operating environment through a *portal system* for every prospective utility service provider and customers willing to form or join any user business cluster or community [7]. The GUISET portal therefore is an interface meant to provide a

street level entrance into a *bring-and-share* mode of utility computing [7] for every member or prospective member of the business community.

Essentially, a portal is simply a Web-based application that acts as a gateway between users and a range of different high-level services [8]. A typical first generation portal is a three-tiered architecture, consisting of an *interface tier,* of a Web browser, a *middle tier* of Web servers, and a *third tier* of backend services and resources such as databases, high performance computers, storage, specialized devices, etc [9].

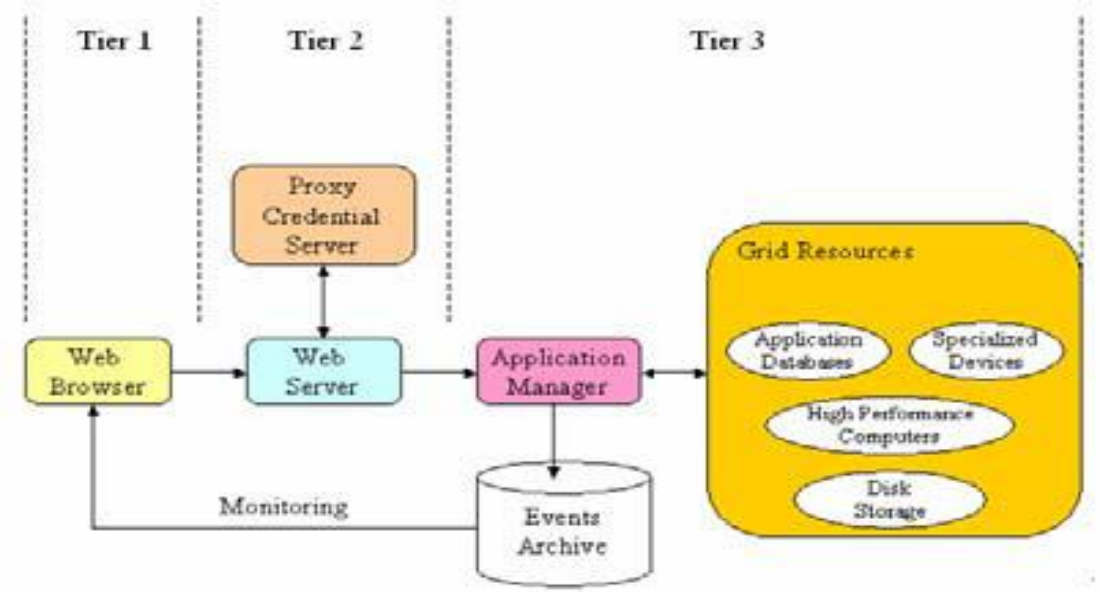


Figure 1.1: **A Simple Portal Architecture** [9]

This generation of portals suffered a number of setbacks due to their lack of customization, restricted grid services, and static grid services [9]. While there are limitations with the first generation portals, the experiences and lessons learned developing portals have paved the way for the development of more sophisticated and user-friendly portals.

In order to overcome the limitations of the earlier portals, the *portlet technology* was introduced, promoted and have been adopted for building new generation portals, often referred to as the *second generation portals* [2, 9]. A so-called second generation portal normally consists of different portlets to process user requests for various services and generate dynamic contents from the responses [10]. From a user's perspective, a portlet is a window or mini user interface in a portal that provides a specific service, for example, a calendar or news feed (see figure 1.2).

Moreso, from an application developer's perspective, a portlet is a pluggable user interface, software component that are managed by a portlet container, which handles user requests and generates dynamic content in a web portal [9].

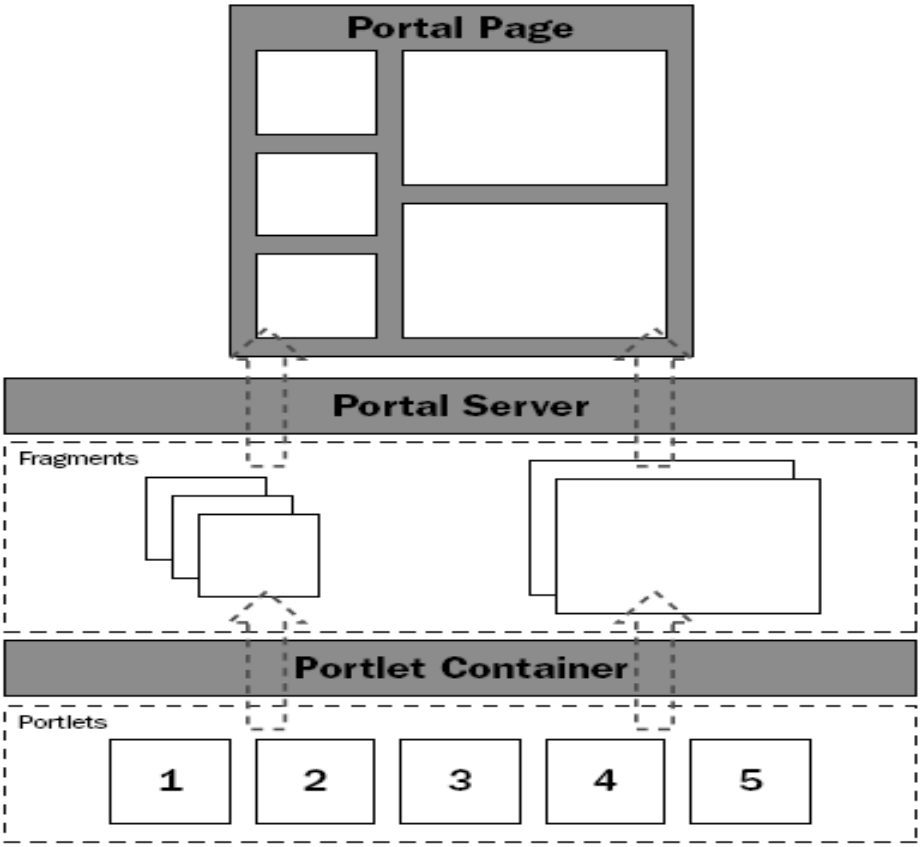


Figure 1.2: **A Portal Composed of Five Portlets** [9]

Grid-enabled portals (portals based on Grid technology) build upon the familiar Web portal model to offer virtual organizations (VO) or communities of users a single point of access to computational resources such as clusters, data servers, applications, scientific tools, and computing services [11]. A grid amongst many ways can be defined as a collection of heterogeneous distributed interconnected computing and data resources that provides large scale virtual resources with an appropriate single user interface [12].

Grid-enabled portals are emerging technologies and are currently gaining a lot of popularity. They are receiving more attention among developers and programmers due to their ease in development, richness in functionality, pluggable architecture and customization of interfaces [10]. They are web-based applications that act as a gateway between grid users and a range of different grid

resources and services [10]. They provide uniform access or single point entry to underlying grid services and resources.

Grid-enabled portals are currently developed using *portlets*. Portlets can be thought of as a miniature web application that is running inside a portal page alongside a number of similar entities [13]. They are user-facing, multi-step, interactive modules that can be plugged into a portal application [10]. Portlets rely on the overall portal framework to access user profile information, participate in a presentation interface, communicate with other portlets to access remote contents, lookup credentials, and store persistent data [9].

Portal frameworks are development platform for portal development. They are design structures that contain various modules, methods and software features used for developing specific portal [8, 10]. With the popularity of portals today, there are many portal frameworks that are available as open source and the list of these open source frameworks is all the time increasing.

A typical grid-enabled portal has the following capabilities [10]:

1. Registration of Users.
2. Accessibility of various users to a range of underlying grid services and resources.
3. Provision of personalization, single sign-on (SSO), aggregation and customization.
4. Robust Application Integration
5. Security
6. Redundancy, failover and Load balancing. etc.

Portlets are used in portal development as self-contained pluggable user interface components to encapsulate one or more applications or services that can be aggregated and transferred as information to a presentation layer of a portal system [14]. This new approach to portal development takes its cue from the concept of Service Orientation [10].

Service Orientation is a paradigm that utilizes services as building blocks to enable the development and flexible composition of distributed applications that are realizable through the service-oriented architecture, SOA [15]. It is an architectural paradigm for developing and deploying application quickly and cost effectively. SOA is an architectural paradigm and a discipline that may be used to build systems or infrastructure enabling those with needs

(consumers) and those with capabilities (providers) to interact via services across disparate domains of technology and ownership [15]. It is an architectural model for building systems based the interaction of services. SOA is now used to support grid-enabled portals [16].

SOA applications can be developed using one of the evolutionary software development approaches known as Component-Based Development (CBD). CBD enables development of software application by assembling and use of existing components. These components can be acquired by leveraging legacy systems, as commercial-of-the-shelf, COTS systems and some others are basically open source, from a third party developers or vendors, developing components in order to enable reusability. This facilitates shorter time to market, reduced cost, and increased reuse [17]. In SOA, software components are encapsulated as Services. A Service is a software component that enables access to one or more capabilities with prescribed interfaces [18]. Some properties of Services include: Loose Coupling, Reusability, Autonomy, Discoverability, etc [2, 17].

The development of grid-enabled portal is also taking advantage of the increasing advancement in Web technology [19]. Beyond providing a medium of access to various distributed resources and services to users, portals are also developed to enable community user interactions and forum, social networking, etc. This new dimension to grid-enabled portal development is based on the Web 2.0 technology.

Web 2.0 is a Web technology that results from the advancement of the Web from being a document delivery system to an application platform [20]. Sometimes it is called "*Web as platform*" [20]. It is a more socially interactive platform where users can network and collaborate for socio-economic benefits, giving various users an opportunity to contribute to the community as much as they consume [20].

There are a number of Web-based services and applications that demonstrate the foundations of the Web 2.0 concept, and they are already being used within the grid portal context too. These are not really technologies as such, but services (or user processes) built using the building blocks of the technologies and open standards that underpin the Internet and the Web. These include blogs, social networks, wikis, multimedia sharing services, content syndication, podcasting and content tagging services [20].

1.2 STATEMENT OF THE PROBLEM

Conventional grid-based portal designs have been largely influenced by the usual functional requirements such as security requirements, job management requirements and grid resource requirements [14, 21]. However, the 'pay-as-you-use' service provisioning model of utility computing platforms mean that additional requirements must be considered in order to realize effective grid-enabled portal designs for such platforms [4, 5, 22, 23]. Consequently, the following research question suffices: *What are the relevant requirements that must be considered for effective design of grid-enabled portals for utility computing contexts?*

1.3 AIM AND OBJECTIVES OF THE STUDY

The aim of this research work is to investigate the relevant additional requirements that can be catered for in the design and development of grid-enabled portals for utility computing contexts.

In order to be able to achieve this aim, the following objectives were formulated:

1. To study issues on current grid-enabled portal frameworks and development toolkits as profiled in the literature in order to identify a research goal.
2. Based on a thorough literature review, to identify a number of relevant additional requirements that can be catered for in the design and development of grid-enabled portals for utility computing contexts.
3. To design and develop a grid-enabled portal prototype for GUISET with some features that caters for these identified requirements.
4. To conduct an evaluation of the grid-enabled portal prototype using a set of benchmark requirement standard, and also the usability evaluation of the portal prototype.

Table 1.1: **The Research Objectives**.

S/N	OBJECTIVES	PROPOSED METHODOLOGY
1	To study issues on current grid-enabled portal frameworks and development toolkits as profiled in the literature in order to identify a research goal.	• Literature Review • Investigation & Evaluation of Existing Tools
2	Based on a thorough literature review, to identify a number of relevant additional requirements that can be catered for in the design and development of grid-enabled portals for utility computing contexts.	• Literature Review • Requirement Elicitation
3	To design and develop a grid-enabled portal prototype for GUISET with some features that caters for these identified requirements.	• Proof of Concept Prototype • Case Study
4	To conduct an evaluation of the grid-enabled portal prototype using a set of benchmark requirement standard, and also the usability evaluation of the portal prototype.	• Benchmarking • Usability Evaluation • Case Study

1.4 RESEARCH METHODOLOGY

The research approach of this study will be both theoretical (descriptive) and formulative. The former entails a thorough review of literature on relevant techniques, methods and technologies used in grid-enabled portal designs and development. This include: SOA, CBD, grid technologies and Web 2.0. The motivation for the selected approaches is highlighted below:

- SOA: The various applications or components would be exposed as services, and SOA is the most appropriate option to achieve that [17]. SOA is an architectural model for building systems based the interaction of services and it is also currently used to support grid-enabled portals [16].
- CBD: To easily achieve the building of an efficient grid-enabled portal from a set of interconnected software tools or components that perform specific task [24].
- Grid Technology: It is the first type of distributed system to fully actualize interoperability. It however, has greater future prospects for successful implementations [21].
- Web 2.0: For collaborations and community enablement through user communities and groups, online forums and discussion boards, chat groups and e-mails, blogs and white boards, etc [20].

A formal design model of the proposed system was built using the Unified Modeling Language (UML). The use case diagrams were used to capture the system requirements. The class diagram and collaboration diagram were used to design the various entities and their interactions within the system. The sequence diagram, activity diagram and state diagram were used to model the activities and business logic of the system. Furthermore, in addition to the traditional requirements for design of grid-enabled portals, we identified a number of relevant requirements and implemented a grid-enabled portal prototype for GUISET with certain features that cater for these additional requirements. The portal prototype is built using Liferay 5.2.3 portal tool kit [10, 25, 26] bundled with Tomcat 6.0.18 as the core components adopted for the implementation.

An evaluation of the grid-enabled portal prototype was conducted using a set of benchmark requirement standards, and also the usability evaluation of the portal prototype in a controlled experiment by a group of experienced users.

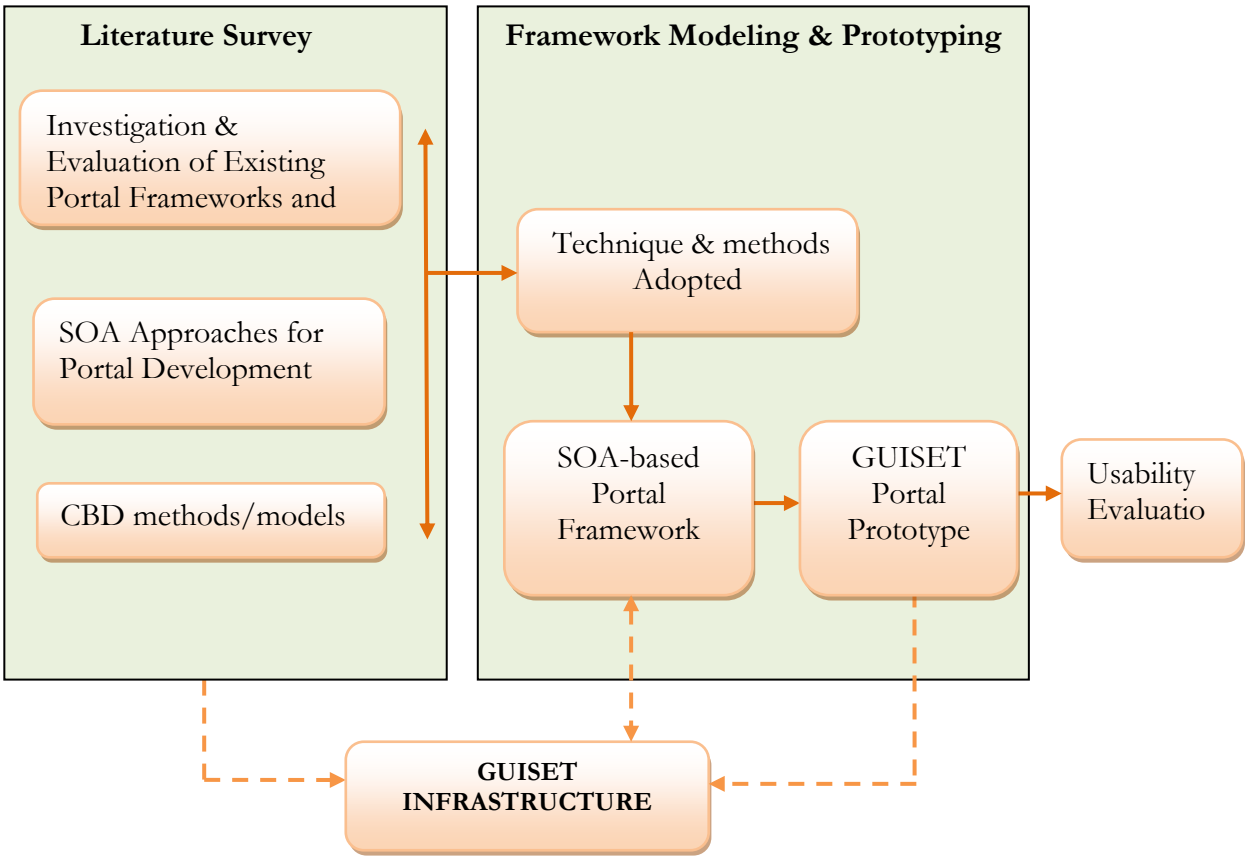


Figure 1.3: **A Schematic Diagram of the Research Framework**

1.5 SIGNIFICANCE OF THE STUDY

This study has relevance to the practice of grid-enabled portal development, and also provides a model for national economic development. This is because:

1. It reveals the minimum relevant requirements apart from the traditional requirements that must be fulfilled to realize effective design of grid-enabled for the utility context, and

2. The implementation of the GUISET portal offers a usable prototype that facilitates the realization of ODC platform for improved wealth creation and affordable access to scarce and expensive computing, particularly among SMMEs and rural-based businesses.

1.6 CONTRIBUTION TO KNOWLEDGE

The contribution of this work is that it presents a case study of grid portal design and development for utility computing contexts in an elegant and repeatable way. The perspective of grid portal for utility computing embraced by this study is not yet common in the literature, hence it is valuable for the advancement of literature and industry practice.

1.7 LIMITATIONS OF THE STUDY

This research work is part of a bigger on-going research endeavor embarked on by the Center of Excellence for Mobile e-Service, Department of Computer Science, University of Zululand, South Africa, which aims at building an evolutionary system - GUISET infrastructure. In this work not all identifiable additional requirements for design of grid-enabled portals have been considered. The scope of this work is limited to those that do not require the expensive third party infrastructure and usage access rights such e-Billing and e-Payment. Therefore, only a selected set of additional requirements have been considered and not all that is possible.

1.8 DISSERTATION OUTLINE

The Remainder of this dissertation is organized as follows: Chapter two contains a review of relevant literatures on state-of-the-art SOA concepts, Component-Based Design techniques; Web 2.0 concepts and technologies were done. In view of describing this work in the light of what currently exist in the research community an exploration of existing Grid-enabled portal frameworks, tools, technologies and standards was also done. Chapter three presents the requirement analysis and designs. Chapter four discusses the implementation of the grid-enabled portal prototype for GUISET with relevant features that cater for the identified additional requirements. Chapter five presents a report on the evaluation of the portal based on a set of benchmark requirements standard. The summary, conclusion and recommendations for future work were also presented in this chapter.

CHAPTER TWO

2.0 THE REVIEW OF RELEVANT LITERATURES

In this chapter some relevant theoretical foundational concepts that pertain to this research work were explored. Modern and relevant computing theories that suit the research concept were also explored. Some of these theories include: portal-oriented architectural framework, concepts of grid-enabled portal & portlets, service-oriented architecture, web service concepts, mobile mode of utility computing to service delivery, Web 2.0, concepts of user interface, etc. We also did An investigation on different existing major portal frameworks, tools and technologies for grid-enabled portal development was done and suitable tools for achieving the GUISET portal functionality and capabilities were adopted.

.

2.1 GRID-BASED PORTAL-ORIENTED ARCHITECTURE

2.1.1 Grid-enabled Portals with Portlets

Portals are receiving more interest and attention among software developers due to their ease in development, customization of interfaces, richness in functionality, and pluggable architecture [10]. A portal is a software application that provide uniform medium through which the users secure access to an online environment of various resources and services [27].

Grid-enabled portals (portals based on Grid technology) are built upon the existing Web portal model to provide virtual organizations (VO) or communities of users a single and uniform medium of access to computational resources: software application and computing services, data servers, clusters, and scientific tools [11]. They are similar to websites but the main disadvantage of websites is that once they are developed and configured, it is very difficult to adapt them to new applications [14].

A Grid-enabled portal acts as an access medium between grid users and a range of different grid resources and services [10]. It provides a uniform access or a single point entry to these underlying grid services and resources [25]. It provides personalization, single sign-on (SSO), aggregation and customization features [10]. A so-called “second generation” portal normally consists of different portlets to process user requests to the Grid services and generate dynamic content from the responses [25]. Portlets can be thought of as a miniature web application that is running inside a portal page alongside a number of similar entities [36]. Portlets are used in portals as self-

contained pluggable user interface components to the services [5]. They are user-facing, multi-step, interactive modules that can be plugged into a portal application [36].

Portals employ portlets as pluggable and related components that provide a presentation layer and user interface, allowing Grid portals to have a complete and easy to maintain structure by reducing the element of cohesion [28]. Portlets are managed by portlets containers and they process requests from the grid clients through the web and generate dynamic contents that create the portion of a web page or interface. The content generated by a portlet is also called a "fragment" [14]. A fragment is a piece of markup such as HTML, adhering to certain rules and can be aggregated with other fragments to form a complete document [14]. The content generated by a portlet can be aggregated with the contents generated by other portlets to form a portal page. These portal contents may vary from one user to another depending on the user configuration for the portlet. A user can select the portlets he needs and even rearranges the positions of the portlets. These configurations will be saved persistently for this user. Thus, the user can configure a portal environment that suits him best [14].

Portlets have become an increasingly popular concept to describe visual user interfaces to a content or service provider [29]. Technically, they represent modular, reusable software components that may be developed independently of the general portal architecture [8], and offers a specific set of operations. This concept has brought immense benefits to this field in that, portlets developed by different groups can be easily reused if they conform to the same standard such as Java Specification Request, JSR-168 [18]. It also can be easily removed from or added into the portlet container by changing configuration files or by using certain tools provided by the portal framework.

However, portals can therefore be either *portlets-based or non portlets-based* [25]. *Portlets-based portals* are web component that generates fragments – pieces of markup (e.g. HTML, XML) adhering to certain specifications. Fragments are then aggregated to form a complete web page. Figure 2.1 below depicts a generic portlet-based portal architecture.
Non Portlets-based portals are usually built based on 3-tier web architecture: (i) Web browser, (ii) application server/Web server which can handle HTTP request from the client browser, and (iii) backend resources that include computing resources, databases, etc. Many of the early grid portals

or early versions of grid portals are non portlets-based, for example, the Astrophysics Simulation Collaboratory (ASC) portal [30], UNICORE [31], etc.

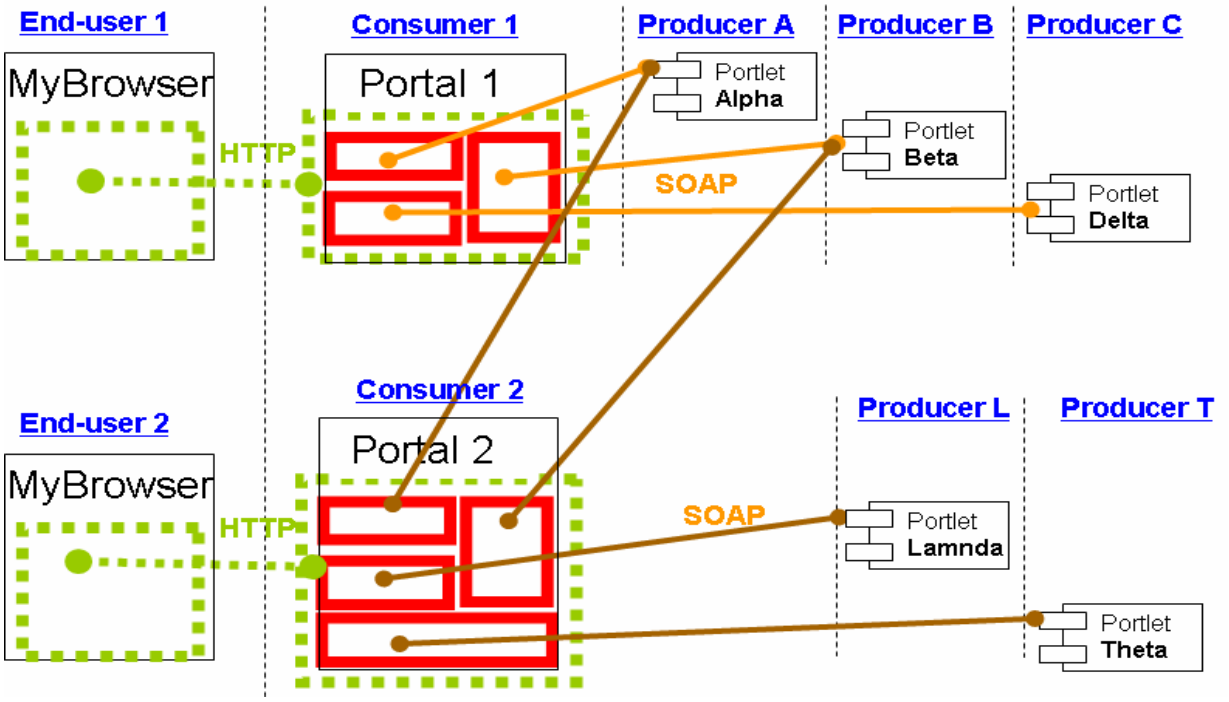


Figure 2.1: **A Portlet-based Portal Architecture** [28].

The development of portlets-based portals has brought many benefits to both end-users and developers, and this is getting more recognition. An investigation of a set of related works done in this research work revealed that, a combination of Grid technologies and portal technologies has been substantially achieved in order to make grid-enabled portals more flexible and to improve their reusability. This is as reported in [14].

Grid technology however, has matured to the point where many different communities are actively engaged in building distributed application infrastructures to support discipline-specific problem solving. It enables coordinated sharing and use of distributed software resources, hardware resources and information in virtual organizations [8]. A Grid is a combination of network infrastructure and software framework delivering computing services based on distributed hardware and software resources. This distributed infrastructure is based on web and web service technology that brings data, various tools and applications to the users in ways that facilitates

collaborative works [21]. Grid enables communities (VO) to share geographically distributed resources in the absence of central control, omniscience, and trust relationships [21].

From a user's perspective, a Grid based portal system usually should provide the following functionalities or services:

A. Authentication Service.

Any user that intends to log onto the portal using a web browser is first authenticated by means of a user-id and password. Once the user is authenticated, it is the job of the portal server to act as the user's proxy in most grid interactions, by obtaining a proxy certificate that it can use on behalf of the user. The standard approach is to have the user submit a proxy to a *MyProxy* [32] server. The proxy can then be retrieved by the portal server from the MyProxy server and holds the proxy for the duration of the user's session. This process of making the user manage a key pair and submit proxy certificates to a MyProxy server is extremely unpopular with users.

B. Remote File Management.

A central requirement for the portal is the access to file metadata directories and remote file archives. Simple tools for Grid file transfer protocol, (Grid FTP) are essential here, but many files are likely to be managed by a virtual data system, where data is cataloged and staged by back-end grid services.

C. Remote Job Submission and Monitoring.

The ability to submit jobs to the grid infrastructure for execution and monitoring is a classic requirement for portals [14]. Users with specific resources allocation want to be able to see job queues on those resources and consult scheduling assistants. They need to be able to keep track of job execution and understand when things fail by reading logs. The most ambitious grid applications are those whose execution is defined by complex workflows [14].

D. Access to collaboration.

Resource sharing is also vital within any VO. The ability to use real-time collaboration tools as well as asynchronous collaboration is emphasized here.

Grid systems have some common services or functionalities, and these services are independent from each other somewhat. Thus, one or more Grid services can be encapsulated into a *grid portlet* [14], and enable portlets to interact with each other if needed. An authentication portlet can be designed to authenticate users. A Grid FTP portlet can enable users to access remote files if they are authenticated users. A job submission portlet allows users to summit jobs and a resource monitoring portlet to enable users to check remote resources such as CPUs or memories.

Collaboration portlets such as chatting portlet allows users to collaborate with each other. Grid users can choose the portlets they need to configure their own portals.

2.1.2 Grid-enabled Portals Models

Grid-enabled portals exists essentially in two different classes – based on their *execution form* and *ubiquitous access* (presence everywhere) [8], and *User Control* [33]. However, depending on their *execution form*, and *ubiquitous access*, they can be classified into two models. The Open Grid Computing Environment, OGCE [33] and HotPage [34] are examples of portals that provide ubiquitous access; the user handles certain aspects of the grid portal including accounts with high performance computing resources, have setup and configuration responsibilities, and others. On the other hand, portals that are exclusively controlled by the administrator correspond to the second model. An example of the second is Punch In-Vigo, where the administrator is responsible for almost all the manufactured Grid portal [35]

Furthermore, according to [36] portals can again be classified into two models depending on *user control*. Some portals provide access to high-performance computing resources at anytime; users have control access to any resource to install what they need. In this type of portal model, users have knowledge of the grid middleware itself, and therefore are required to implement changes. Conversely, in the other portal model, access and resource control is assigned to a specific user administrator who is responsible for installing, configuring, or running applications and middleware. Users (clients) only interact with the portal and are unaware of the procedures running under it.

The second model of the latter class provides a more stable and controlled setting, because the access and resource control of the portal is the responsibility of the administrators and users only perform the operations they need, knowing that the level of knowledge of the administrator, a specialized user, is much higher than that of the users although they know the Grid works partially in the first model [8]. These two models have advantages and disadvantages therefore; the model that best suits the needs of what wants to be deployed is the one that should be used.

2.1.3 Grid Portlet Services

The grid based portlets contain portlets services that offer a high-level application programming interface (API) and model of the grid, enabling developers to reuse the functionality offered in grid portlets to develop custom grid based portal applications [11]. The Grid Portlet Services API is a collection of Java interfaces and extensible base classes that built upon simple concepts to provide more complex services, resources and tasks [11]. These classes can be supported with Globus Toolkit 2 (GT2), GT3, GT4 and other service-oriented technologies [37] in order to provide support for persisting information about resources and tasks performed by users on the Grid.

The grid portlet services API is further supported by base implementations that make it easy to develop support for particular grid middleware technologies. Because GT2 is the most widely deployed and supported version of the Globus toolkit [37], grid portlets is distributed with implementations for utilizing GT2 resources, such as the Globus Gatekeeper [38] and Grid FTP [39] However, support for Open Grid Services Architecture, (OGSA) [40] and Web Service Resource Framework (WSRF) [41] based resources are offered in the GT3 portlets and GT4 portlets applications respectively. These applications provide GT3 and GT4 implementations of the Grid Portlet Services API and are distributed separately.
Within a portal a number of internal services are needed to address of issues of the coordination of tools (portlets) within an overall framework. Methods can be provided as an "internal" class library, which resides alongside the portlet and service APIs. Each portal framework could have the same, or a different set of tools, but the way they are integrated may differ between user groups. Alternatively, the services could be federated and available via Web Services calls to specialized servers elsewhere in a virtual research environment.

Some example portal integration services are now listed [11]:

- ***Session Management*:** This involves the management of a session key and related issues. It requires database access for storing and retrieving other items relevant to the session. User can authenticate and start a new session or revert to a previous one. The service can open and close sessions and log the state of a session from. Features like rollback and replay, including personal workflow, can also be available.

- ***Authentication*** using MyProxy, which is a repository of valid proxy certificates for authenticated users. The portal can download these for delegation to trusted external services.

A service can also check that certificates are, for example, still valid and refresh them if not. Part of the integration API would allow storing and retrieval of the proxy into the portal database for later use. This will be done using a session key and user id, (UID) (e.g. Distinguished Name (DN) or unique e-mail address). Having the certificate associated with the session key allows authorization issues to be tackled, e.g. using subsidiary certificate or another method.

- ***VO Management*** that could for example be based around a project which would typically have its own portal and Grid. VO users will have been authenticated and have received a digital identity (certificate). They are then given rights based on the roles they are taking in this VO and thus can be authorized to access services.

- ***Integrated State*****:** This is related to the need to manage data related to state information for a portlet UID. There is a general need to develop the concepts related to integrated state. For example:

    - State can be used as an event trigger,
    - State needs to be logged for session management or workflow,
    - What states can portlets and services have which are meaningful for rollback and replay?
    - Service and Portlet location, which can be published, queried, and looked up in a registry. This also requires semantic support as it is import to annotate service information with further information such as what the service does and why.

- ***Portal Preferences***, which can be built up from a "preferred set" of services and portlets and be based on usage. This service can also log semantic information and build a related ontology. The service extends the idea of a workspace toolset allowing dynamic semantic/function-driven choice.

- ***Semantic/Ontology Support*** for information about services and portlets in the framework. These services will be used for decision support and choice, augmenting stored preferences. These services would not cover generic semantic issues, which would need separate tools.

- ***Workflow*** via directed links between components (typically graph based). An event mechanism is used to trigger actions within portals and attached services. The graphs within

the portal will be mostly predefined, but with constrained facilities to swap in and out components and provide additional inputs at decision points.

- **Trails and Personalization** could involve logging of usage for off-line mining and analysis, e.g. for developers to improve presentation, ease of use, and optimization.

- **Inter Portlet Communication and Event Management:** This will provide message-based communication mechanism between portlets, possibly with event triggers and asynchronous handlers.

Grid Portlets also defines several portlet service interfaces; these services include [11]: *resource registry service, resource provider services, credential manager service, credential retrieval service, logical file browser service, and job submission service.* The various portlets leverage the above grid portlet services to provide a generic, yet powerful set of user interfaces for using the resources on the grid. These portlets include [11]: *resource registry portlet, credential manager portlet, resource browser portlet, credential manager portlet, file browser portlet, file activity portlet, and job submission portlet.*

2.1.4 Grid-enabled Portal Frameworks

In [8], portal framework, middleware, a set of Java components, servlets, and portlets, were identified as the design components of grid portals, including technical differences depending on the architecture, structure, functions and components. Frameworks are designed structures that contain various modules, methods and software features whose functions is to serve as template for developing applications derived from the same frameworks [10].

Conceptually, frameworks are composed of two sections: the '*Core'* and the '*Slots'* [8]. The core represents *classes, libraries, methods* and *modules* that are equal and that serve as the basis for all applications, thus, all applications generated by the same framework will have the same basic features stored in this layer. The second section, *Slots* represent the elements or features that can be adapted, added, or simply ignored in the application. Slots can be imagined as checklists where additional methods and functionalities that the application could have and the elements in the slots that are not required for the framework to function can be chosen [8].

According to [8], frameworks are also characterized by different spots: *Frozen Spots, Hot Spots, and Extension Spots.* The *Frozen spots* represent those points of the framework that are neither extensible nor adaptable. Frozen spots are the basic components of the framework. The *Hot Spots*

are highly extensible, adaptable and configurable, while the *Extension Points* are the segments that might link hot and frozen spots, or two hot spots.

Portal frameworks are development platform for portal development [10]. They are design structures that contain various modules, methods and software features used for developing specific portal [8, 10]. In a typical Service-Oriented Architecture, SOA [15], the portal framework is depicted as an extra layer in the architecture that provides a standard (presentation) interface for business logic independent of programming languages or platforms [10]. It is responsible for providing the required resources and environment for proper functioning of the components plugged into it.

In literal sense, a portal framework provides a skeleton to plug and play various portlets [16]. At its core, there is a universal API built on the top of the application architecture. Unlike the 3-layered (database, application logic and interface) architecture [9] for traditional application development, a portal framework has this fourth presentation layer that sits between the application logic and the user. The portal, apart from being used to present the application logic contained in software components/agents, it can as well be used to coordinate different loosely coupled services into a single concrete service, by providing the related gluing framework [10].

Furthermore, the responsibility for message flow from users to services and for inter-portlets communication rests with the portal framework [9, 42]. The messages which can be stateless or stateful, but are normally stateless as software agents are context independent. However, in order to facilitate multiple interaction per user, the framework either adds state information to the message or stores the state information in a persistent way thus, removing the need for services to maintain state when invoked from different portlets [9]. Hence, the failure of any service does not result in loss of state as the state of services is known [9]. A service providing the software agent can even be replaced dynamically during the execution with another equivalent one. This potentially makes recovery from partial failure relatively easy and services seen by the user can be made reliable [12].

Traditionally, stateful services would demand that both the service provider and consumer share the same consumer-specific context [9, 10]. This in turn reduces the overall scalability of the service provider component and increases the coupling between the service provider and consumer making switching of service providers more difficult [10]. Maintaining state through the portal

framework and aggregating services as portlets is however not a large overhead and the main purpose and benefits of the SOA are then not compromised [10]. The ability to use stateless idempotent services results in less overhead on the service-providing component and uniform behavior when components are used in different ways. These core functionalities in a portal framework make it a most appropriate companion to the SOA [10].

Another component of a grid portal that is worth mentioning is the *middleware.* A good example is the In-Vigo [35]. The middleware represents the connectivity software that serves as the intermediary between the various platforms in which the portals can be [8]. It is platform-independent and application-independent; it interacts with physical resources such as built applications with the aim of integrating various applications from different platforms and/or operating environment [8].

There are various set of development tools with features that are incorporated in the design and development of grid-based portals. The *Grid Portal toolkit* [41] and *Globus Toolkit* [37] are two of these tools. The *Grid Portal toolkit* is a development tool for creating web portals and applications for grid infrastructures [41]. It was developed by the Texas Advanced Computing Center (TACC) [27], and its security model based on Globus GSI (Grid Security Infrastructure) for a single login and authentication to remote servers. It allows to automatically incorporate the use of MyProxy, to be possible to authenticate users through identification and credentials for a short period of time [41], using the JSR168 standard.

The *Globus Toolkit* on the other hand is used for building Grids. It was developed by Globus Alliance in 1998, the most popular versions incorporate GT 3.x OGSI (Open Grid Service Architecture), GT 3.9 and GT 4.0 supported by WSRF (Web Service Resource Framework) [14]. Globus Toolkit includes features that can be used either independently or together with the application such as: security software, information infrastructure, resource and management, fault detection, among others.

In order to design a grid-enabled portal, a chosen portal framework, with a middleware, java components or development toolkits is required. The Portal frameworks are structures defined for specific portal models, while middleware facilitates communication between structures, and toolkits allow the development of portals from pre-developed tools [8]. A portlet-based grid enabled portal consists of portal framework, portlet container and grid portlets which interact with the grid tools including Globus toolkit and grid middleware.

2.1.5 Portals Development Standards and Technologies

There are various generally embraced technologies designed to standardize how portal components are being designed and developed. According to [16], all frameworks are currently incomplete and deficient in terms of extensibility and reusability across various application domains. In order to address this limitation, certain standards were formulated and have been adopted by standard organizations consisting of major industry players and research groups with the hope of improving the current situation. These standards are otherwise referred to as *portlets specifications* [13, 18] and this work considered two of these major standards - The *Java Specification Requests* 168 (JSR 168) [18], and *Web Services for Remote Portlets* (WSRP) [13].

These specifications define a common portlets application programming interface (API) and infrastructure that provide facilities for personalization, presentation, and security [13]. Portlets using this API and adhering to the specifications will be product agnostic, and may be deployed to any portal framework that conforms to the specifications. This helps to facilitate the support for multiple portal applications thus accommodating the various needs of the users. The compliant portlets can be deployed to all compliant portal frameworks without extensive engineering changes [18].

Moreso, the specifications define how to leverage Simple Object Access Protocol (SOAP)-based web services [43, 44], that generates mark-up fragments within the portal application, by defining a set of common interfaces, thus allowing portals to display remotely-running portlets inside their pages without requiring any additional programming by the portal developer. To the end-user, it appears that the portlets are running locally within their portal, but in reality the portlets reside in a remotely-running portlets container, and interaction only occurs through the exchange of SOAP messages [13].

2.1.5.1 The Java Specification Requests 168 (JSR 168)

The Java Specification Request, JSR-168 lays a foundation for a new open standard for portal development frameworks. The Java portlet API JSR-168 emerged from the Java Community Process (JCP) principally from the Apache JetSpeed portal project in April 2001[18]. JCP is an open process involving the organization of Java developer institutions with the remit to develop and revise specifications and reference implementations for the Java platform. JSR 168 is designed in order to enable interoperability for various portals within different portal frameworks. It defines

a set of APIs for interacting between the portlets containers and various portlets addressing the areas of presentation, personalization and security [18]. It also seeks to provide a portlet abstraction together with portlet API thus enabling interoperability between portals and portlets [16].

The JSR 168 specification [18] is based on the mature servlets standard following a community review in 2003. The behavior of portlets is similar to that of servlets in many ways, i.e. both portlets and servlets are Java-based web components, managed by a container, used to generate dynamic content and interact with Web clients via a request/response paradigm. Unlike servlets, portlets have additional features and limitations, for example, portlets only generate markup fragments and have pre-defined modes and states, but there are optional extensions allowed. JSR 168 defines a standard API for J2EE-based portal platforms. Its goal is to provide a set of standards so that any compliant portlets can be deployed on any portal which supports the specification [13]. It also handles the presentation end of information enabling reuse of portlets in different containers [9]. The figure 2.2 below depicts a traditional portal model where the portal has a portlet container which hosts a number of discrete portlets. Each of these portlets generates fragments of mark-up which the portal ultimately pieces together to create a complete page that is presented to the user.

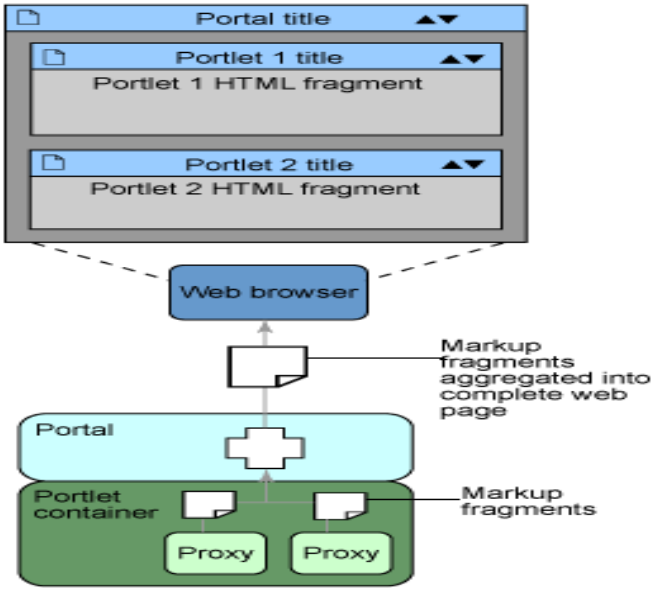


Figure 2.2: **A Portal Aggregating Mark-up from Local Portlets** [13].

For developers, the specification offers *code reusability*, as developers who intend to portal-enable their application only need to create and maintain just one set of JSR-168 compliant portlets. This also implies that portlets can be easily reused if they conform to the same JSR-168 standard. These portlets can run on any JSR 168 portlets specification compliant portal server with few, if any, modifications [18], and because this API is specifically designed for portlets creation, developers will benefit from additional functionalities beyond the standard ones.

The standard also creates a viable market for portal tools: IDEs, performance measurement tools, test tools, etc can be offered to a wider range of users. Amongst the various aspects addresses by this specification are: the portlets containers contract and portlets life cycle management, the definition of window states and portlets modes, portlets preferences management, packaging and deployment, security, etc.

2.1.5.2 The Web Service for Remote Portlets (WSRP)

The Web Service for Remote Portlets, WSRP [13] was formulated to complement the effort of JSR 168. WSRP emerged from the world of Web services which uses Web Service Description Language, WSDL [45] description on how to publish service information after it was adopted by the Organization for the Advancement of Structured Information Standards, OASIS [13] (which also reviewed the JSR-168 standard), a world-wide consortium that drives the development, convergence and adoption of e-Business standards [16].

The ultimate goal of WSRP is to bring web services and the benefits of SOA to the end-user [13]. WSRP is suggested to be a natural tool for SOA systems; providing the missing presentation layer with additional needed features in the existing SOA [12]. It will essentially allow portals to retrieve contents from other portals via their portlet containers and other data sources [9]. The specification defines a common, well-defined interface for communicating with pluggable, presentation-oriented Web services. These services process user interactions and provide mark-up fragments for aggregation by portals [13].

From the above definition, a special look at a couple of the most important terms; first, the services are *presentation-oriented* which means that they provide a user interface that allows an end-user to interact directly with the service. This is starkly different from a traditional perspective of web service which focuses on processing a request and generating a response at a more programmatic

level. Second, the specification defines a *common, well-defined interface* governing how a portal communicates with the service and collects the mark-up fragments it needs to present a page to the end-user. It is precisely this common interface which allows portal applications to generically consume portlets running in remotely-running containers [13].

Portlets are not confined to one portal framework; WSRP defines a standard for interactive, user-facing web services to make portlets hosted by different geographically distributed portal frameworks accessible in a single portal [13]. Unlike traditional web services however, it is a cross-vendor protocol that defines a set of interfaces for enabling portals and non-portal web applications alike to incorporate portlets deployed remotely. It is based on mature extended markup language (XML) and web services specifications that allow the plug-and-play of services in portals and other web applications that aggregate content from disparate sources [46]. WSRP thus enables developers to consume portlets published by other portal sites, irrespective for the varying portal frameworks.
It also allows business level tools to integrate different similar and related portlets services in a dynamic fashion (coupling of services on the fly) by publication and discovery in registries such as the Universal Description, Discovery, and Integration, UDDI [47] using WSDL. Like any other web service, WSRP is also language agnostic although tooling currently only exist in java [12]. It is built upon existing web services standards like SOAP, WSDL, and UDDI. This relationship is briefly depicted in the figure 2.4 below:

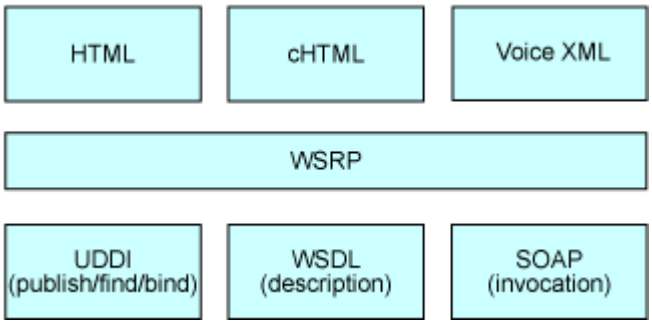


Figure 2.3: **WSRP and Existing Web Service Technologies** [13]

One of the primary benefits of using a portal is that a portal user can customize the set of applications (portlets) that are available to him. However, to customize the portal in such a manner, he must first be aware of what portlets are available. If there is a central registry (or

potentially several registries), portal users can dynamically discover and bind to new portlets, thus creating a work environment tailored to their specific needs.

From the portlet providers' point of view, a centralized registry is equally important since it allows them to publish and describe the portlets which they offer. Providers can provide textual descriptions, categorizations, and other meaningful metadata which richly describe their portlets so that consumers can more effectively discover these services. After all, what is the point of offering portlets to a community if no one knows that they exist?

UDDI provides just such a mechanism to bring together WSRP producers who have portlet services to share and WSRP consumers who are seeking to leverage new applications within their work environment. Since UDDI has become the standard for Web services discovery, it is only natural that it also serves as the backbone for *portlet discovery*. Portlet discovery, however, does throw a few quirks into the mix - a portlet is not a true Web service after all.

As mentioned above, in the WSRP world there are WSRP producers which are true Web services and WSRP portlets which can only be accessed through the API provided by their producer. While a WSRP portlet can be logically thought of as a service, it is not a true Web service since it does not offer an API by which a consumer could invoke it directly.

However, the most common use case when dealing with portlet discovery would involve an end-user looking to find a portlet to add into one of his portal pages. The end-user has no concept of a WSRP producer, nor should he have any reason to understand the underpinnings of WSRP. However, since a portlet can be only be accessed through its producer, both the WSRP portlet and the WSRP producer must be published in the registry.

It should be noted that once a consumer has discovered a portlet service within the registry, the portlets' metadata will contain all of the information necessary for the consumer to directly contact the producer and consume the portlet. Portlet discovery strictly acts as a mechanism to allow producers to describe their portlet services in a central location where potential consumers can discover them.

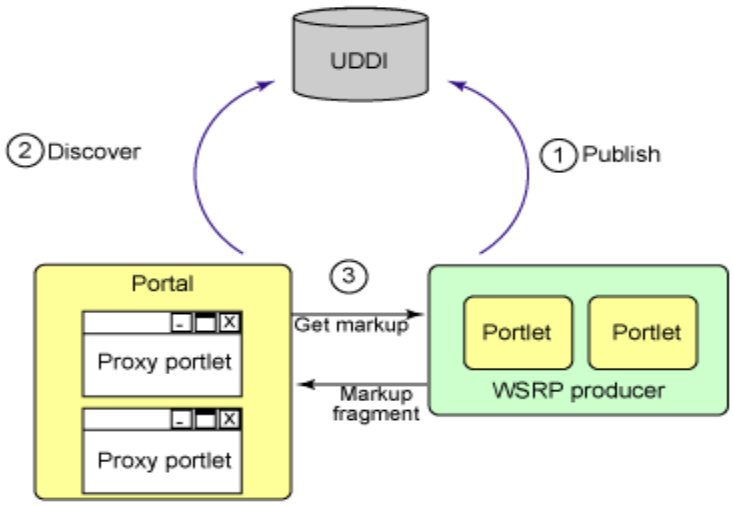


Figure 2.4: **A Typical Publish-Find-Bind Usage Scenario Involving WSRP** [13]

A typical usage scenario normally has the following steps:

1. A provider has developed a set of portlets which have been made available by setting up a WSRP producer and exposing them as WSRP portlets. The provider wants to make these portlets available to the community, so he publishes them to a central UDDI directory. Since UDDI exposes a Web service interface itself, the provider would likely perform this task either through a custom built user interface, UI or through one provided by the UDDI Server.

2. An end-user is looking to add a portlet to his portal. Using the tools provided by his portal (or a custom-written tool specifically for the purpose), he performs a search for portlets. Once the user has found a portlet that he wishes to add to his portal, he easily adds the new portlet application to one of his portal pages. Alternatively, a portal administrator could search the UDDI registry for portlets and make them available to end-users by adding them to the portal's internal registry.

3. When the user now accesses the page to which he added the new portlet, it now contains the remotely-running portlet. Behind the scenes the portal is making a Web service request to the remote producer, and the producer is returning a mark-up fragment for the portal to integrate into the portal page. However, the end-user is completely shielded from the nitty-gritty details of WSRP -- all he knows is that he was able to seamlessly and easily integrate a new application into his portal.

WSRP provides enhanced and extended support for the following crucial requirements of SOA in a portal context [12]: (i) Secured Login; (ii) Single Sign-On, SSO; (iii) Quick Development; (iv) Component Reuse; (v) Loose Coupling; (vi) Ease of Configuration and Use; (vii) Identity Management; (viii) Plug-and-Play Architecture; (ix) Granularity; (x) Flexibility; (xi) User Interaction; (xii) Application Connectivity; (xiii) Information and Process Integration; (xiv) Extensibility; (xv) Statefulness.

Although WSRP is still at an early stage as far as implementation is concerned, it indicates the future of portlet/ portal development [13]. Ideally, a deployed service with a portlet interface can be published and consumed in many different portals frameworks. This remote sharing of a single portlet will greatly ease the construction of large-scale portal based systems, or virtual research environment (VRE), enabling them to be more scalable, manageable and maintainable.

In addition to being published as remote portlets within a normal portal framework which has WSRP producer support, portlets can be published through 3rd-party WSRP producers like WSRP4J [13] which in turn makes use of Pluto [48] as its portlet container. As reported in [19] neither producers nor consumers are however fully functional.

Both JSR-168 and WSRP alongside the Web service technology are aimed at delivering the benefits of SOA to end-users. While Web services offer a mechanism to create platform-independent services and JSR-168 defines a standard by which to develop portlets. And WSRP enables the reuse the entire user interface.

A. The Components of WSRP

The WSRP architecture is made up of different components which are often referred to as the *Primary Actors* within the architecture [46]. Figure 2.5 illustrates each of these primary actors, how they fit with each other and the roles they play within the architecture. Although this figure depicts a WSRP consumer serving only one user or web browser, the consumer can also serve many users. In fact, WSRP consumer has portlet repositories that contain mapping of WSRP portlets offered by respective WSRP producer and personalization information of each WSRP consumer and selected WSRP portlets [46].

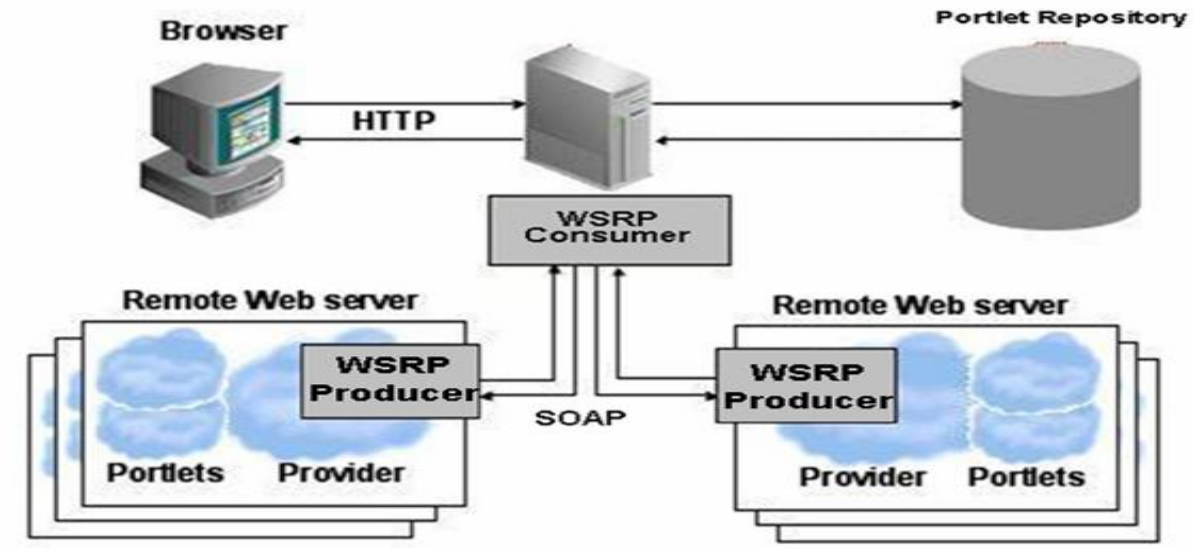


Figure 2.5: **The Components of WSRP Architecture** [46]

In the following figure 2.6 below also, a portal is depicted to consume WSRP portlets from only a single producer. This is no way limit the number of producers to one because a portal can consume portlets from any number of WSRP producers.

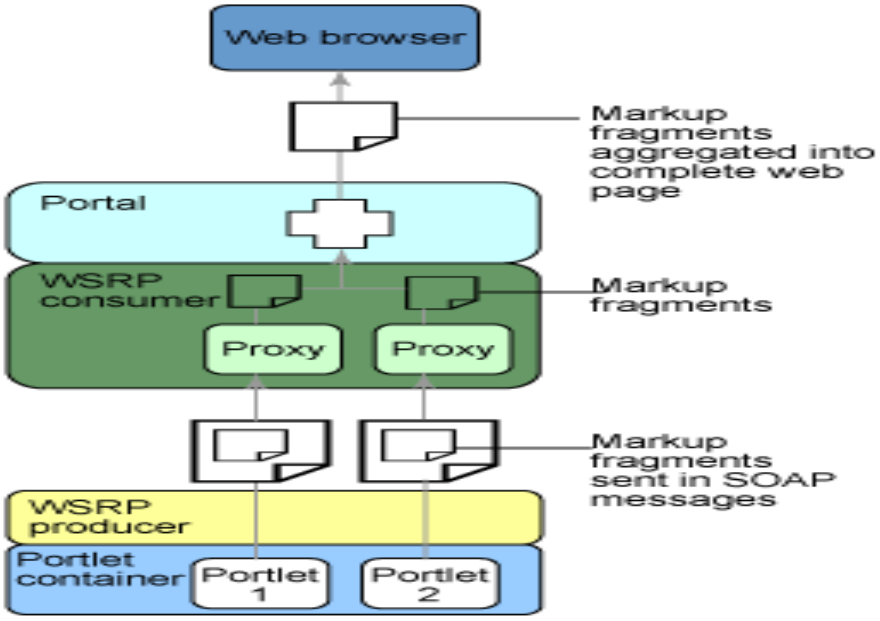


Figure 2.6: **A Portal acting as a WSRP Consumer to Aggregate Mark-up from Remote Portlets** [13]

The WSRP Specification defines the following actors within WSRP architecture [46]:

- **WSRP Producer**: Producers are modeled as containers of portlets. These are web services that offer one or more portlets and implement a set of WSRP interfaces, thus providing a common set of operations such as: self description, mark-up, portlet management, etc. for consumers. Depending on the implementation, a producer could offer just one portlet, or could provide a run-time (or a container) for deploying and managing several portlets. They can optionally manage the registration of consumers and require them to pre-register prior to interacting with portlets. Registration establishes a relationship between the producers and the consumers. The WSRP producer is a true Web service, complete with a WSDL and a set of endpoints. Every producer in WSRP is described using a standardized WSDL document.

- **WSRP Portlets**: A WSRP portlets are pluggable user interface components that live inside of the WSRP producers and are accessed remotely through the interfaces defined by that producers. A WSRP portlets are not web services in their own rights (portlets cannot be accessed directly, but instead must be accessed through their parent producers).

- **WSRP Consumers**: These are web service clients that invoke producer- offered WSRP web services and provide an environment for users to interact with portlets offered by one or more such producers. The most common example of a WSRP consumer is a portal.

In WSRP, the consumer and provider interact via a pre-defined message exchanges independent of the content and context of the problem which is key for a scalable and practical SOA [12].

According to the JSR168 specification [18], a portlet should render different content and perform different activities depending on the current context. Part of this context is the portlet mode. A portlet mode is a way of behaving. For instance, when in the "*view*" mode, the portlet renders fragments which support its functional purpose. Other modes include the "*edit*" mode, where the portlet provides content and logic that let a user customize the behavior of this portlet; the "*help*" mode, where a portlet may provide help screens that explain the portlet purpose, and its expected usage, and, finally, the "*preview*" mode, which serves to pre-visualize the portlet before adding it to a portal page. Other non-standard modes include the "*config*" mode which can be used during configuration to set the appropriate parameter values.

B. The Interfaces of WSRP

In order to standardize communication between the WSRP producers and consumers, WSRP defines a set of common interfaces that all WSRP producers are required to implement and which WSRP consumers must use to interact with remotely-running portlets [13]. Standardizing these interfaces allows a portal to interact with remotely-running portlets generically, since it has a well-defined mechanism for communicating with any WSRP-compliant producer. The WSRP Specification requires that every producer implement two required interfaces, while allowing them to optionally implement two others as well [13]:

- **Service Description Interface (required):** The Service Description Interface allows a WSRP producer to advertise its capabilities to perspective consumers. A WSRP consumer can use this interface to query a producer to discover what portlets the producer offers, as well as additional metadata about the producer itself. This interface can act as a discovery mechanism to determine the set of offered portlets, but also importantly allows consumers to determine additional information about the producer's technical capabilities. The producer's metadata might include information about whether the producer requires registration or cookie initialization before a consumer can interact with any of the portlets.

- **Mark-up Interface (required):** The Markup Interface allows a WSRP consumer to interact with a remotely running portlet on a WSRP producer. For example, a consumer would use this interface to perform some interaction when an end-user submits a form from the portal page. Additionally, a portal might need to simply obtain the latest mark-up based on the current state of the portlet (for example when the user clicks *refresh* or interaction with another portlet on the same page takes place).

- **Registration Interface (optional):** The Registration Interface allows a WSRP producer to require that WSRP consumers perform some sort of registration before they can interact with the service through the Service Description and Mark-up interfaces. Through this mechanism a producer can customize its behavior to a specific type of consumer. For example, a producer might filter the set of offered portlets based on a particular consumer. In addition, the Registration Interface serves as a mechanism to allow the producer and consumer to open a dialogue so that they can exchange information about each others' technical capabilities.

- **Portlet Management Interface (optional):** The Portlet Management Interface gives the WSRP consumer access to the life cycle of the remotely-running portlet. A consumer would have the ability to customize a portlets' behavior or even destroy an instance of a remotely-running portlet using this interface.

As earlier said, portlets are currently employed in Grid-enabled portal design and development as self-contained pluggable user interface components used to encapsulate one or more applications or service components that can be aggregated and transferred as information to a presentation layer of a portal system [14]. These services which are independent of each other somewhat can interact through the portlets if needed. Users of grid services are also enabled to configure services as needed [14].

This new approach to portal development takes its cue from the concept of Service orientation [10]. In the next section an exploration of the service-oriented architecture was done.

2.2 INTRODUCTION TO SERVICE-ORIENTED ARCHITECTURE (SOA)

There has been a lot of buzz and hype (some factual, some not so well-founded) surrounding the opportunities presented by SOA and its implementation as web services. Several predictions by analysts have been made, and various companies have scurried to sell what they had, as SOA products but often misses the point that SOA is not a product rather a more of an architectural style or concept that is about bridging the gap between business enterprise and I.T through a set of business-aligned I.T services using a set of design principles, patterns, and techniques [15].

Service Oriented Architecture is an architectural paradigm and discipline that may be used to build infrastructures enabling those with needs (consumers) and those with capabilities (providers) to interact through services across disparate domains of technology and ownership [15]. It is an approach to designing integration architecture based on concept of service. A Service is a software component that enables access to one or more capabilities with prescribed interfaces [10, 15].

SOA is a software architecture that starts with an interface definition and builds the entire application topology as a topology of interfaces, interface implementations and interface calls [17]. It presents an approach for building distributed systems that deliver application functionality as services to either end-user applications or other services [3]. The key characteristic of these

services is a loosely-coupled, reusable business components; building blocks of SOA application with the intent to provide services to either end user applications or other services through published and heterogeneous network addressable software component [44]. A service in other words can be also described as a function that is self-contained and immune to the context or state of other services [10, 15]. SOA however, describes the relationship between these services and the service consumers [3].

SOA as a conceptual model is also based on an architectural style that defines an interaction model between the three parties: service providers, service consumers, and service broker [15, 17]. The service provider publishes a service description and also provides the implementation for the service. A service consumer can either use the uniform resource identifier (URI) for the service description directly or can find the service description in a service registry and bind and invoke the service [15]. The service broker provides and maintains the service registry. A meta-model showing these relationships is depicted below in figure 2.7. SOA is essentially a collection of self contained, pluggable, loosely coupled services which have well-defined interfaces and functionality with little side effect [10]. These services can communicate with each other either by explicit messages which are descriptive, rather than instructive or there could be a number of "master" services coordinating or aggregating activities, e.g. in a workflow [3, 17]. A service invokes a unit of work done by a service provider to achieve desired end results for a service consumer.

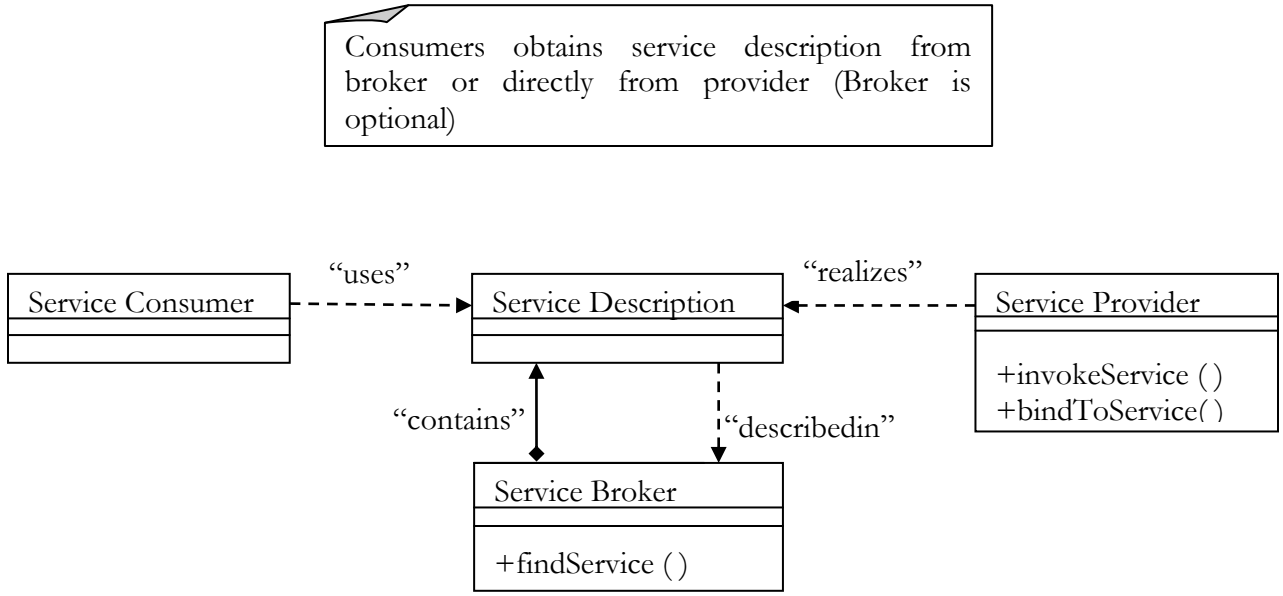


Figure 2.7: **A Conceptual Model of a SOA Architectural Style** [15]

The consumer-provider role is abstract and the precise relationship relies on the context of that specific problem [2]. SOA achieves loose coupling among interacting software agents by employing two architectural constraints: (i) a small set of simple and ubiquitous interfaces to all participating software agents; (ii) the interfaces should be universally available for all providers and consumers [10].

Services are software modules that are accessed by name via an interface, typically in a request-reply mode [2, 3]. It is a software resource (discoverable) with an externalized service description. It is also described as a unit of work such as a business function, a business transaction, or a system service completed by a service provider to achieve desired end results for a service consumer [2]. Service consumers are software that embeds a service interface proxy (the client representation of the interface). The service provider realizes the service description implementation, and also delivers the quality of service requirements to the service consumer. The SOA concept separates the service's implementation from its interface [17]. Service consumers view a service simply as an endpoint that supports a particular request format or contract [3, 17]. Service consumers are not concerned with how the service goes about executing their requests; they expect only that it will undoubtedly produce an answer.

Consumers also expect that their interaction with the service will follow a contract, an agreed-upon interaction between two parties [10, 17]. The way the service executes tasks given to it by service consumers is irrelevant. The service might fulfill the request by executing *servlets*, a mainframe application, a C# or a Visual Basic application. The only requirement is that the service sends the response back to the consumer in the agreed-upon format [10].

2.2.1 THE CHARACTERISTICS OF SOA

The concept of services in software engineering has been in existence long before the advent of service-oriented architecture. However, service-oriented software architecture like every other software architecture reflects principles that make it suitable for implementing distributed functionalities as services. The following are SOA characteristics [17, 49]:

1. **Discoverable and Dynamically Bound**: SOA supports the concept of service discovery. A service consumer that needs a service discovers what service to use based on a set of criteria at runtime. The service consumer asks a registry for a service that fulfills its needs.
2. **Self-Contained and Modular**: Services are self-contained and modular. A service supports a set of interfaces. These interfaces should be cohesive, meaning that they should all relate to each other in the context of a module. The principles of modularity should be adhered to in designing the services that support an application so that services can easily be aggregated into an application with a few well-known dependencies.
3. **Modular Decomposability**: The *modular decomposability* of a service refers to the breaking of an application into many smaller modules where each module is performing distinct function within an application. This is sometimes referred to as "top-down design," in which the bigger problems are iteratively decomposed into smaller problems. The crust of this is to achieve reusability. The goal for service design is to identify the smallest unit of software that can be reused in different contexts.
4. **Modular Composability**: The *modular composability* of a service refers to the production of software services that may be freely combined as a whole with other services to produce new systems. Service designers should create services sufficiently independent to reuse in entirely different applications from the ones for which they were originally intended. This is sometimes referred to as bottom-up design.
5. ***Modular Understandability**:* The modular understandability of a service is the ability of a person to understand the function of the service without having any knowledge of other services.
6. **Modular Continuity:** The *modular continuity* of a service refers to the impact of a change in one service requiring a change in other services or in the consumers of the service. This is as a result of an interface not sufficiently hiding its implementation details. It will require changes to other services and applications that use the service when the internal implementation of the service changes. Every service must hide information about its internal design. A service that exposes this information will limit its modular continuity, by exposing internal design decision through the interface.
7. **Modular Protection:** The *modular protection* of a service is sufficient if an abnormal condition in the service does not cascade to other services or consumers. Faults in the operation of a service must not impact the operation of a client or other service or the state of their internal data or

otherwise break the contract with service consumers. Therefore, we must ensure that faults do not cascade from the service to other services or consumers.

8. **Direct Mapping:** A service should map to a distinct problem domain function. This is to allow service designers create a self-contained and independent module.
9. **Conceptual Service Model:** The conceptual service model consists of a model of the problem domain. Techniques for defining module interfaces assume that the problem domain is known a priori. The conceptual model of the business is simply the *business architecture.* A conceptual model is one created without regard for any application or technology. It typically consists of a structural model derived from a set of use cases that illustrate how the business works.
10. **Contracts and Information Hiding:** An interface contract is a published agreement between a service provider and a service consumer. The contract specifies the arguments the service requires to be invoked, the return values a service supplies and the service's pre-conditions and post-conditions. The pre-conditions are those that must be satisfied before calling the service, to allow the service to function properly.
11. **Interoperability:** Service-oriented architecture stresses interoperability: the ability of systems using different platforms and languages to communicate with each other. Each service provides an interface that can be invoked through a connector type. An interoperable connector consists of a protocol and a data format that each of the *potential* clients of the service understands. Interoperability is achieved by supporting the protocol and data formats of the service's current and potential clients.
12. **Loose Coupling:** *Coupling* refers to the number of dependencies between modules. There are two types of coupling: loose and tight. Loosely coupled modules have a few well known dependencies. A system's degree of coupling directly affects its modifiability. The more tightly-coupled a system is, the more a change in a service will require changes in service consumers. Coupling is increased when service consumers require a large amount of information about the service provider to use the service. In other words, if a service consumer knows the location and detailed data format for a service provider, the consumer and provider are more tightly coupled. If the consumer of the service does not need detailed knowledge of the service before invoking it, the consumer and provider are more loosely coupled.

    SOA accomplishes loose coupling through the use of contracts and bindings. A consumer asks a third-party registry for information about the type of service it wishes to use. The registry returns all the services it has available that match the consumer's criteria. The consumer chooses which

service to use, binds to it over a transport, and executes the method on it, based on the description of the service provided by the registry. The consumer does not depend directly on the service's implementation but only on the contract the service supports.

Although coupling between service consumers and service producers is loose, implementation of the service can be tightly coupled with implementation of other services. For instance, if a set of services shares a framework, a database, or otherwise has information about each other's implementation, they may be tightly coupled.

13. **Network-Addressable Interface:** A service must have a network-addressable interface. A consumer on a network must be able to invoke a service across the network. The network allows services to be reused by any consumer at any time. The ability of an application to assemble a set of reusable services on different machines is possible only if the services support a network interface. The network also allows the service to be location–independent, meaning that its physical location is irrelevant.
14. **Coarse-Grained Interfaces:** The concept of granularity applies to the scope of the domain the entire service implements and also the scope of the domain that each method with the interface implements. If a service implements all the functions in its domain, it is referred to as coarse grained, but if it implements just a function in its domain, we consider it as fine grained. The appropriate level of granularity for a service and its methods is relatively coarse. A service generally supports a single distinct business concept or process. It contains software that implements the business concept so that it can be reused in multiple large, distributed systems.

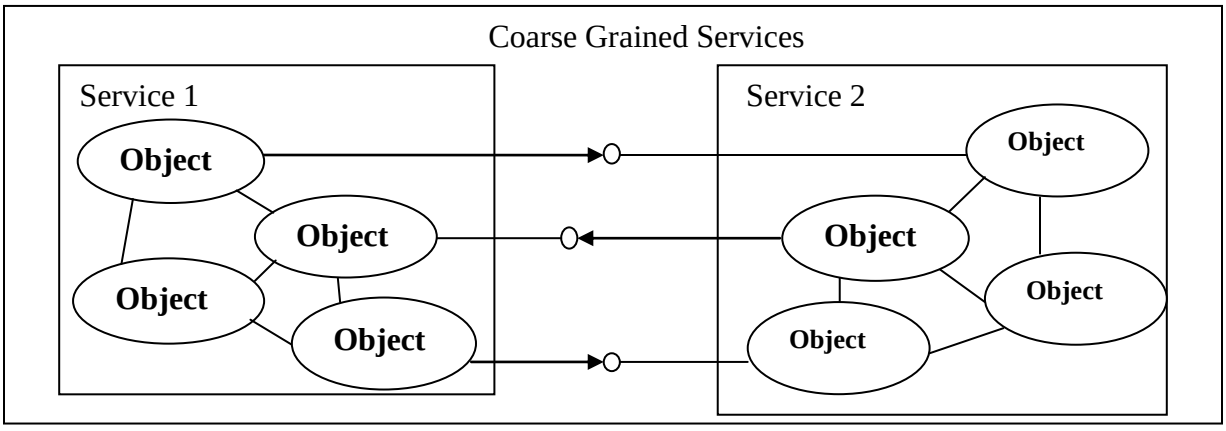


Figure 2.8: **Coarse Grained Services** [17]

15. **Location Transparency:** Consumers of a service do not know a service's location until they locate it in the registry. The lookup and dynamic binding to a service at runtime allows the service implementation to move from location to location without the client's knowledge. The ability to move services improves service availability and performance.
16. **Composability:** A service may be composed in three ways: application composition, service federations, and service orchestration. An application composition is essentially an assembly of services, components, and application logic that binds these functions together for a specific purpose. Service federations are collections of services managed together in a larger service domain. Service orchestration is the execution of a single transaction that impacts one or more services in an organization. It is sometimes called a business process. It consists of multiple steps, each of which is a service invocation. If any of the service invocations fails, the entire transaction should be rolled back to the state that existed before execution of the transaction.

    For a service to be composed into a transactional application, federation, or orchestration, the service methods themselves should be *sub-transactional*. That is, they must not perform data commits themselves. The orchestration of the transaction is performed by a third-party entity that manages all the steps. It detects when a service method fails and asks all the services that have already executed to roll-back to the state they existed before the request. If the services have already committed the state of their data, it is more difficult for the method to be composed into a larger transactional context.
17. **Self-Healing:** A *self-healing* system is one that has the ability to recover from errors without human intervention during execution. *Reliability* measures how well a system performs in the presence of disturbances. Reliability depends on the hardware's ability to recover from failure.

    Service-based systems require that the interface be separate from the implementation, implementations may vary. For instance, a service implementation may run in a clustered environment. If a single service implementation fails, another instance can complete the transaction for the client without the client's knowledge. This capability is possible only if the client interacts with the services interface and not its implementation.

2.2.2 The Requirements for SOA

SOA should be developed to meet the following requirements in one way or the other to address problems and issues that led to the concept of SOA [10, 12]:

1. Leveraging existing assets is the most important requirements here. Strategically, the objectives is to build a new architecture that will tactically integrate existing systems, such that over a period, they can be componentized or replaced in manageable, incremental projects;
2. Support all required types or "styles" of integration such as:
   - User Interaction – being able to provide a single interactive user experience achievable through portals and portlets,
   - Application Connectivity – communication layer i.e. middleware that underlies all of the architecture,
   - Process Integration – choreographs applications and services through the process model called "workflow",
   - Data Integration – incorporate data flow within aggregated grid service,
   - Portal Integration – provides presentation layer to diverse resources to access them through single location;
3. Architecture should allow incremental implementations and migration of assets. Due to project complexity, cost and unworkable implementation schedules, many integration projects have failed;
4. Developments environment built around standard components framework such as portal/portlets specifications, WSRP, Web Services; promote better reuse of modules and systems and allows timely implementation of new technologies;
5. Allow implementation of new computing models; specifically new portal-based client models, grid computing, and even on-demand computing.

2.2.3 The Collaboration between SOA Entities

SOA consist of three entities: *Service provider, Service Consumer* and *Service Registry* [17, 49]. The collaboration between these three entities follows the "find, bind, and execute" paradigm as shown in Figure 2.3, allows the consumer of a service to ask a third-party registry for the service which it is intending to bind to. If the registry has such a service, it gives the consumer a contract and an endpoint address for the service.

1. **Service Consumer**

   The service consumer is an application, service, or some other type of software module that requires a service. It is the entity that initiates the location of the service in the registry, binding to the service over a transport and executing the service function. The service consumer executes the service by sending it a request formatted according to the contract.

2. **Service Provider:** The service provider is a network-addressable entity that accepts and executes requests from consumers. It can be a mainframe system, a component, or some other types of software system that executes the service request. The service provider publishes its contract in the registry for access by service consumers.

3. **Service Registry:** A service registry is a network-based directory that contains available services. It is an entity that accepts and stores contracts from service providers and provides those contracts to interested service consumers.

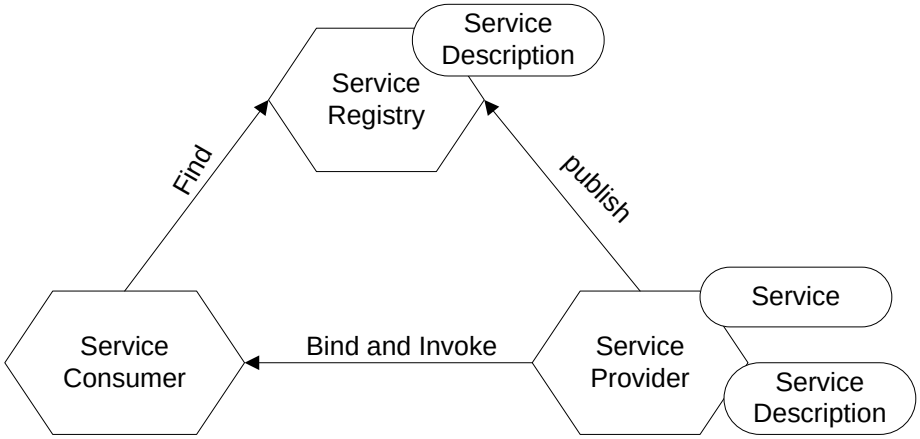


Figure 2.9: **The Collaboration in Service Oriented Architecture** [17]

The operations in a SOA are:

- **Publish:** for a service to be accessible, the service provider publishes the service description so that it can be discovered and invoked by service consumers.
- **Find:** service requester locates a service by querying the service registry for a service that meets its criteria.
- **Bind and invoke:** after successfully retrieval the service description, the service consumer then invokes the service based on the information provided by the service description.

4. **Service Contract:** A contract specifies the way a consumer of a service will interact with the provider of the service. It specifies the format of the request and response from the service. A service contract may require a set of preconditions and post-conditions. The preconditions and post-conditions specify the state that the service must be in to execute a particular function. The contract may also specify quality of service (QoS) levels. QoS levels are specifications for the non-functional aspects of the service [3]. For instance, a quality of service attribute is the amount of time it takes to execute a service method.

2.2.4 Service Provider and Service Consumer Relationship

In a SOA there are certain relationships that exist between the service providers and consumers [49]. These are briefly stated below:

1. **Negotiated** - both consumer and provider jointly agree to how the services should work. In scenarios where there are many participants and where services are common to many providers, it is important that the industry considers standardizing those services [49]. This include:
   - Close partners agreeing on the service interface as a natural part of reaching and implementing a commercial agreement
   - Forming standard for vertical sectors in the industry.
2. **Mandated** - this is a take-it or leave-it scenario [17]. Very large or dominant organization(s) dictate the business practice in their industry. Examples include:
   - *Provider-led situations* – such as Ford Motors "use this service or we can't do business".
   - *Consumer-led situations* – such as Wal-Mart and Tesco [50].

2.2.5 SOA Architectural Style and Principles

The architectural style that defines a SOA describes a set of patterns and guidelines for creating loosely coupled, business-aligned services that, because of the separation of concerns between description, implementation, and binding, provide unprecedented flexibility in responsiveness to new business threats and opportunities [15].

As enterprise-scale I.T architectures, SOA is used for linking resources on demand. Participants in a line of business, enterprise (typically spanning multiple applications within an enterprise or across multiple enterprises) can access available resources in a SOA. It consists of a set of business-aligned I.T services that collectively fulfills an organization's business processes and

goals. These services can be choreographed into composite applications and then invoked through some standard protocols. This is depicted in the figure 2.10 below.

SOA provides the emerging trend for organisations' transformation program. This is to make information resources substiantially independent, reusable, and to create an adaptable environment [17]. Business and technical services are published using open standard protocols that create self-describing services that can be used independently of underlying technology [17].
Technical independence allows sevices to be more easily used in different contexts to achieve standardisation of business processes, rules and policies. Collaboration, internal and external to an enterprise, can more easily be established through improvement in process and information consistency [17, 49].

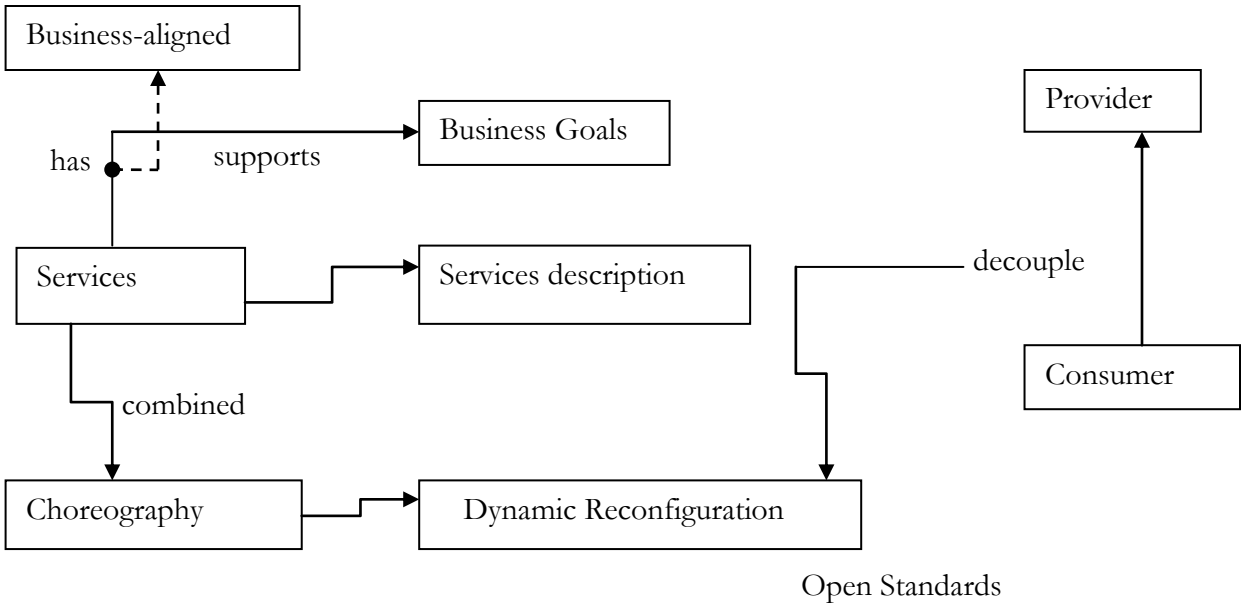


Figure 2.10: **The Attributes of SOA** [15]

Most significant SOAs are proprietary or customized implementations based on reliable messaging and Enterprise Application Integration middleware (for example WebSphere Business Integration, WBI).

2.2.6 SOA Implementation Models

SOA is an architectural style that presents an approach for building distributed systems that deliver application functionality as services to either end-user applications or other services [17, 49]. Early adopters of the service-oriented architecture approach used messaging systems to create service-

oriented enterprise architecture. Examples of these include IBM WebSphere MQ [51]. Currently, the SOA arena has expanded to include the World Wide Web (WWW), Web Services (WS) and Enterprise Service Bus (ESB) [52, 53].

An ESB is an architectural practice for implementing a service-oriented architecture [53]. As shown in Figure 2.11 below, it establishes an enterprise-class messaging bus that combines messaging infrastructure with message transformation and content-based routing in a layer of integration logic between service consumers and providers.

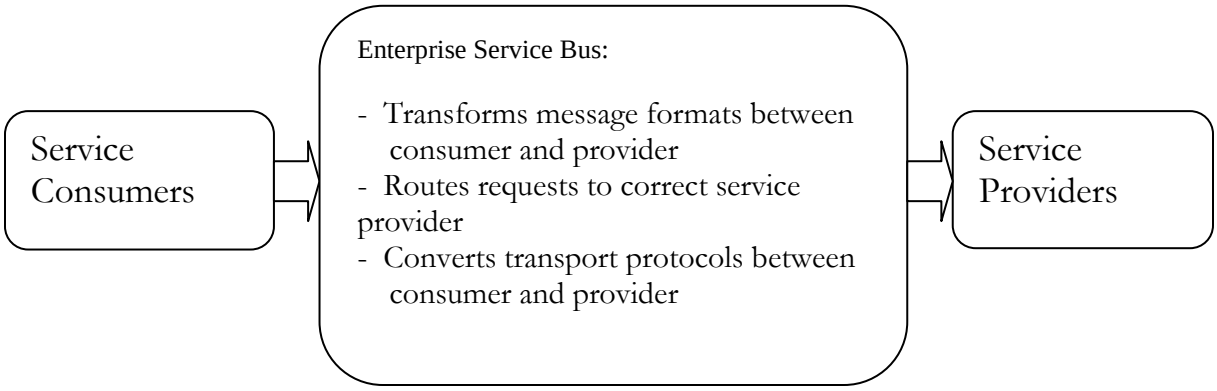


Figure 2.11: **The Enterprise Service Bus** [53]

The ESB incorporates a standards-based, enterprise-class messaging backbone, together with enhanced systems connectivity using Web services, Java 2 Enterprise Edition (J2EE), Microsoft .NET, and other standards. In essence, ESB makes large-scale implementation SOA principles manageable in the heterogeneous world [52, 53].

The ESB helps to provide virtualization of the enterprise resources, by allowing the business logic of the enterprise to be developed and managed independently of the infrastructure, network, and provision of those business services [52]. Using ESB, one can link individual enterprises together for extended process efficiency across the supply chain and allow them to become more flexible and adaptable to rapidly changing requirements. The ESB lets an enterprise leverage its previous investments by supporting the deployment of processes over existing software and hardware infrastructure [52]. ESB supported standards include:

- Java Message Service (JMS) for communication;
- Web services, J2EE, and .NET for connectivity to various systems;
- Extensible Stylesheet Language Transformation (XSLT) and Xquery for transformation;

- Lightweight Directory Access Protocol (LDAP), Secure Sockets Layer (SSL), and others for security.

Implementing an Enterprise Service Bus requires an integrated set of middleware services that support the following architectural styles [52]:

- *Services-Oriented Architecture:* where distributed applications are composed of granular re-usable services with well-defined, published and standards-compliant interfaces.
- *Message-Driven Architectures* (MDA)*:* where applications send messages through the ESB to receiving applications
- *Event-Driven Architectures* (EDA)*:* where applications generate and consume messages independently of one another.

Other technologies that are at partly service-oriented and have been widely used in achieving interoperability include: Common Object Request Broker Architecture (CORBA) [54], Remote Method Invocation (RMI) [55], and Distributed Component Object Model (DCOM) [56].

2.2.7 Web Services

The implementation of SOA applications is made possible through the realization of Web Services, WS [48]. According to W3C [57], "A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols". It is a software system designed to support interoperable machine-to-machine interaction over a network [58]. It is also described as a software component representing specific set of business functions that can be described, published and invoked over the Internet using XML-based open standards such as SOAP [43], WSDL [45] and UDDI [47]. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols [57]. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [58].

Web service is a useful tool in enabling collaboration and sharing of business process between two or more enterprises. It offers technology neutrality and standard approach than using proprietary

integration technologies [57]. Web services promises to offer enterprise application the capability that World Wide Web did to interactive end-user application. Primarily, web service is a technique that allows disparate server systems to communicate with each other and exchange information for which the web and traditional web browser is the primary data access point [57]. Web service is a good beginning toward implementing service-oriented architecture because its concept supports many of the characteristics of service-oriented architecture [17, 49].

2.2.8 Web Service Architecture

According to [59] Web services architecture describes the relationship among various components and technology that comprises web services "stack". A valid implementation must consist of at least the components in the basic architecture. The basic architecture includes web services technology that allows:

- Exchange messages;
- Describing web services; and
- Publishing and discovering Web service descriptions as depicted in figure 2.12 below.

The Web Service architecture models the interactions between three roles: the *service provider*, *service discovery agency*, and *service requester*. The interactions involve *publish*, *find*, and *bind* operations. In a typical scenario, a service provider hosts a network accessible software module (an implementation of a web service). The service provider defines a service description for the web service and publishes it to a requester or service discovery agency. The service requester uses a find operation to retrieve the service description locally or from the discovery agency (i.e. a registry or repository) and uses the service description to bind with the service provider and invoke or interact with the web service implementation. Service provider and service requester roles are logical constructs and a service may exhibit characteristics of both.

The architecture also defines an interaction between software agents as an exchange of messages between service requesters and service providers. A software agent in the web services architecture can act in one or multiple roles, acting as requester or provider only, as both requester and provider, or as requester, provider, and discovery agency. Requesters are software agents that request the execution of a service. Providers are software agents that provide a service. A service is invoked after the description is found, since the service description is required to establish a binding.

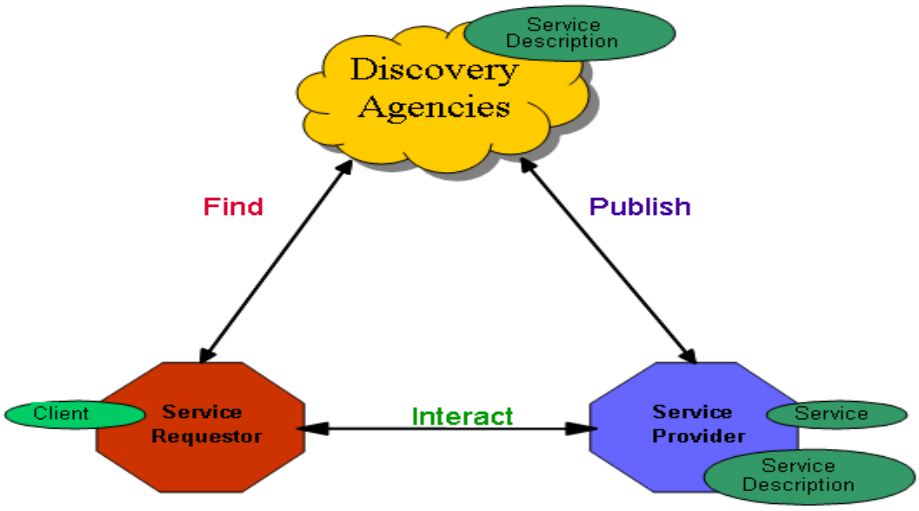


Figure 2.12: **Web Service Architecture** [59]

Figure 2.12 illustrates the basic Web service architecture, in which a service requester and service provider interact, based on the service's description information published by the provider and discovered by the requester through some form of discovery agency.

2.2.9 Web Service Technology

Within the framework of web service are a number of various technologies that underline the web service execution. This is depicted in figure 2.13 below.

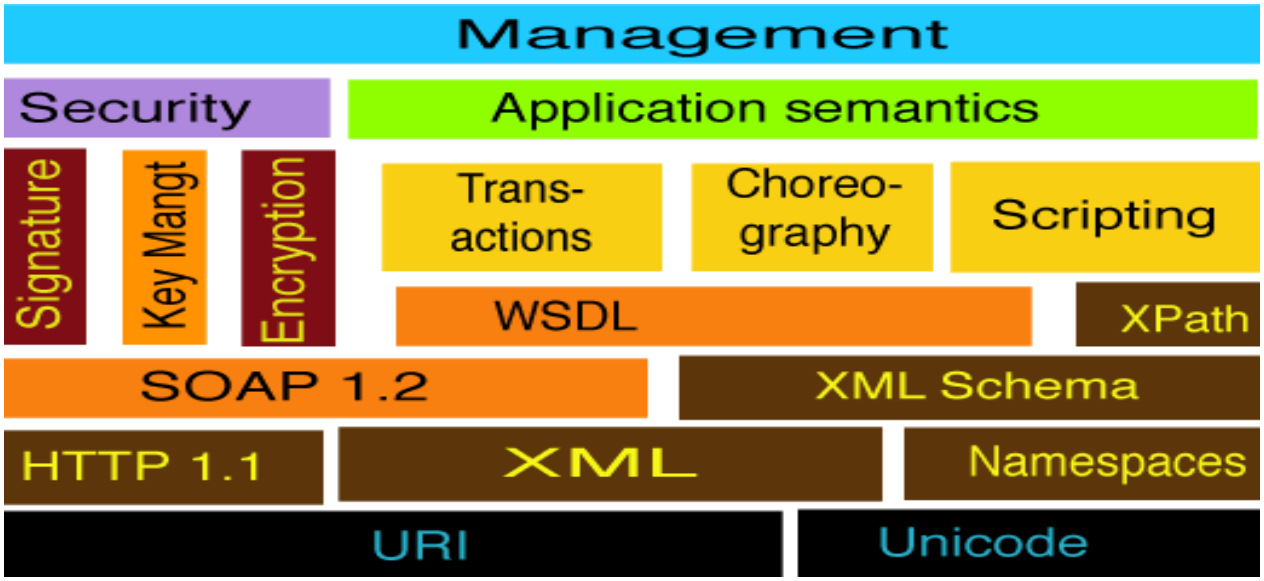


Figure: 2.13: **Technologies within the Web Service Technology** [58].

- The *Management Layer* is a supervisory layer allowing the control of the many agents involved in a web services-based operation.

- The *Application Semantics* layer indicates the necessity for any useful interoperability.

1. **Web Services Description Language (WSDL)**

Web Services are defined by a Web Services Definition Language, WSDL. A WSDL is an XML format to describe how a particular web service can be called. WSDL description specifies how to interact with the web service, what data must be sent, what operations are involved, what protocol is to be used to invoke the service, and what data can be expected in return. The WSDL provides an Interface Definition Language (IDL) that is used to describe the operations (method calls) that a web service may invoke as well as protocol bindings used to transport method invocations.

A WSDL document uses the following elements in the definition of network services [15, 59]:

- **Types -** a container for data type definitions using some type system.
- **Message** - an abstract, typed definition of the data being communicated.
- **Operation** - an abstract description of an action supported by the service.
- **Port Type -** an abstract set of operations supported by one or more endpoints.
- **Binding** - a concrete protocol and data format specification for a particular port type.
- **Port** - a single endpoint defined as a combination of a binding and a network address.
- **Service** - a collection of related endpoints.

2. **Simple Object Access Protocol (SOAP)**

SOAP is an open Internet standard for achieving message exchange amongst interactive agents. It is used for the invocation of web services and consists of a messaging layer described by XML over a transport protocol, often HTTP, although any protocol may be used e.g. FTP, JMS.

SOAP is designed with three goals in mind [43]:

- It should be optimized to run on the Internet.
- It should be simple and easy to implement.
- It should be based on XML.

It supports two types of message patterns: the first is the one-way exchange, where a client issues a request against a server, and will not receive an answer. The second is the pattern which consists of

request response interaction. Here, the client use HTTP request for a resource on a server, and the server replies by sending a HTTP response.

3. **Universal Description, Discovery, and Integration (UDDI)**

The UDDI specification provides a framework for describing and discovering web services. It supports application developers in finding information about web services so that they know how to write clients applications that can interact with those services. It also enables dynamic binding by allowing clients to query the registry and obtain references to services in which they are interested. The information within a UDDI registry can be categorized as follows [47]:

- Listings of organizations, contact information, and services that those organizations provide;
- Classifications of companies and web services according to taxonomies that are either standardized or user defined;
- Descriptions of how to invoke web services, by means of pointers to service description documents, stored outside the registry, for example, at a service provider's site.

A UDDI registry contains web services descriptions with four different kinds of information elements described as follows:

- **Business Entity:** An organization that provides web services, including the company's name, address, and other contact information.
- **Business Service:** A group of related web services offered by a business entity. Typically, it corresponds to one kind of service (such as a procurement or reservation service).
- **Binding Template:** Technical information needed to use a web service, such as the address at which the web service can be found and references to documents (called tModels) that describe the web service interface and other service properties. It also defines how operation parameters should be set and what the default values are.
- **tModel:** Technical model, which is a container for any kind of specification. For example, it might represent a WSDL service interface, a classification, or an interaction protocol, or it might provide the semantics of an operation.

2.2.10 Web Service Characteristics and Best Practices

The realization of SOA is centered on Web Services (WS) [44]. It is important to understand fully the characteristics of Web Services, in terms of the dos and don'ts for WS, which form the basis of the best practices for Web Services development. These characteristics affect the design and

implementation of Web Services. The following sub-sections discuss the characteristics of Web Services and its associated best practices [44].

- **Web Services Styles (WSBP1):** There are two most common styles of Web Services, namely *Remote Procedure Call* (RPC) style WS and *Document Style* WS. The differences between these two styles are summarized in Table 2.1 below.

	RPC-Styled	Document-Styled
Processing Mode	Business object-centric	Document-centric
Interaction	Request and Response (Synchronous)	Fire and forget (Asynchronous)
Implementation	Java, EJB	JMS

Table 2.1: **Web Services Styles** [44]

The RPC-styled offers simplicity and better tooling support. The document-styled offers greater flexibility and decoupling of services [44].

- **Web Services Interaction Mode (WSBP2):** Web Services have four interaction modes [44]. They are: *synchronous* interaction (i.e. request and wait for response), *asynchronous* interaction (i.e. fire and forget), *solicit-response* interaction (i.e. the service sends a message followed by a correlated message from client), and *notification* interaction (i.e. the service sends a message). Any one of this mode will affect the way of designing and implementation Web Services.
- **Web Services Client Implementation-Interaction Mode (**WSBP3**):** The client implementation will be determined by Web Services Interaction modes. If it is an asynchronous WS, an asynchronous WS client implemented using Java API for XML Messaging JAXM, for example, will be used. Otherwise, Java API for XML for Remote Procedure Call (JAX-RPC) will be used.

- **Web Services Client Implementation – Client Types (WSBP4):**

  The client implementation is affected by the types of Web Services client. Particularly in Java–based RPC service, there are three different types of Web Services client, namely *static stub*, *dynamic proxy* and *dynamic invocation interface* (DII) of web service clients for consuming a service. The three types of client offer different degree of client flexibility. For example, static stub is the least flexible as any changes to the service would require rebuilding of service client. DII is the most flexible as the client parses the service's WSDL in constructing a SOAP message for service invocation. Any change to the end point service does not require rebuilding of client.

- **Right Level of Service Interface Granularity (WSBP5):** The granularity of the service interface affects the design and implementation of web service. In addition, it also affects the performance of the service. The finer the granularity for service interface, the slower the performance as it is an overhead to the network and drop in web service performance.

- **Interoperability (WSBP6):** Interoperability issues could be caused by different versions of SOAP standard implementation, different types of security algorithms for digital signature, encryption/decryption, and variation in supporting Web Services standards from multiple vendors. The adoption of primitive data type for parameters whenever possible. Avoid using customized SOAP serializer/deserializer and different types of encoding standards.

- **Binding Style (WSBP7):** The use of *RPC/encoded or Document/literal* binding style is determined by the needs of data information being exchange between Web service client and the service. If it is data intensive or the exchanged information is a file, then document/literal binding is preferred. If the data information exchanged is relatively static, then *RPC/encoded* binding is preferred.

- **Request and Response Performance (WSBP8)**: Web Services itself is network intensive. It demands extra network bandwidth and CPU processing time and memory due to the needs for SOAP message serialization and de-serialization overhead. The common practices for optimizing the request and response performance are: (a) perform data caching in either client or server side, (b) decide Web Services operation granularity, (c) Use XML judiciously in document-centric Web Services by careful considering whether to use whole or segment of XML document.

- **Security (WSBP9)**: There are various ways to secure the information sent between initial SOAP sender and ultimate SOAP receiver via numerous intermediary SOAP nodes. Different means of security can affect the way how Web Services are designed and implemented. The security means can be through:
  a.) *Transport Level Security* (TLS). In this mean, it leverages on transport security mechanism. Only the initial SOAP sender and ultimate SOAP receiver are secured. Intermediary nodes are not secured. The two most common means are secure socket layer (SSL) or HTTPS.
  b.) *Message Level Security* (MLS). In this mean, message can be secured throughout the whole SOAP message path. Standards such as XML Encryption, XML Signature, XML Key Management, WS-Security, etc. can be applied to secure the XML message.
  c.) *Infrastructural Level Security*: In this mean, it leverages on the security mechanism provided by Web Services hosting platform.
- **Web Services Implementation Technology & Platform (WSBP10):** What is the technology platform, such as J2EE or .NET based, to be used? What kind of application server is required to host services? The understanding these lead to better services interoperability.
- **Industry Standard Conformance (WSBP11)**: The conformance of industry standard, such as RosettaNet™ provided by the service determines the type of services. As it gives rise to the consideration of the requirement for well-formed XML document and document-styled Web service for the service.
- **Addressable Software Component (WSBP12):** Every end point service is identifiable using universal resource locator (URL). To know whether service is available, an invocation test to the service URL would provide the availability status of the service.
- **Web Services Needs (WSBP13):** Web Services Technology is applied to meet certain business needs and objectives. The considering factors include reuse business components, integrate different IT platforms and disparate islands of technologies, direct business-to-business integration (B2Bi) between partners to facilitate information sharing. Understanding the basic needs would ascertain better drive of Web Services Technology to be applied appropriately.
- **Web Services Layering Architecture (WSBP14)**: The consideration for hierarchical abstraction for Web Services enables the decoupling relationship for services. This facilitates

layered hierarchy representing ordered grouping of functionality abstraction for domain-specific application (upper layer), across domains application (middle layer), and deployment environment-specific application (lower layer).

An Understanding of the best practices of Web Services helps in addressing SOA design and implementation issues. However, the best practices for Web services (i.e. the dos and don'ts of Web services) are essentially based on the characteristics of Web Services listed above.

2.3 COMPONENT BASED DEVELOPMENT

SOA application development involves developing software components for software reuse and wrapping software components as Web services for end user applications or other services consumptions [44]. The nature of these applications is centered on software components [44, 60].
A *component* is popularly defined as *"each reusable binary piece of code"* [60]; it is an independent part of a system having complete functionalities. It is also described as a reusable software building block: a pre-built piece of encapsulated application code that can be combined with other components and with additional code to produce custom application [61]. Just like Patterns, a component drives the developers to use the predefined procedures and meet the specifications to plug it into the new components [60].

A *software component* therefore, is a unit of composition with contractually specified or defined interfaces and conforms to a prescribed behavior [44, 60]. It could also be referred to as an independently deliverable package of reusable software services. Software components are the reusable building blocks of SOA application [44]. The need for components arose as a result of inherent problems identified with Object-Oriented Development (OOD) [60]. In OOD, objects appeared are too complicated and provides limited functionality to be useful to many clients without giving room for plug-and-play, while components such as plug-ins provide a high-level feature that can be installed and configured by the users [60].

In SOA, software components are encapsulated as Services [10]. Therefore, SOA applications can be developed using one of the evolutionary software development approaches known as Component-Based Development (CBD). CBD enables development of software application by

assembling and use of existing components. It is a technology that facilitates the *reuse* of the existing components into new ones [60]. These components can be acquired by leveraging legacy systems, as commercial-of-the-shelf (COTS) systems and some others are basically open source, from a third party developers or vendors, developing components in order to enable reusability. SOA however, provides a mechanism for integrating existing legacy applications regardless of their platform or language [10, 17]. This facilitates shorter time to market, reduced cost, and increased reuse [17]

.

CBD is a software development approach where the entire lifecycle of the software creation, development deployment and maintenance is centered on the *start-to-finish* concept of component lifecycle [44, 60]. The CBD process has five phases, namely *Requirements*, *Analysis*, *Design*, *Implementation* and *Testing* [60]. Artifact is produced at each phase which in turn is the inputs to different types of testing shown in Figure 2.14 below. Each component has its own lifecycle and it is related with the whole system lifecycle. Agile software development can be applied in component-based software development [62] and any of the agile software development methodologies such as extreme programming (XP) [63], IBM Rational Unified Process (RUP) [64], etc can also be applied to component-based software development.

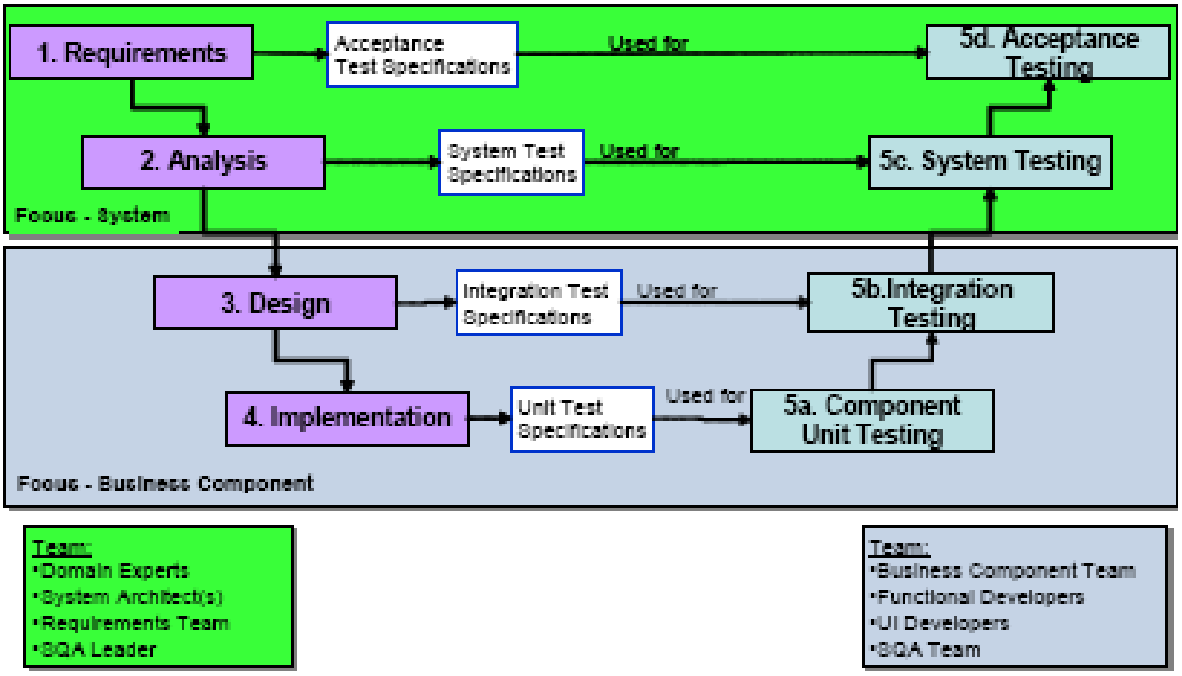


Figure 2.14: **An Illustration of CBD Process** [44]

Furthermore, there also exist some relationships between software component, Web services and SOA application. This is illustrated in the figure 2.15 below. This shows that the development of Web services is based on software component through public interfaces exposed for services consumption [44]. For instance, in the figure 2.15, *Order Analyzer* and *Order Generator* are software components derived from objects or classes. The *Order Processor* is a web service that uses components *Order Analyzer* and *Order Generator* to provide richer business functionality as building blocks for SOA application.

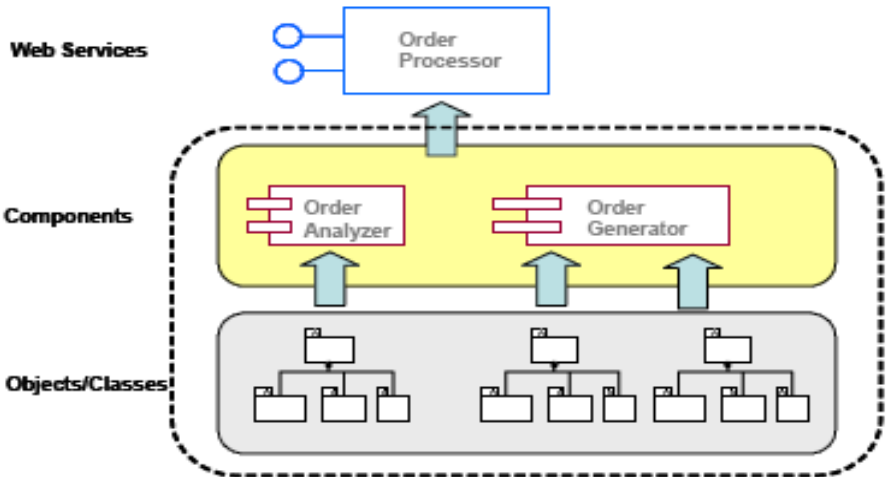


Figure 2.15: **Web Services CBD Development** [44]

Over the years, several component architectures have been proposed up to now, for CBD. Microsoft Corporation introduced the ActiveX technology which is categorized into the Component Object Model (COM), Distributed Component Object Model (DCOM) and Object Linking and embedding (OLE). Sun Microsystems also introduced the Enterprise Java Beans (EJB) [19] and Remote Method Invocation (RMI), Object Management Group (OMG) introduced Common Object Request Broker Architecture (CORBA); Microsoft, IBM and Lotus corporations introduced Simple Object Access Protocol (SOAP) [43]. Each of these architectures has been adopted over time to further facilitate application development through CBD. CBD also shifts the development emphasis from "*programming software"* to "*composing software"* [44, 60].

2.3.1 Process Model of Component Based Development

The Object Oriented Process model (OOPM) is the only process model that indicates the reuse of existing software parts until the advent of service orientation [65].

This process model however, can be modified to implement the reuse of component-based development. The main phases of process model are: *Customer Communication*; *Planning*; *Analysis*; *Engineering, Construction and Testing*; and *Customer Evaluation*. The engineering, construction and testing phase reflect the reuse of existing classes. The main phases of CBD process model are shown, in figure 2.18 below.

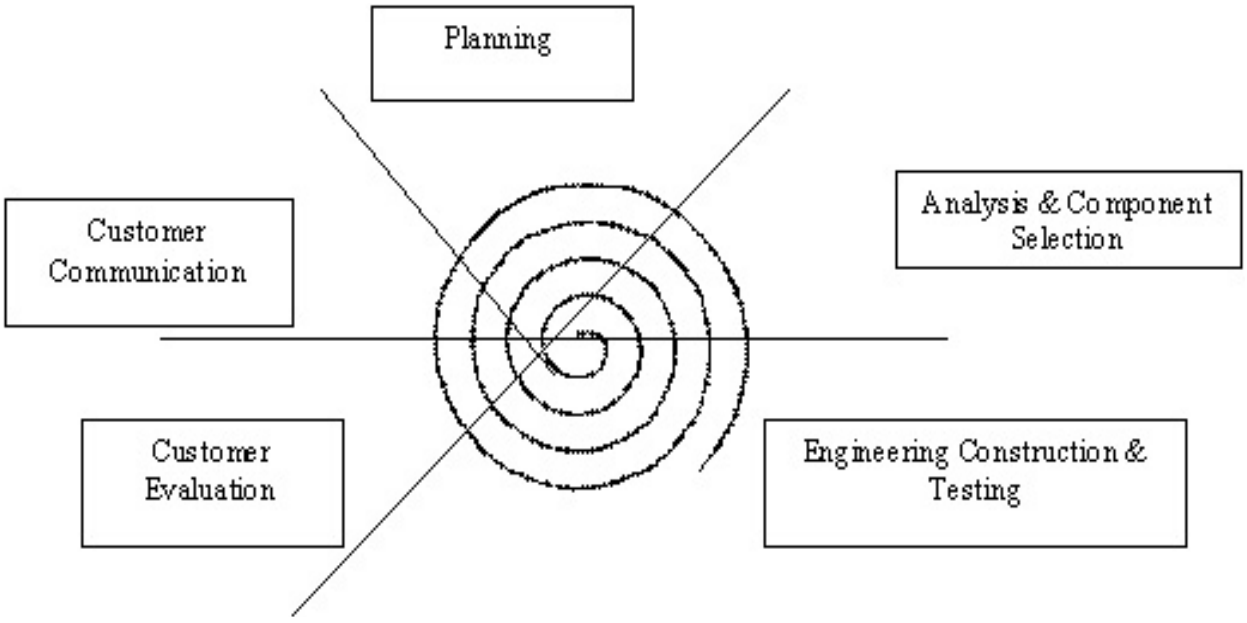


Figure 2.16: **The CBD Process Model** [60]

The analysis phase of the OOPM is modified according to the CBD process model. It is newly termed 'Analysis and Component Selection' phase. This is the phase where an analyst gathers the detailed requirements and tries to identify and select those components that can be reused. The relationships among the components are identified. The properties and behaviors of the components are identified as well [66]. The core objective of this modified phase is to reuse maximum components, rather than reinventing the wheel.

Engineering, construction and testing phase of the OOPM matches the requirements of CBD process model. The newly components are designed, developed and tested. The integration and

system tests of newly developed as well as of the reused components are performed. The customer evaluation phase of OOPM also fits to the requirements of the CBD process model.

Application development using CBD offers developers a number of benefits. As identified in [60], some of these benefits include:

- Component Reusability
- Interoperability
- Upgradability
- Saving the programmers from complexity
- Development Time and Cost effectiveness
- Makes programmers Efficient
- Reliability
- Improved Quality

Component reusability is an important advantage of developing applications using CBD. It helps the developers to concentrate on adding more complex functionalities to the applications rather than focusing on developing basic components. These merits notwithstanding, there are also a number of issues around CBD in terms of reuse. These include:

- Customization
- Adaptability
- Integration
- Security
- Efficiency

Customization of an already developed component according to the requirements of new application is a major issue in CBD [60]. The developers also face a problem to adapt a component to a new platform if it were not developed for that platform. Also, the integration of a reusable component into new component is also a major problem faced by most of the developers. Security is another major concern for the developers who reuse the components available over the Internet.

There may be a virus inside that component and may pass all the information of the business organization to attacker, who uses such an application. Efficiency of the applications developed using CBD is also debatable. The component to be reused may have extra functionalities that may be a requirement when it was developed. The new application that does not require extra functionalities becomes less efficient because of the loading time of those functions.

In all of the aforementioned, CBD is still more cost-effective; time saving and productive for the software application development, according to the state of art tools, and meets the tight deadlines of the market [49].

2.4 WEB 2.0 - Concept and Technologies

The development of grid-enabled portal is also taking advantage of the increasing advancement in Web technology [19, 29]. Beyond just providing a medium of access to various distributed resources and services to users, portals are also developed to enable community user interactions and forum, social networking, etc. This new dimension to grid-enabled portal development is based on the Web 2.0 technology.

Web 2.0 is a Web technology that results from the advancement of the Web from being a document delivery system to an application platform [20]. Sometimes it is called "*Web as platform*" [20]. It is a more socially interactive platform where users can network and collaborate for socio-economic benefits, giving various users an opportunity to contribute to the community as much as they consume [20].

There are a number of Web-based services and applications that demonstrate the foundations of the Web 2.0 concept, and they are already being used within the grid portal context too. These are not really technologies as such, but services (or user processes) built using the building blocks of the technologies and open standards that underpin the Internet and the Web. These include blogs, social networks, wikis, multimedia sharing services, content syndication, podcasting and content tagging services [20].

2.5 INTRODUCTION TO UTILITY COMPUTING

The term *utility* is used to make a description of certain services, such as electrical power services, water or natural gas, home telephone services, etc that are provided to meet the dynamic needs of various consumers [4]. The various consumers are charged for the resources based on usage rather than on a flat-rate basis. This approach, sometimes known as *pay-per-use* or *metered* services is becoming increasingly common in enterprise computing and is sometimes used for the consumer market as well, for Internet service, Web site access, file sharing, and other applications [4, 22].

Utility computing is therefore a service provisioning model in which a service provider makes computing resources and infrastructure management available to the customer as needed, and charges them for specific usage rather than a flat rate. Like other types of on-demand computing (such as grid computing), the utility model seeks to maximize the efficient use of resources and/or minimize associated costs [4].

It is a business model for computing in which resources (CPU power, storage space, etc.) are made available to the user solely on request [22]. The goal of the utility computing model is to maximize the efficient use of computing resources and minimize user costs [4, 22]. Users are able to dial up or dial down usage in real time, to meet the varying demands of business.

A utility computing infrastructure consists of both hardware resources (servers, storage, network appliances) and software resources (operating system, middleware and applications) [4]. Utility services can also be in the following domain: m-Commerce, cyber e-Health, scientific applications, e-governance, etc. Another version of utility computing is carried out within an enterprise. In a *shared pool* utility model, an enterprise centralizes its computing resources to serve a larger number of users without unnecessary redundancy. This system has the advantage of a low or no initial cost to acquire hardware; instead, computational resources are essentially rented.

A utility computing system is essentially characterized by dynamic adjustments of resource allocation for smooth service provision due to the dynamic nature of the customer requirements [23]. However, some of the factors guiding proper configuration and allocation of resources include: Performance Monitoring, Service Layer Agreement Goals, Business Objectives or Human Interaction. Utility computing services could be broadly applied on three levels, namely, infrastructure, application and business process [23].

2.5.1 Utility Computing Framework

Utility computing framework can be used to automatically create and manage multiple utility computing services on a shared infrastructure [4]. The combined potential of utility computing offers a new approach to deliver a cost effective and efficient e-commerce on-demand or pay-as-you-go kind of services to service consumers. However, this framework has also its risks and issues [4]. Utility Computing offers tremendous potential to develop a sustainable e-commerce framework. The extent to which the computing services have become scalable and economical is very large and hence, the economies of scale typical to the public utilities should also apply to utility computing [4, 5].

Based on this premise, significant amount of work has been done in developing the technologies, both hardware and software, to adapt to such an architecture. The technologies like provisioning, virtualization, consolidation, etc have now made it possible to share the computing resources among various parties without affecting the throughput and reliability requirements for the respective parties [4].

This work revolves around one of such conceptual framework for providing efficient and effective SMME e-Commerce on-Demand (SEConD) at low cost engaging the utility computing paradigm to provide the technology component to be shared among various service offered by the SMME community [7]. This is the main thrust of GUISET research agenda [1, 7].

Utility computing is an IT Infrastructure management technique that allows computing resources to be available to a customer on demand [22]. The customers subscribe to the services of the utility provider and pay only for the quantum of the resources used. In a utility computing scenario, considerable flexibility has been achieved in terms of what can be offered. A utility computing set up is basically an IT infrastructure having servers, mass storage, computing resources, middleware and applications developed into a shareable model using the technologies like consolidation, virtualization and provisioning [4]. A typical utility computing framework is represented in a diagram depicting the major components in Figure 2.17.

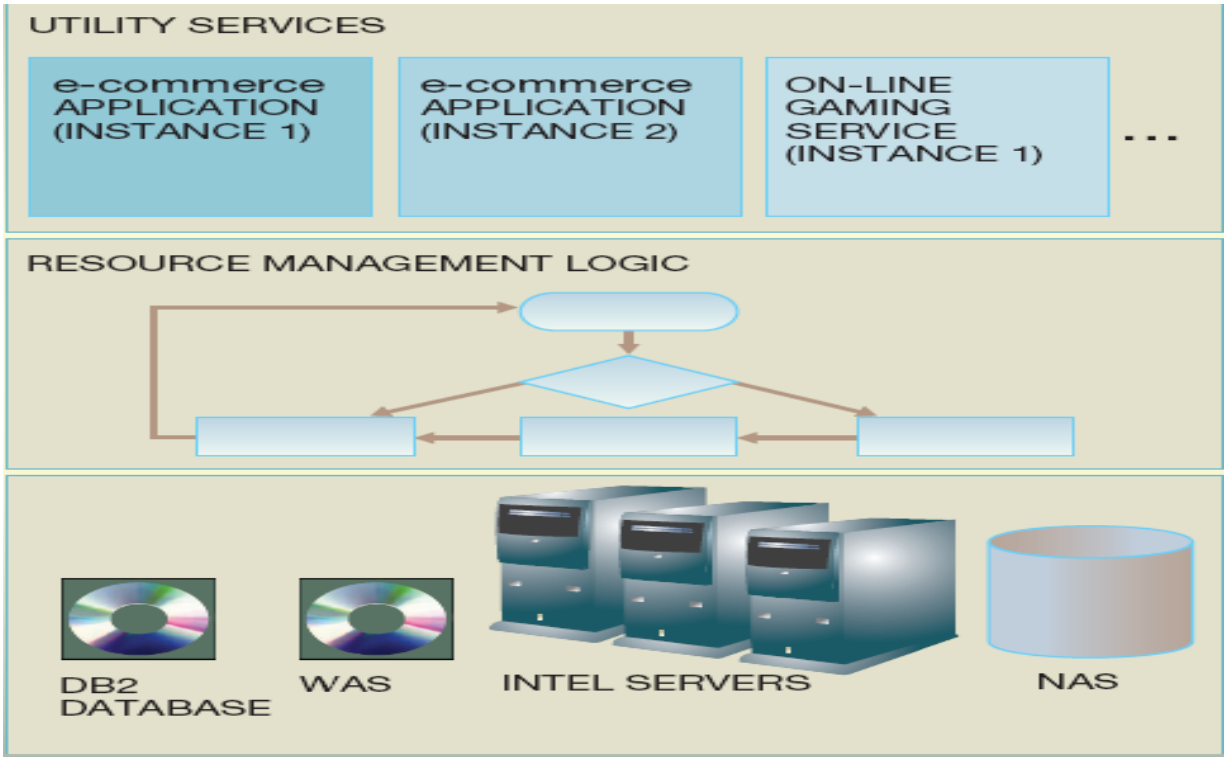


Figure 2.17: **Three Layers in a Utility Computing System** [4]

The framework has three basic layers; however the exact arrangement may vary from configuration to configuration and also the complexity and scale of the whole system.

The *Top Layer* is the Application or Utility Services layer which will have the application instances subscribed by or required by the customer. Different customers may have totally different set of applications running for them.

The *Middle Layer* typically runs a resource management logic that defines who is to use what resource, how much resource and for how long based on the customers' subscription details or the Service Level Agreement (SLA) and the demand.

The *Bottom Layer* utility computing system is the actual IT infrastructure that provides the computing power to the various services. The IT infrastructure is abstracted into "containers" wherein each customer has an independent application environment and has an independent view of the underlying infrastructure. This has been achieved due to the advancements in the Server Virtualization and Consolidation technologies [4]. The utility computing system typically runs from inside a data-center which is like a large container inside which the whole IT infrastructure is assembled as a single unit.

2.5.2 Utility Computing Approach to Service Delivery (Pay-As-You-Use)

This is the most attractive feature of a utility computing model [67]. It allows the user of the computing services to pay according to the usage of the customer. This provision allows any customer to cut the costs on its IT spending as there is now no need to procure and maintain the complete capacity infrastructure [22, 67]. Customers can simply subscribe to the utility computing service provider and use the computing resources at will while paying only for as much as they use. Typical measures of usage include metered CPU hrs, memory space usage and other such metrics [67].

The utility computing community has developed a number of pricing models to address various needs of the customer. As highlighted in [67], some of them are:

1. Fixed Price Model
2. Cost Plus Model
3. Subscription Model
4. Pricing as a function of business revenue generated by the customer using the utility computing services.

There is also a consideration of how to implement variable pricing mechanisms. Some of the options available are:

1. Price rate is directly proportional to the usage. So the price rates increase as usage increases to discourage the wastage of computing resources.

2. Price rate inversely proportional or discounts off as the usage level increase. Thus rewarding the customer on higher usage patterns and incentivizing the customer on using more computing resources.

3. Subscription plus price slabs for different usage levels.

Any of the above mentioned approaches to pricing can be implemented by the utility computing providers based on their business model and context.

2.5.3 The Benefits of Utility Computing

The Utility Computing based solution will deliver the computing power to all the participating and potential enterprises as a “utility” like electricity or water supply service. This shift in how the

computing power is sourced has significant and everlasting benefits for SMME e-commerce communities. The major benefits are [5]:

1. *Decrease in the total cost of capital expenditure:* The enterprises join the community of utility service users are relieved from the need of procuring the entire IT infrastructure required for enabling their business processes. A utility computing service provider will provide the IT infrastructure required.

2. *Maximum Resource utilization:* The under utilization of the IT resources is eliminated as the utility computing service provider will provide the infrastructure on a shared basis to other applications resulting in maximum utilization.

3. *Minimizing Resource Wastage*: The shared approach to resource utilization will result in minimum wastage of the computing resource due to less idle time.

4. *Quality of Service:* The utility computing approach has the provision to enforce penalties on the service providers if they fail to meet the performance criteria set in the contracts or the Service Level Agreements. This will ensure the consistency in the level of quality of the services offered.

5. *Integrative Approach:* Utility computing offers a unique opportunity to develop an integrated e-commerce framework providing a powerful platform to provide end-to-end service options to the customers/users.

6. *Offsetting Human Resource need:* The enterprises will be totally free from the responsibility of maintaining the IT infrastructure required for enabling their business. The human resource required for operations and maintenance efforts will be sourced by the service providers.

7. *Flexibility and Adaptability:* Since such a model abstracts the users from the backend IT infrastructure, therefore it will be easier to change the backend IT configurations without affecting the end – user in a quick time.

Utility computing cut across a number of application areas and boundaries. Some of these application areas include: computing service, network service, data center service, Web hosting services, e-mail services, groupware services, office suite services, payroll service, CRM service, ERP service, storage service, etc. One of the boundary areas include: strategic applications where

flexibility and customization are critical. The benefits that utility computing approach has to offer are multidimensional and well suited for the SMME e-commerce domain.

2.6 OVERVIEW OF GRID-ENABLED PORTAL SYSTEMS

There have been many Grid based portals for specific application [29]. In this section a classification of the categories of Grid portals and their development frameworks is reported. Again, portals are defined as graphical user interfaces that a user employs to interact with one or more infrastructural resources.

Grid based portals can be classified into five (5) categories and two (2) development frameworks [8]. These five categories include:

1. Portals providing a single access point for user support. e.g. Global Grid User Support System, GGUS [68]
2. Portals providing a user-friendly access to services of a single grid; e.g. MD-web [69], the Grid Enabled Web eNvironment for site Independent User job Submission, GENIUS grid portal [70], and AccessGrid [71].
3. Portals providing access to services of multiple grids; e.g. The P-Grade Portal [72]
4. Portals supporting grid enabling applications; e.g. LUNARC Application Portal [73].
5. Portals supporting workflow. E.g. P-Grade [72]: (PG-web) specifically supports workflows, and Batch Object Submission System, BOSS [74].

The Portal development frameworks include:

1. Frameworks for building grid portals; and
2. Frameworks supporting grid accessibility via various media delivery channels.

2.6.1 Grid-enabled Portal Development Frameworks

Portal frameworks are development platform for portal development. They are design structures that contain various modules, methods and software features used for developing specific portal [8, 10]. They provide a skeleton to plug and play various portlets [8].

A. Frameworks for building grid portals

EnginFrame (EF-web) [75] is a web-based innovative technology, by the Italian company Nice Srl [90], for grid enabling web portals. Grid-enabled web portals using EnginFrame can access services of various grids (including Globus [37], and gLite grid middleware [76]. The objective of EnginFrame is to facilitate the task of grid enabling web portals [85, 77]. The Genius portal (Genius-web) [70] is an example of a portal built using EnginFrame. However, there a number of open source portal frameworks which include: uPortal [78], Liferay [26], GridSphere [79], Grid Portal development Kit [80], etc. The most widely-used portal construction technology is the GridSphere JSR-168 compliant portlet container [18, 79]. The P-Grade (PG-web) [72] is an example of a portal built using GridSphere.

B. Frameworks Supporting Accessibility via Alternative Delivery Channels

Beside the above overview of grid portals and frameworks for grid enabling web portals and applications, several research efforts are underway to provide grid accessibility via alternative delivery channels including hand-held devices and mobiles [81, 82]. Most of the research focuses on re-engineering existing grid middleware to achieve this aim. This is mainly due to the fact that most grid services involve computational and resource intensive tasks that cannot be easily ported to end-user devices [30, 81, 82].

2.6.2 Evaluation of Grid-enabled Portal Development Frameworks

With the popularity of portals today there are many open source portal frameworks available and list of these open source frameworks is all the time increasing. A thorough and non biased evaluation of some of these frameworks with respect to a broad range of criteria to accommodate the specialty of each framework and maximum consideration of user requirements was done as reported in [10, 25] and a summary is presented here in order to provide a guideline for portal developers to choose appropriate ones.

These criteria were based on core and optional functionalities/requirements and they include in the order of perceived importance: (i) Their compliant with the development standards, JSR-168, and WSRP; (ii) Ease of Installation; (iii) Documentation Standard (iv) Online Support; (v) Portal Management; (vi) Portlet Resources; (vii) Performance and Scalability; (viii) Security; (ix) Technology Used; (x) Portal Features; and (xi) Server Dependency. Each Portal Framework was

given a score of 1 to 5 against each criterion, 5 being the most effective. The total score of each portal framework is shown at the end of the tabular comparison below, with a visual Bar Graph also following.

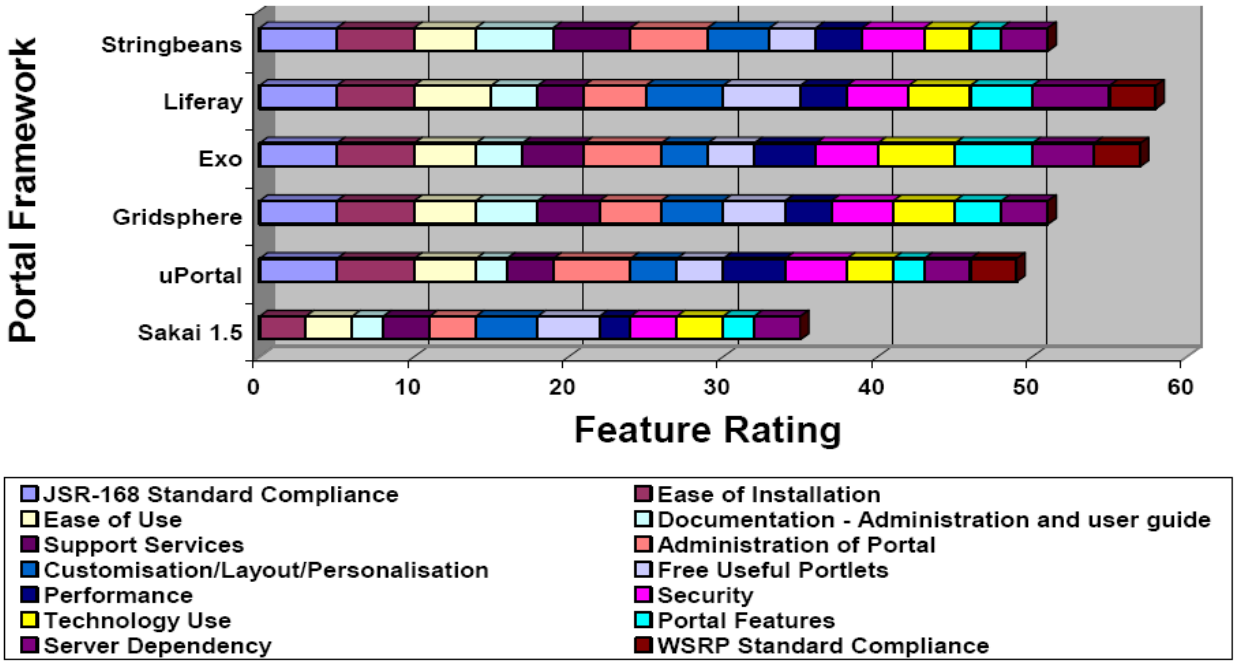


Figure 2.18**: The Evaluation Result as Bar Chart** [10]

Criteria	Portal Framework					
	Sakai 1.5	uPortal	Gridsphere	Exo	Liferay	Stringbeans
JSR-168 Compliance	0	5	5	5	5	5
Ease of Installation	3	5	5	5	5	5
Ease of Use	3	5	4	5	4	5
Documentation	2	2	4	3	3	5
Support Services	3	3	4	4	3	5
Administration of Portal	3	5	4	5	4	5
Customisation	4	3	4	3	5	4
Free Useful Portlets	4	3	4	3	5	3
Performance	2	4	3	4	3	3
Security	3	4	3	4	4	4
Technology Use	3	3	4	5	4	3
Portal Features	2	2	3	5	4	2
Server Dependency	3	3	3	4	5	3
WSRP Compliance	0	3	0	3	3	0
Total	35	49	51	57	58	51

Table 2.2: **The Evaluation Result** [10]

2.6.3 Review of Related Existing Works

Several research projects including the HotPage user portal, the Gateway project, and UNICORE [31] have employed the concept of developing a web enabled access medium to the Grid [80]. These several projects had similar goals in trying to grant a uniform and easy access to Grid resources and services, though with differing technologies and design. Several grid-enabled portals have also been developed for specific applications [29].

In this section, a review of a number of the works done so far are reported as used as theoretical baseline in this research work. The Grid Portal Development Kit addresses many of the same issues related to providing secure, web-based access to resources as the previous projects, but differs in three important ways. First, the core of GPDK resides in a set of generic, reusable, common components used for accessing the various Grid services that are supported by the Globus toolkit [37]. Second, a portal user is provided with a persistent, customizable profile that contains information that is stored securely on the portal and provides details on past jobs submitted, the set of computers they have access to, and any other information that is of interest to a particular user. Third, GPDK is designed to provide a complete development environment for building customized application specific portals that can take advantage of the core set of GPDK Grid service components and the Model-View-Controller, MVC architectural model [80, 83].

A core part of the design philosophy of GPDK was the separation of logic from presentation by adopting the Model-View-Controller, MVC design pattern [83]. This makes it easier to develop new functionality that could plug into the existing framework by following a prescribed recipe. The design is also based on providing multi-user access to Grid services and resources [80]. From the GPDK project came many lessons about building a framework and developing reusable components. Ultimately, while the GPDK template/demo portal could be enhanced to create a project specific portal, users had to become familiar with the source code in order to add the features they needed. Another major limitation was the lack of any reusability in the presentation layer. Developers would need to handcraft customized presentation pages to re-use the GPDK services provided to create a new portal instance. This further increase the development time and cost.

A similar design to the GPDK is the Astrophysics Scientific Collaboratory (ASC) portal [30], but it was ultimately specialized in its functionality and services to suit the needs of a particular user community. In both the GPDK and the ASC portal, an emphasis was placed on providing value-added capabilities that would encourage users to perform their work via the portal. One of the lessons learned in putting the portal into use is that the portal is only as good as the services used; hence a major difficulty was managing the underlying, quick changing grid software libraries that are used.

In trying to build support for the Grid user community, Novotny, et al. (2004), developed the GridSphere portal framework [11, 29] based on the many previous lessons and best practices learned from passed notable Grid portal projects such as the GPDK and the ASC portal. It was aimed at offering external developers a model for easily adding new functionality and hence increasing community collaboration.

The GridSphere Portal Framework [11] is a portlet JSR-168 compliant portlet container that offers a set of base classes and tools for developing portlet application. It has been used as a development platform for a number of projects around the world such The UK GridLab project [10], the UK e-Science projects [16], etc. The Grid portlet web application [43], released for the first time in June 2005, builds on the core features in the GridSphere portal framework to provide developers with a framework for developing Grid-enabled portlets [11]. These portlets are built upon reusable Java Server Pages, JSP [84] based user interface components.

GridSphere enables developers to quickly develop and package third-party portlets based web applications that can be run and administered within the GridSphere portlet container [13]. Although, GridSphere does not in itself contain any support for using Grid technologies, it only contains the core functionality necessary to develop a web portal. It is based on IBM's WebSphere [51] and provides a "*white-box"* framework in which users can override base classes and 'hook' in their method. It therefore requires that developers and users have some knowledge of base framework classes and interfaces. The GridSphere adopted the portlet technology, but with no major emphasis on the implementation of the WSRP specifications [10, 25].

A portlet-based Grid portal architecture was proposed in [14] by encapsulating one or more Grid services to a portlet. The aim was to make portals more flexible and easily configured to suit a Grid user's need and improve the reusability of the grid portal. The work proposed a portlet-based Grid portal architecture for integrating existing technologies under a common interface. It

developed a prototype of the portlet-based Grid portal using Jetspeed-2 [85] as the portal framework which employs Pluto [48] as the portlet container. GT 3.2 was used as the underlying Grid infrastructure. The work adopted the JSR 168 specification for the Grid-based portal development but, yet to support the WSRP specification.

2.7 OVERVIEW OF EXISTING METHODS

SOA-based applications are built on a layered architecture [15]. A four-layered SOA architecture was presented in [86] in a bid to propose a systematic service oriented analysis and design (SOAD) process for developing highly adaptable services. Each layer of the architecture is defined with its own goal and their artifacts [86]. This is as depicted in the figure 2.19 below.

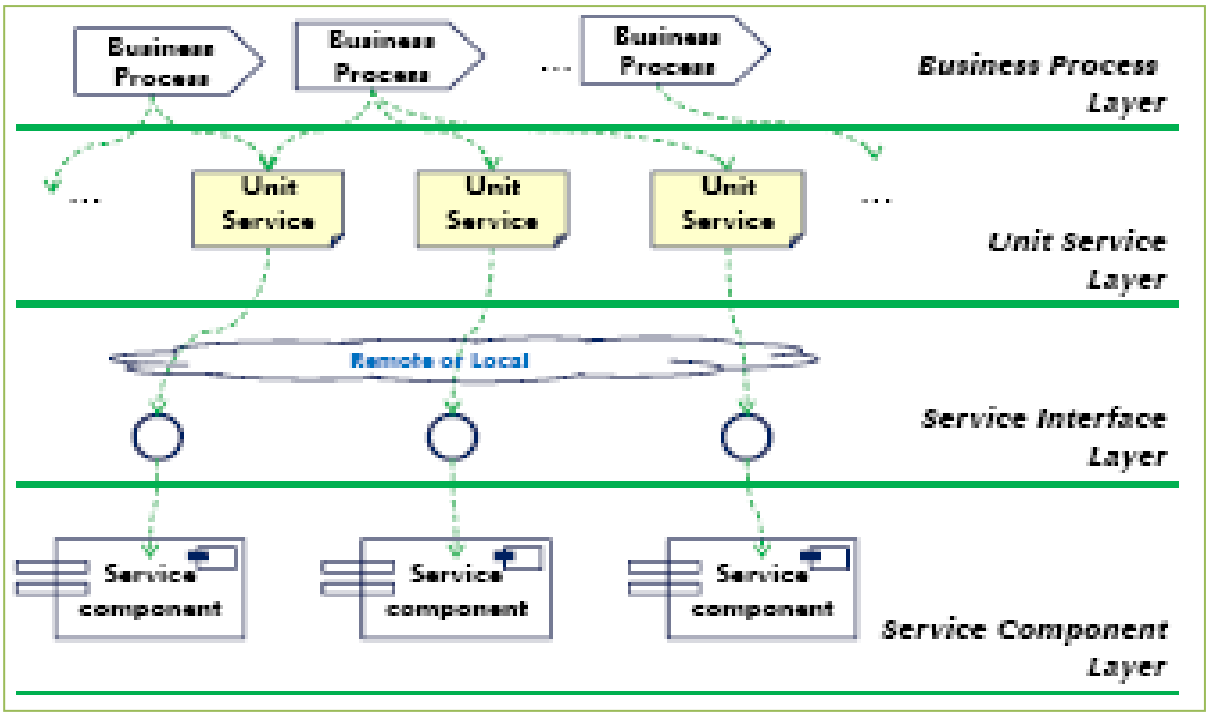


Figure 2.19: **A Four-Layered SOA Architecture** [86]

- ***Business Process Layer****:* As the top layer, it is to define business processes expected by service clients. *Business process (BP)* represents a cohesive unit of the service perceived by service clients, not by component engineers. Hence, it is defined independently from implementation technology and platforms. Typically, a BP is a larger grained than a use case and a method of objects, and it is defined with a *service workflow* among smaller grained *activities.* Hence, *Business Process Specification* includes *workflows* of participating *activities.*

- ***Unit Service Layer****: Activities* of a business process are conceptual units of works, perceived by clients. It will eventually be performed by a software element, which was called a unit service, *Unit Service.* That is, an activity is fulfilled by running a unit service. The main distinction between them is that an activity is a conceptual unit perceived by clients and a unit service is its corresponding task defined from engineering perspective. Hence, the notion of unit service is a vehicle that bridges clients' view to engineers' view. Another key value of introducing unit service is that it can be reused by more than one activity, i.e. more than one BP. That is, activities of the workflows can be analyzed, and a set of unit services also defined. Some unit services may be common among the business processes, and hence they are reusable among several business processes.
- ***Service Interface Layer****:* In Service-Oriented Computing (SOC), the interfaces of services are specified separately from service components, and service providers publish the services in WSDL in UDDI service registries. Hence, the unit services identified should be bound to *interfaces* of the published services which fulfill the requirement of the unit services. Therefore, the *Service Interface Layer* contains the interfaces of services published by service provides, and it separates the unit services from the service components. By having this layer, unit services can be bound to any compatible interfaces, and the interfaces can be realized by and bound to any compatible service components.
- ***Service Component Layer****:* This layer is to specify service components which implement the service interfaces. Some components are like the one in component-based development (CBD), and typically implemented with objects on OO/CBD platforms such as EJB. Other components can be simply wrappers of legacy applications. There is a difference between the two types on how components are implemented, but they both have to provide physical interfaces that conform to the published interfaces (in WSDL) of the *Service Interface Layer*. For example, we may implement service components in EJB and provide physical interfaces in the forms of *EJB Home* and *Remote* interfaces, which conform to the WSDL service interfaces.

Service-Oriented Modeling and Architecture (SOMA) [15] focused on the techniques for the identification, specification, and realization of services, service flows, compositions and enterprise-scale components. To model the architecture, the work proposed a seven-layered architecture shown in the figure 2.20 below.

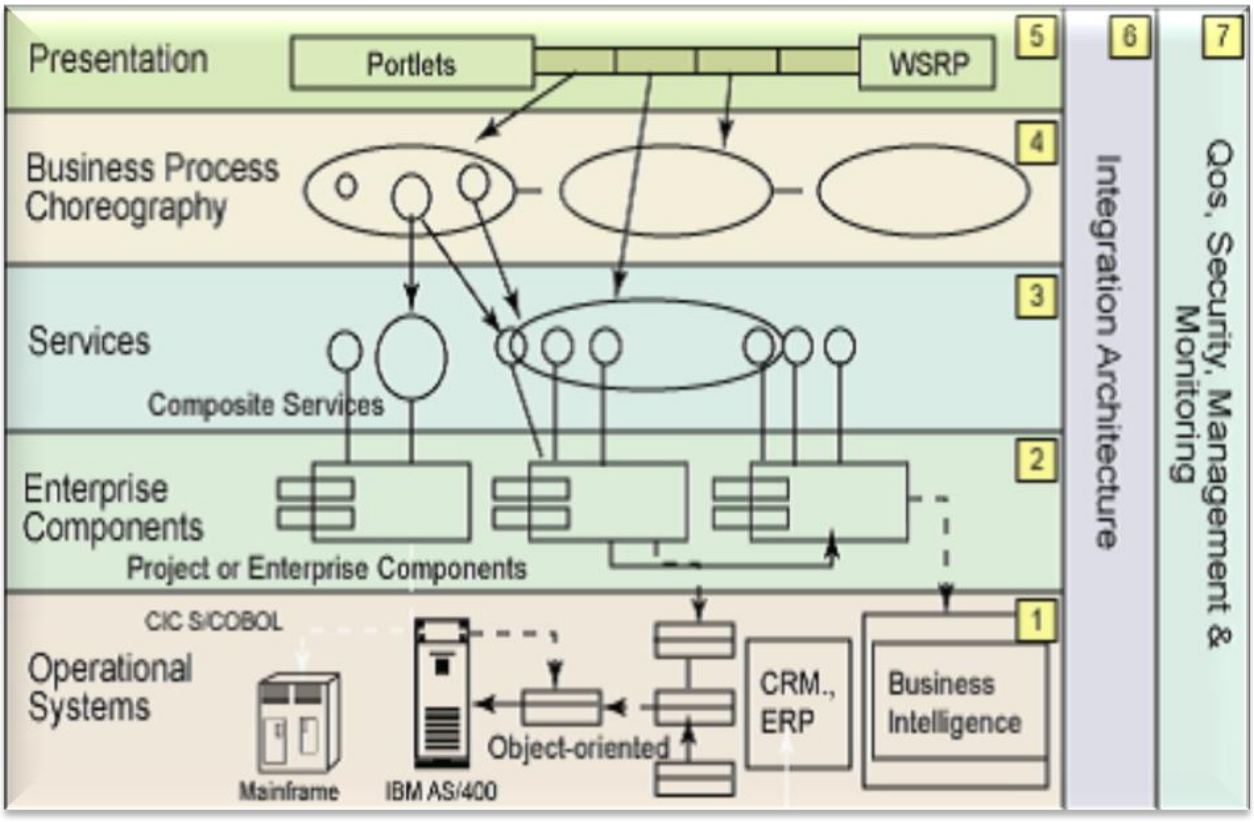


Figure 2.20: A Seven-Layered SOA Architecture [15]

The core of the GUISET research framework is built around this reference architecture and a number of both on-going and completed research works at the center of excellence for Mobile e-Services, Department of Computer Science, University of Zululand, RSA were situated within this architecture [15]. The various layers are highlighted below.

- ***Operational layer*** (layer 1): This layer of the SOA architecture consists of various existing custom built application otherwise referred to as *legacy systems*, including existing Content Resource Manager (CRM) Enterprise Resource Planner (ERP) and other packaged applications, and older object-oriented system implementations, as well as business intelligence applications. The composite layered architecture of a SOA can leverage existing systems and integrate them using service-oriented integration techniques.
- ***Enterprise Components Layer*** (layer 2): This is basically responsible for the realization of various functionalities and maintaining the quality of service, QoS of the exposed services. These special components are a managed, governed set of enterprise assets that are funded at

the enterprise or the business unit level. As enterprise-scale assets, they are responsible for ensuring conformance to service layer agreements, SLAs through the application of architectural best practices. This layer typically uses container-based technologies such as application servers to implement the components, workload management, high availability, and load balancing.

The enterprise components layer is at the core of the GUISET architecture which aimed at developing components for the composition of product family members: a product instance determines the functionality that is exposed as web services to clients. Service Component Architecture, SCA and Case-Based Software Engineering, CBSE paradigms are both being explored to determine how the components are built in line with the SOA principles.

- ***Service Layer*** (Layer 3): This layer houses the various services the business chooses to fund and expose. These services can be *discovered* or be statically bound and then invoked, or possibly choreographed into composite service. This service exposure layer also provides for the mechanism to take enterprise scale components, business unit specific components, and in some cases, project-specific components, and externalizes a subset of their interfaces in the form of service descriptions. Thus, the enterprise components provide service realization at runtime using the functionality provided by their interfaces. The interfaces get exported out as service descriptions in this layer, where they are exposed for use. They can exist in isolation or as a composite service.
- ***Business Process Choreography Layer*** (layer 4) is otherwise called the *Business Process Composition layer*. Compositions and choreographies of services exposed in layer 3 are defined in this layer. Services are bundled into a flow through orchestration or choreography, and thus act together as a single application. These applications support specific use cases and business processes. Here, visual flow composition tools, such as IBM® Websphere® Business Integration Modeller or Websphere Application Developer Integration Edition, can be used for the design of application flow.
- ***Presentation Layer*** (layer 5): This is often referred to as the *Access Layer*. Although this layer is usually out of scope for most discussions around a SOA, it is actually gaining much relevance because there is an increasing convergence of standards, such as Web Services for Remote Portlets, WSRP version 2.0 [13], and Java Specification Request, JSR 168 [18], and other technologies, that seek to leverage web services at the application interface or presentation level. We envisage this layer as a future layer that we would need to take into

account for future solutions. It is also important to note at this point that SOA decouples the user interface from the components, and that an end-to-end solution would ultimately be needed from an access channel to a service or a composition of services [15]. This layer houses the portal frameworks and models that are the research focus of this work.

- ***Integration Architecture*** (layer 6) enables the integration of services through the introduction of a reliable set of capabilities, such as intelligent routing, protocol mediation, and other transformation mechanisms, often described as the ESB [52]. WSRP [13] specifies a binding, which implies a location where the services is provided. On the other hand, an ESB provides a location independent mechanism for integration [52]. The integration layer house the core functionalities of the GUISET infrastructure for service integration.
- ***Quality of Service, QoS layer*** (layer 7): provides the capabilities required to monitor, manage and maintain QoS such as availability, performance, responsiveness and security [52, 87]. This represent a background process through sense-and-respond mechanism and tools that monitor the health of the SOA applications, including the all important standards implementation of WS-management and other relevant protocols and standards that implement QoS for a SOA [52].

SOA, from an abstract view is depicted as a partially layered architecture of composite services that align with business processes. The representation of this type of architecture is depicted in figure 2.22. The relationship between services and components is that enterprise-scale components (large-grained enterprise or business line components) realize the services and are responsible for providing their functionality and maintaining their quality of service [15]. Business process flows can be supported by choreography of these exposed services into composite applications. Integration architecture supports the routing, mediation, and translation of these services, components, and flows, using an ESB. The deployed services must also be monitored and managed for quality of service and adherence to non-functional requirements.

CHAPTER THREE

3.0 REQUIREMENTS ANALYSIS AND DESIGN

3.1 INTRODUCTION

This chapter captures the formulative aspect of this work. It entails the elicitation and elucidation of the system requirements which drive the design choices, and the system design. A formal model of the portal system is built with the use Unified Modeling Language (UML) tools, particularly the portal use cases were modeled using a number of UML use case tools.

3.2 THE SYSTEM REQUIREMENT ANALYSIS

3.2.1 The GUISET Architecture

GUISET is depicted a Mobile Grid-enabled Utility Computing Architecture [7]. It is built on the following motivation:

- The need to leverage the success of handheld devices with mobile mode of utility computing.
- Mobile Computing on fixed infrastructure e.g. Internet suspend/respond.
- Software architectural support for handheld computing that enables composition of large, distributed, decentralized mobile systems.
- *Reference Architecture* (see figure 3.1) that can be used as the basis for sharing domain-specific applications.

According to [7], GUISET is also envisioned as an infrastructure for enabling SMMEs:

- ***Who?*** *:* Under-resourced SMMEs are targeted as the main beneficiaries of this technology;
- ***How?*** *:* Operating overhead is to be reduced to the minimum;
- ***Why?*** : To make the transformation of a small business to an e-business a priority;
- ***What?*** : Acquire capability to use e-Commerce tools without owning them.

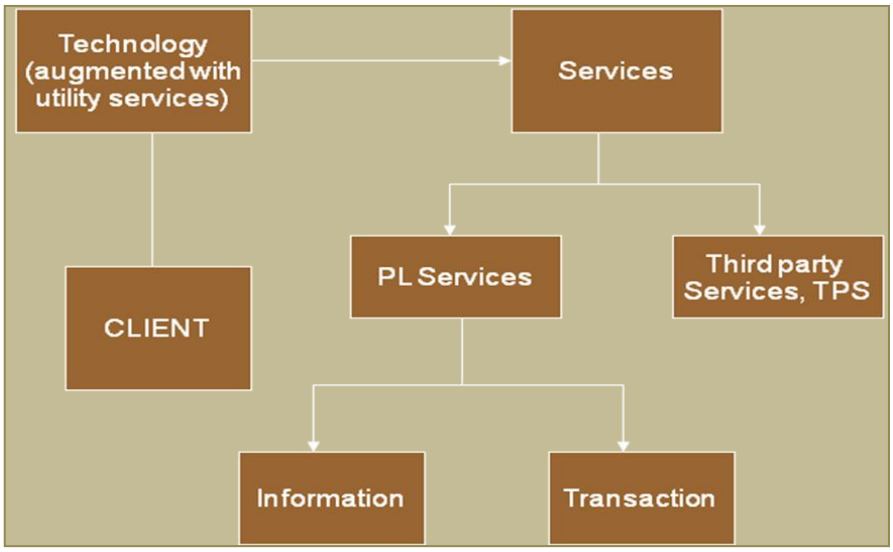


Figure 3.1**: The Reference Architecture** [7].

GUISET is designed as a three (3) layered architecture. It comprises of (i) *Multi-modal Interface layer* (ii) *Middleware layer* and (iii) *Grid Infrastructure Layer*. This is shown in the figure 3.2 below:

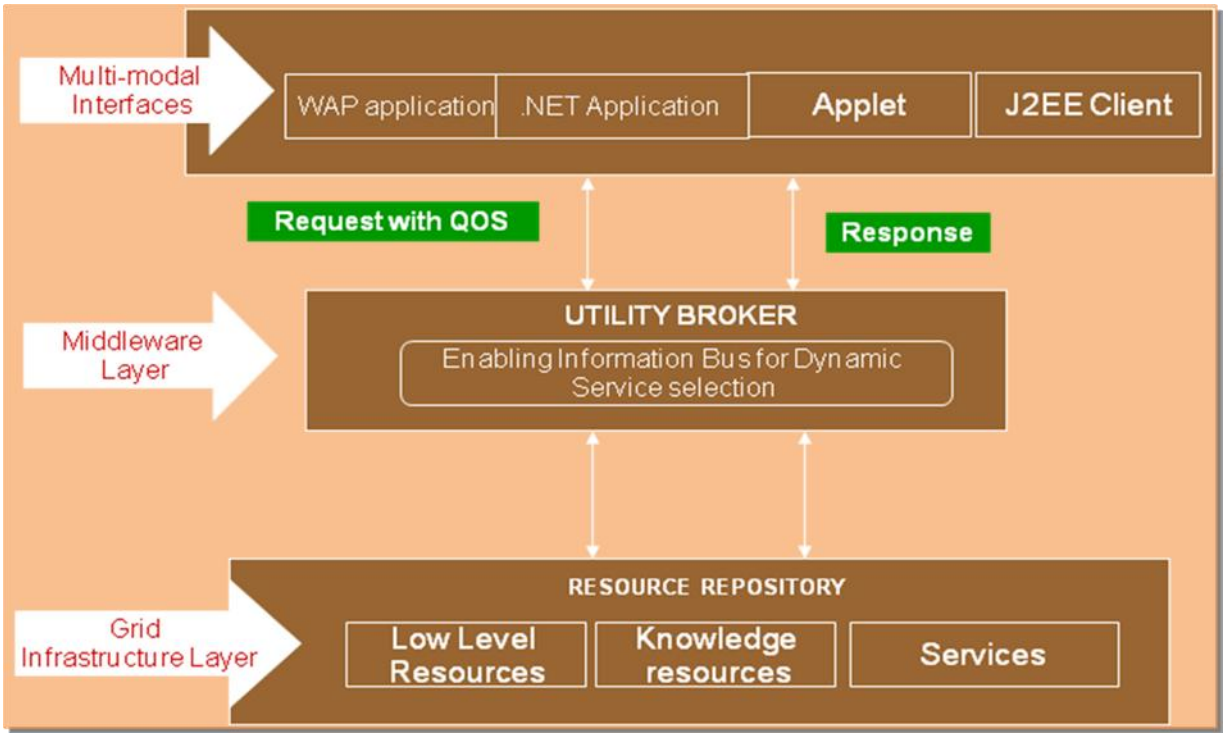


Figure 3.2**: The GUISET Architecture** [7]

The *Multi-modal interfaces layer* houses the various application interfaces designed for accepting customer subscription. The interfaces run on a Grid client which can be a mobile device or laptop. Each client is a potential Grid service provider or resource. The services available are also advertised through these interfaces. This layer also provides a template for customer specification of service parameters. These templates are then passed to the utility broker for a SLA-driven validation of all completed templates.

The *Middleware Layer* comprises the utility broker, enabling information bus for dynamic services selection. The utility broker component works with validated service specification templates. It initiates a negotiation process with customer until a mutual agreement is reached and a contract is established. It also invokes a subscription manager that enforces and manages updates to all existing contracts. The billing component of the broker collaborates with subscription manager to determine what and how services should be billed. The SLA management dynamically increases or decreases user Quality of Service, QoS requirements automatically as dictated by policies or as the premium subscribers choose from time to time.

The *Grid infrastructure layer* is the resource repository that stores all the services and resources.

This single architecture is partitioned into two subsystems:

1. The *Web Infrastructure*, and
2. The *Service Portal.*

The GUISET Infrastructure is based on SOA, and it is designed to provide the operating environment for prospective utility service customers willing form or join a user community. It enables the activation of GUISET membership; accepting the various SMME groups' subscriptions. GUISET is envisioned as a shared infrastructure which will be implemented as a Business-to-Customer, B2C Web service portal core. The business side consists of owners and service providers or sellers, while the consumers' side consist of the subscribers to services and resources.

GUISET provides the technologies for end-to-end SLA, service composition, fault handling, trust-based security services, context-aware adaptation and personalization of services. It is also envisage as creating a set of policies, SLA templates, and a simple web presence for each seller or

provider. Members of a seller group will then be matched to individual QoS specification signified by their membership status.

The GUISET portal however is meant to provide the street level entrance into a bring-and-share mode of utility computing. It is the service portal into which future services can be plugged. It therefore forms the basic infrastructure for the various application projects.

3.2.2 The Proposed GUISET Portal Framework

Prior to this work, in [14] a *portlet-based Grid portal architecture* was proposed for integrating existing technologies under a common interface. GT 3.2 [37] was adopted as the underlying Grid system and a portal prototype was developed using JetSpeed 2 [85] which is JSR-168 compliant, as the portal framework. Unfortunately, the work did not satisfy the WSRP component of the grid-enabled portal development standards and specification.

This work therefore, proposed a modification of the above architecture in order to achieve a more standardized version of the portal architecture by introducing the WSRP component. This illustrated in the figure 3.3 below.

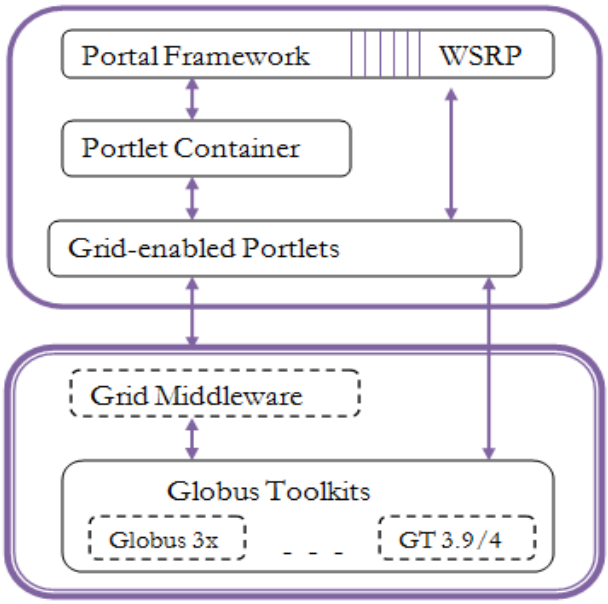


Figure 3.3: **The Proposed Grid-enabled Portal Framework**

The above architecture consists of the portlet-based grid-enabled portal layer and the underlying, enabling Grid technologies. The Portal layer consists of the *WSRP-compliant portal framework* [10], the *portlet container* and the various *grid-enabled portlets for encapsulating one or more Grid services* [25, 29]. The underlying Grid technologies which could also be referred to as the Grid tools, consists the Globus toolkit and Grid middleware. The Grid portlets interact with the Grid tools.

The Grid portlets can access Grid resources either directly through Globus Toolkit or indirectly through Grid middleware. GT 3.x, GT 3.9 and GT 4.0 [37] are the popular versions of the Globus toolkit. GT 3.x is a reference implementation of the Open Grid Services Infrastructure (OGSI) [88], and it conforms to the Open Grid Services Architecture (OGSA) [40]. GT 3.9 and GT 4 are recent versions of Globus Toolkit, supporting Web Service Resource Framework (WSRF) [58, 59].

Grid middleware is another underlying Grid technology or tool that can be introduced between the low-level Grid services offered by Globus Toolkit and the presentation layer such as Grid-enabled Portal [12]. This middleware can ease and reduces the development time and cost Grid-enabled Portal. One of such middleware is the GridPort [41, 82, 89]. Thus, developers can easily develop some of the Grid portlets by using GridPort to access Grid resources. Different composition of Grid portlets provides end-users with different functionalities.

Grid portlets are managed by a portlet container [10], which runs portlets and provides them with the required runtime environment and manages their lifecycle. Pluto [48] is a widely used portlet container and is the reference implementation of the JSR-168 portlet specification. It supports portlets written with many different programming languages, including JSP, JSF, Perl, PHP, etc. Some portal frameworks use their own portlet containers, such as GridSphere [11] and uPortal [78].

3.2.3 The Portal System Analysis

System analysis is a problem solving technique that entails the decomposition of the studied system into its component parts with the view of studying the various functionalities and how those component parts interact to accomplish the system's purpose [90]. It describes a set of activities that are aimed at understanding the system under study and this collectively defines the early phases of the system development.

The portal system analysis however, seeks to identify and analyze the various components of the portal. The portal requirement analysis is also done here. This helps to enhance the system design choices.

A. Informal Requirements (The Portal Scenarios)

The portal is designed to provide a uniform access to various both service providers and subscribers to various services and resources on the Grid-based utility infrastructure. It is used to accommodate different business clusters e.g. SMMEs cluster of business owners and service providers that form the community of registered business entities.

A typical scenario is described as follows and is also depicted in figure 3.4 below.

The prospective utility service clients (providers or customers) such as Kabini B&B, Shebak C&D, Zhuklu A&A, etc. form or join a user community, based on similar business goals or common operating domains such as Tourism, Art & Craft, Fashion, Health, etc. Each community cluster is registered on the GUISET infrastructure as a distinct business entity strictly based on the business domain or type of goods and services offered. Each cluster is also associated with specific type of goods or services. This makes it possible to validate membership against the vision and mission of the cluster that the client selects to belong. Only SMMEs that qualify are allowed to belong to specific clusters, the unqualified applicants are channeled to appropriate clusters.

Members (service providers) own their resources and they contribute them to a shared pool. However, customers need not own their hardware & software infrastructure, nor know where the services are deployed; they only need to join a user community to have access to GUISET utility services. The GUISET portal is meant for the registration of members – owners, suppliers, customers, subscribers, etc. The client's access to the portal is a Web client and a generic client is included to enable both application and client requests for non-portal services. However, both the

GUISET portal and client have the capability to find and bind to services published in the registry. But portals look for WSRP services only which are portlets.

Services exposed on the GUISET portal have either a Native Component or Portlets as their backend. Therefore, contributed services are exposed on a Hardware-As-A-Service Basis. So the Portal will be designed as an interface for: Community Enablement, Registration of SMMEs clusters, Support Membership Management, Administration of SMMEs subscription, Discussion forum, Links, etc.

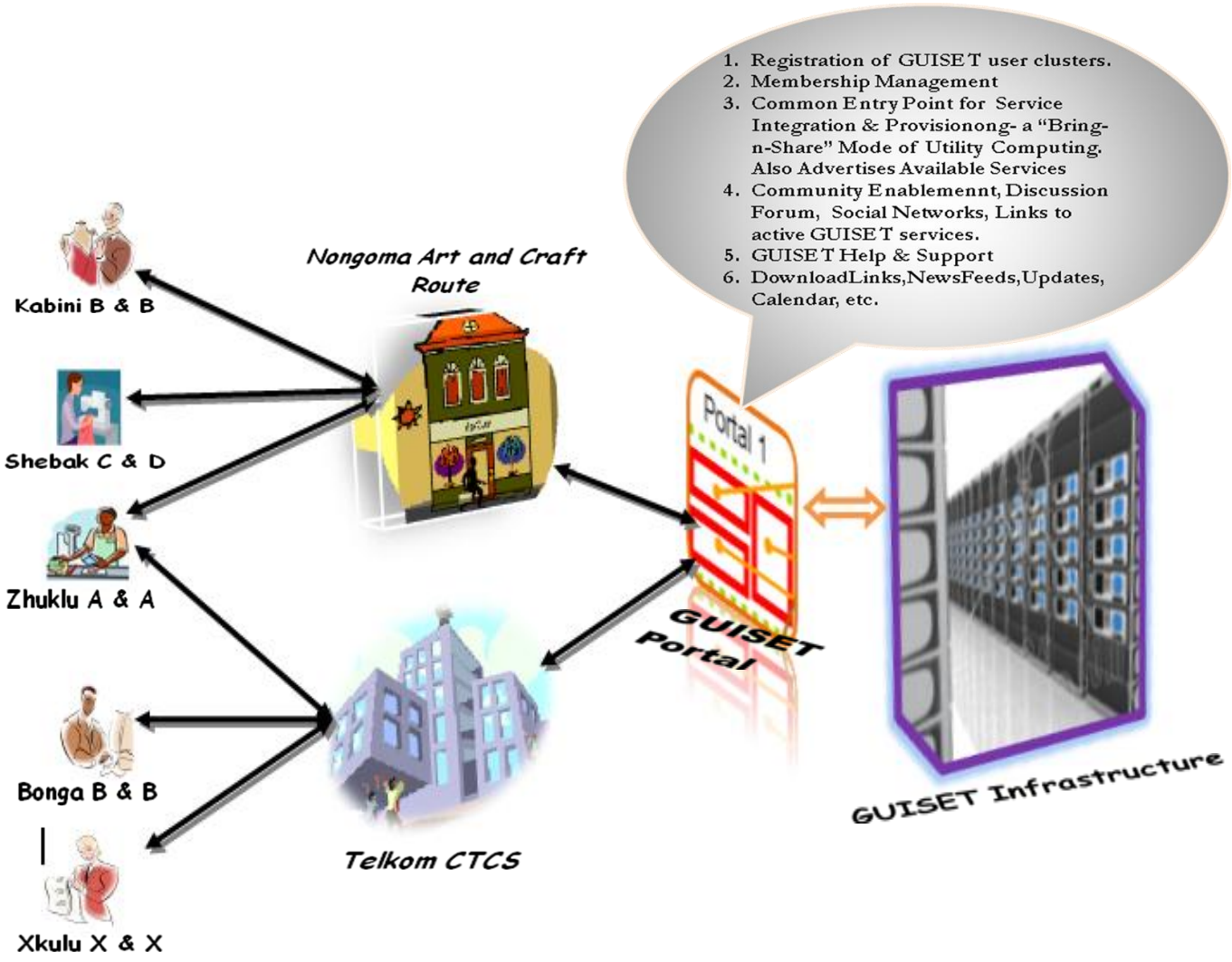


Figure 3.4: **The GUISET Portal Scenario**

B. The Portal Formal (Functional) Requirements Specification

The GUISET portal is conceptualized as two separate web interfaces or sites. These are:

1. GUISET Infrastructure Portal;
2. GUISET Service-driven e-Commerce on-Demand (SEConD) Portal.

While the latter is a sort of complementary portal interface which provides e-Commerce services on-Demand, the former which is the primary focus of this research work to be designed and implemented to meet the following requirements stated as follows:

1. Secured registration of GUISET user communities (Business cluster);
2. Supports membership management;
3. Common/Uniform point of entry for service integration and provisioning (s*ervice provisioning using portal paradigm*) - a "bring-and-share" mode of utility computing;
4. Advertisement of available services or products;
5. Administration of Users' (SMMEs) subscription;
6. Provides a template for customer specification of service parameters;
7. Supports GUISET user community enablement: discussion forum, chat rooms, blogs, newsfeeds and service updates, download links, links to active GUISET services, help and support links, time and calendar, etc.

The Portal Functionalities also envisaged are:

- A Business to Consumer portal capabilities – product or service information and ordering capability available.
- Portlets are provided for specific product or service categories.
- Collaboration functionality is provided by the portal to create places of discussion of products or topics of interest. Customers subscribe to topics of interest so they can see when others of like interest are online for discussion.
- Instant messaging could be used to exchange ideas with those online.

3.3 THE SYSTEM DESIGN

The system design illustrates how the system will fulfill the objectives or requirements identified during the system analysis. It serves as the overall plan or model that consists of the specification about the system - its form and structure, deliverables, and functional components. The system design is based on the requirement and it is aimed at meeting the specification of the studied system.

3.4.1 The Logical Design

The logical design lays out the various functional components and structures of the system and their relationship to one another. In describes inputs and outputs, processing functions to be performed, business procedures, data models and controls.

Apart from the portal data, the portal structure and deliverables required here are:

1. Authentication Subsystem;
2. Membership Management and User Profile Subsystem;
3. Portlet Management Subsystem
4. Content Management Subsystem;
5. Collaboration Subsystem;
6. Service Registry Management Subsystem.

These are conceptually represented in figure 3.5 below.

1. Authentication Subsystem

The authentication subsystem is designed to ensure that only valid users have access to the system. It is responsible for authentication and authorization of the various users. Every request is verified that the user is authorized to perform the operation. It therefore employs one or more of the following services to achieve this objective: *User Manager Service, Login & Logout Service, Proxy Manager Service, Role-based Access Control & Monitoring, Credential Management Service, etc.*

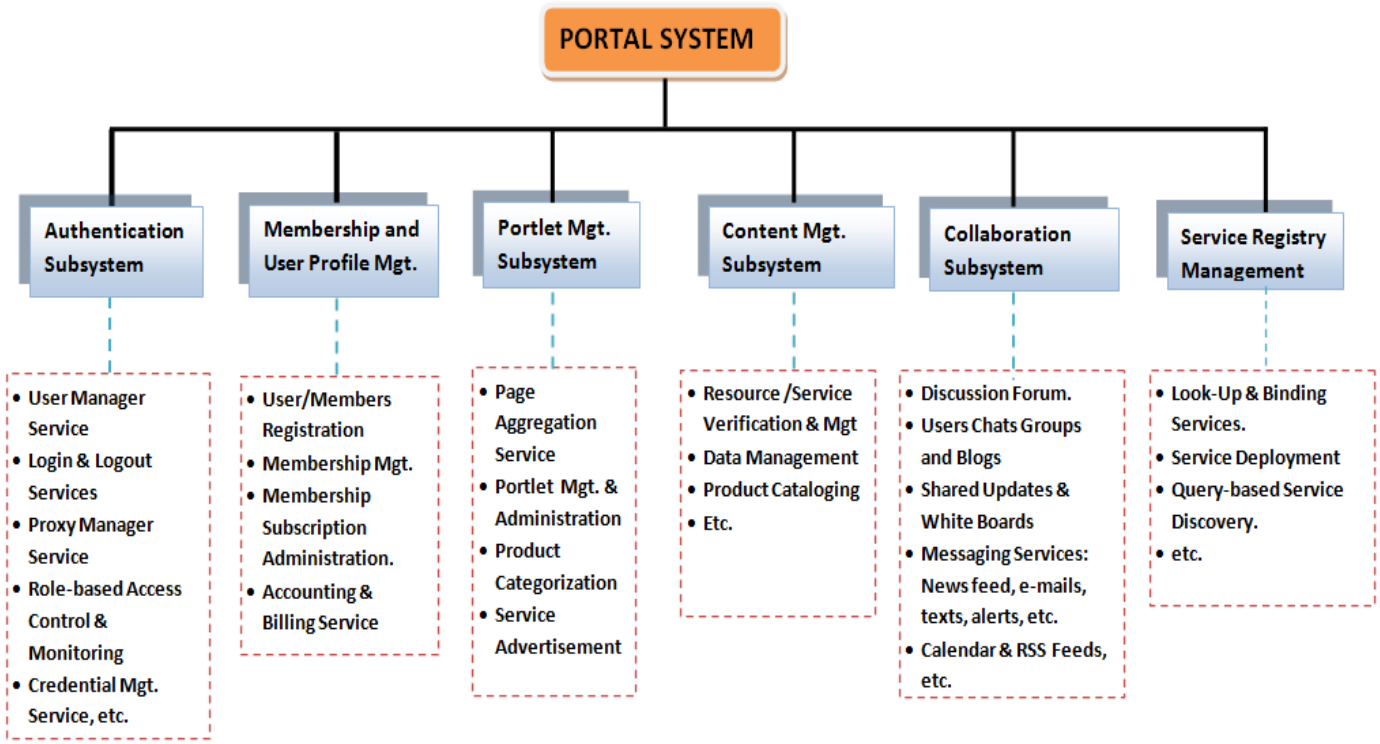


Figure 3.5: **The Conceptual View of GUISET Infrastructure portal**

2. Membership and User Profile Management

This subsystem is responsible for the management of users' registration, and profile management. It administers the members' subscriptions and services, accounting and billing. It employs one or more of the following services: *User Registration & Membership Management Service, Membership Subscription Administration, etc.*

3. Portlets Management Subsystem

This portlets management subsystem is responsible for the overall management of the various service portlets. It essentially houses the portlets containers and a portlet relies on its container for deployment, instantiation, initialization and destruction. A portal supports various portlet display modes such as: *View mode* [default], *Help mode, Edit mode* [setting changes] and *Configure mode* [administration]. It therefore also employs the one or more of the following services: *Portlets Management & Administration Service, Page Aggregation Service, Product/Service Categorization & advertisement.*

4. Content Management Subsystem

The content management system is responsible for the administration and management of the various portal contents – data, resources, services, products, etc. It is used by the tools by the tools or services to share resources with others. It employs one or more of the following services: Data Management, *Resource/Service Verification & Management, Product Cataloging, etc.*

5. Collaboration Subsystem

The collaboration subsystem is designed to enable coordinated interaction and collaboration amongst the various user communities. It entails some of the following services: *Discussion Forum, User Chat Rooms, Blogs and White Boards, Calendars, Messaging Services – shared Updates, News feed, e-mails, texts, alerts, etc.*

6. Service Registry Management Subsystem

There is a registry that serves as a service repository. This Subsystem is responsible for the administration and management of the service registry. It employs some of the following services: *Look-Up & Binding Service, Service Deployment, Query-based Service Discovery, etc.*

These various services are designed and encapsulated in different portlets and will be accessible through the various portlet interfaces, that is, each service might have a user interface which will be a portlet.

3.4.2 The Portal System Modeling

Modeling is the art of building an abstract representation of a concrete entity [90]. It involves the creation of a model of a real life entity, process or situation. A model is a graphical representation of the functionality, or behavior of the system. Systems model play important role in system development [90]. It helps to give a pictorial representation of reality with respect to functionality, or behavior of the system that will satisfy the needs of clients or users.

Modelling is an activity carried out with the aim of producing a correct, complete and consistent representation of the real world – or more precisely that part of the real world which is of interest to the designer of the target Information System, IS. A formal model of the portal system is built using the Unified Modeling Language, UML. UML is a modeling paradigm that provides a set of conventions and tools. These tools are used to describe a system in terms of its component (object) structures and the various interactions within the system and with the external system [90].

The portal system is modeled using these available tools listed below:

1. Use case Diagram
2. Sequence Diagram
3. Collaboration Diagram
4. Class Diagram
5. Activity Diagram

3.4.2.1 Use Case Diagrams

A use case diagram graphically depicts the interactions between the system, the external system and the client or user [90]. Use case diagrams play major roles in system design because they act as roadmaps in constructing the structures of the system; they also define who use the system and in what way is the clients expected to interact with the system.

1. The Authentication Subsystem.

The authentication subsystem is designed to ensure that only valid users have access to the system. The system administrator is responsible for authentication and authorization of the various users based on stipulated policies such as role-based access control (RBAC). The administrator also manages the various clients' access credentials alongside their logs and session. This is illustrated in figure 3.6 below.

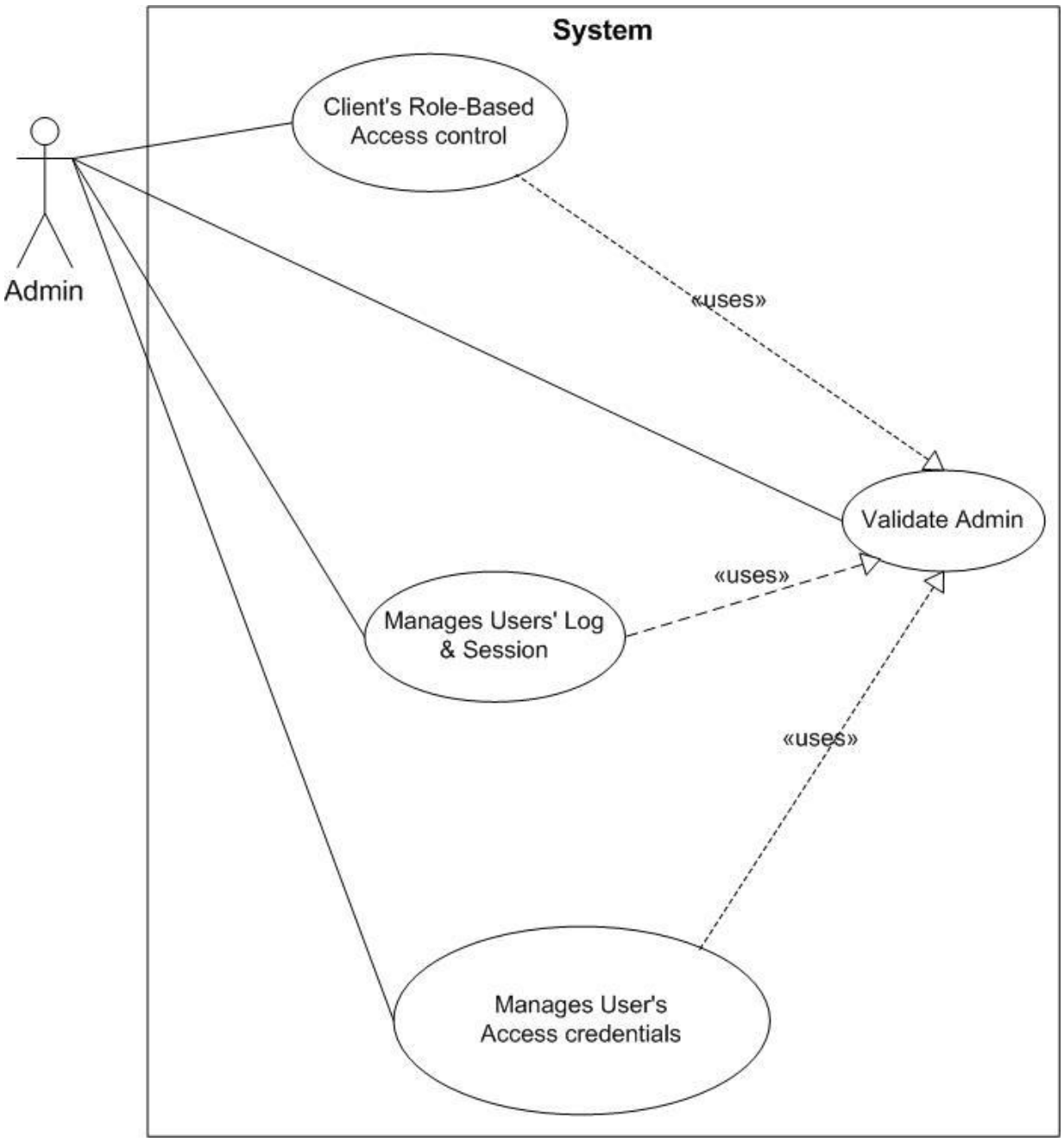


Figure 3.6: **Authentication Subsystem**

2. Membership and User Profile Management

This subsystem is responsible for the management of users' registration, and profile management. A valid client can setup a subscription request after viewing and selecting an available service. The administrator validates the client's registration and subscription request after ensuring that the appropriate requirements have been met. This is illustrated in figure 3.7 below.

Figure 3.7: **Membership and User Profile Management**

3. Portlets Management Subsystem

This portlets management subsystem is responsible for the entire management of the various service portlets. It essentially houses the portlets containers and a portlet relies on its container for deployment, instantiation, initialization and destruction. A valid client can subscribe for a desired available service as well as a service provider advertise services. The administrator however manages the various service portlets and ensures the proper administration of the client's service subscription. This is illustrated in figure 3.8 below.

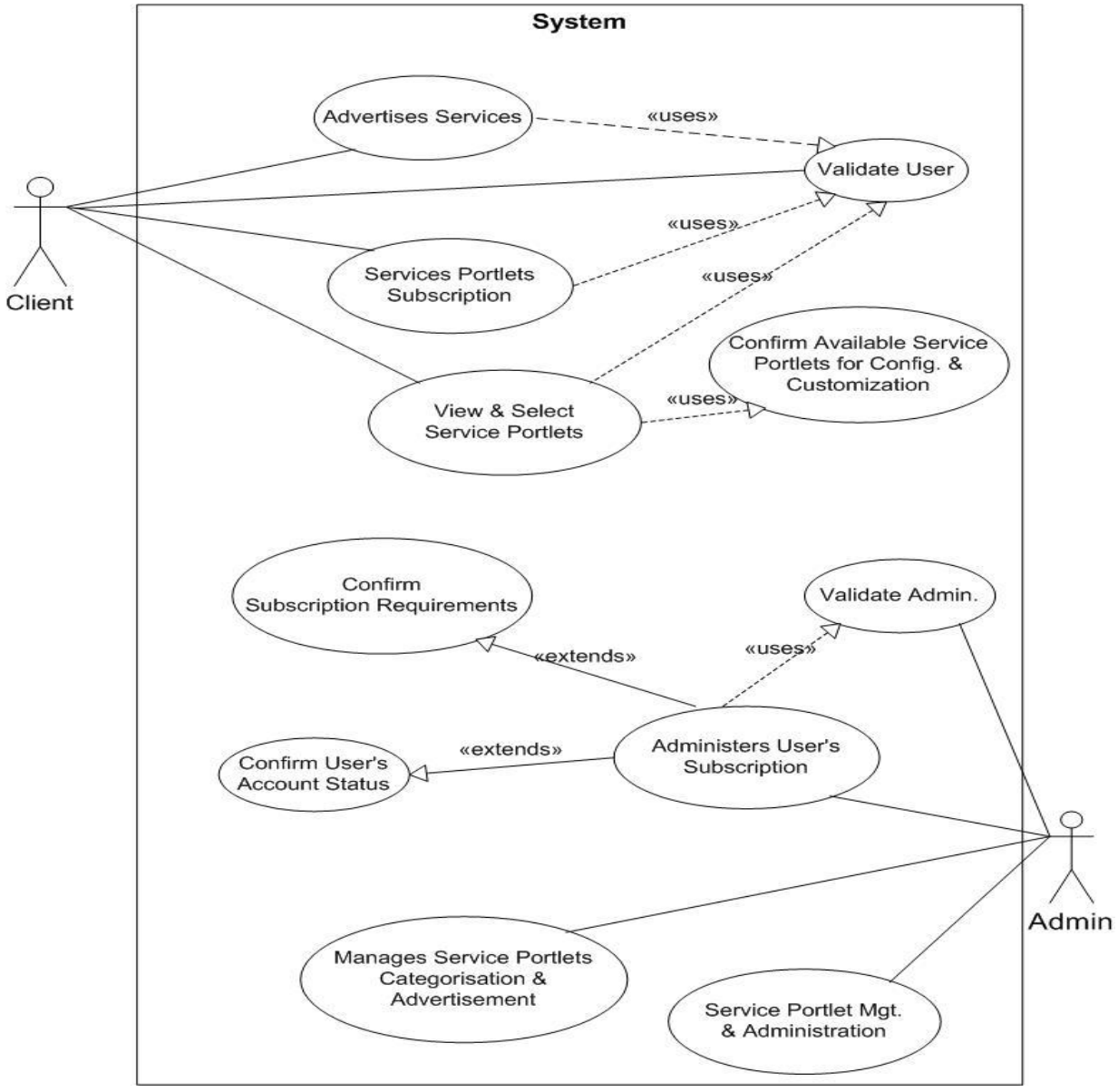


Figure 3.8: **Portlets Management Subsystem**

4. Content Management Subsystem

The content management system is responsible for the administration and management of the various portal contents – data, resources, services, products, etc. A valid client can upload and advertise services for use. The administrator verifies and manages these services for advertisement. He does the service cataloguing and data management. This is illustrated in figure 3.9 below.

Figure 3.9: **Content Management Subsystem**

5. Collaboration Subsystem

The collaboration subsystem is designed to enable coordinated interaction and collaboration amongst the various user communities. A valid user can join an existing user community based on interest or setup a new one, creating a user forum for similar business interest discussion. This is illustrated in figure 3.10 below.

Figure 3.10: **Collaboration Subsystem**

6. Service Registry Management Subsystem

There is a registry that serves as a service repository. A valid client can engage in a service look-up and binding based on the available service description in the registry. The administrator manages the service registry and periodically updates its content based on the client's requests and search. This is illustrated in figure 3.11 below.

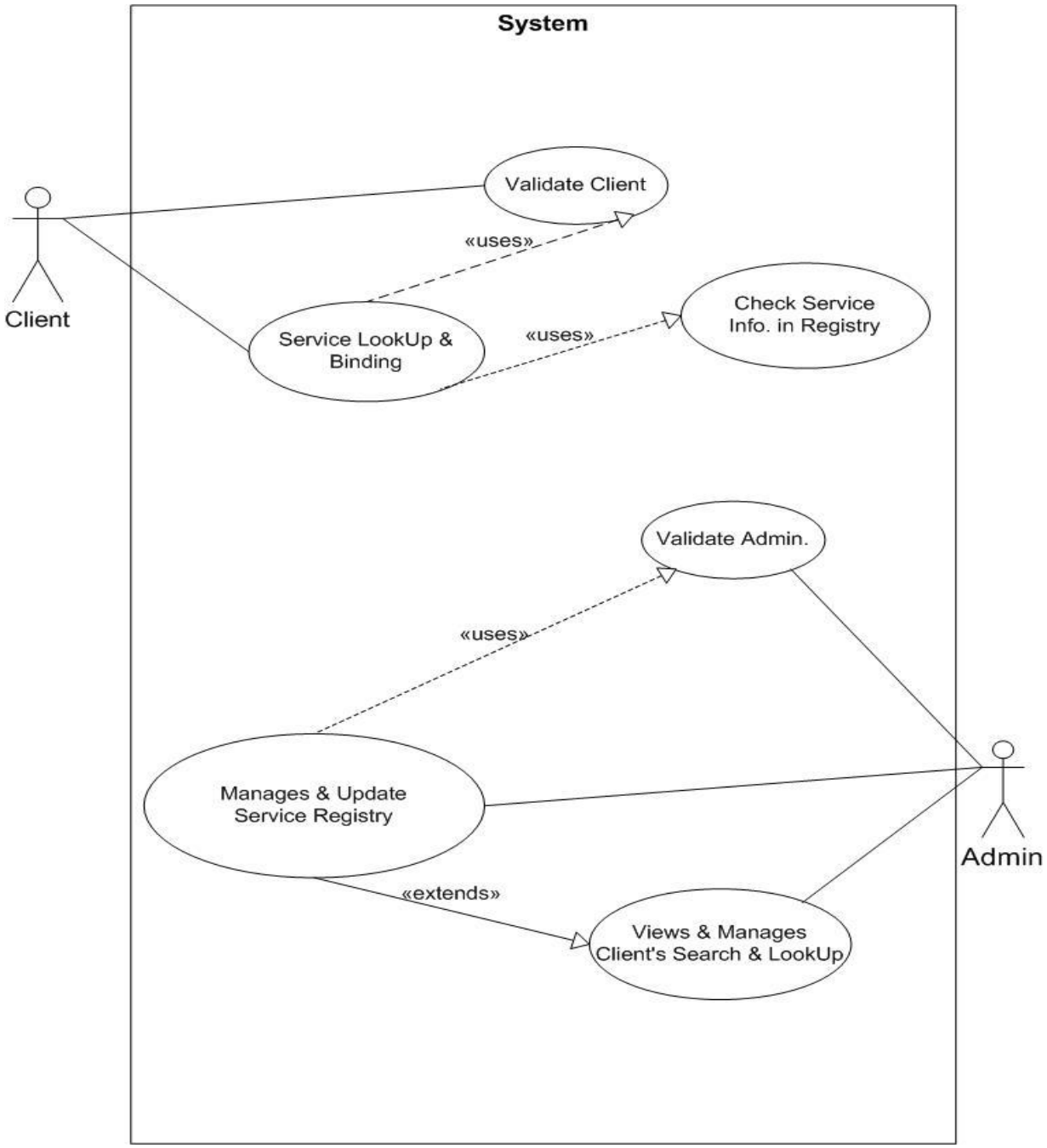


Figure 3.11: **Service Registry Management Subsystem**

3.4.2.2 Sequence Diagram

The sequence captures interaction among the various entities (actors) with the system. This is illustrated as follows: It begins with portlet registration by the portal administrator normally through a portal integration development environment, IDE e.g. WebSphere [51], and ends up with a portal being registered to a given portlet producer. The Figure 3.12 outlines the protocol.

First, an introductory description of the producer is obtained through the "*getServiceDescription( )"* function. If registration is required then, the consumer must register with a producer before accessing any of the producer's portlets. Once registered, the consumer queries again the producer but now, a detailed description of the available portlets is returned. With all this information, the portal IDE creates a WSRP consumer. This WSRP consumer is within the portal realm.

Once registered, the portal is ready to engage the portlet in conversation to deliver its service. This is achieved through a two-step protocol as shown in figure 3.12 below. To begin with, the very first markup realizing the service is obtained through the *"getMarkup( )"* function. The returned markup is aggregated to other markup that built up the portal page which is finally rendered to the end user. Whenever the user clicks on a link of the portlet markup, the portal receives the HTTP request which is in turn, forwarded to the portlet producer by means of the *"performBlockingInteraction( )"* function till it finally reaches the portlet itself. As a result, the portlet can change its state. But no markup is returned to the consumer. This requires the consumer to issue a *"getMarkup( )"* function to recover the eventually new markup associated with this new state.

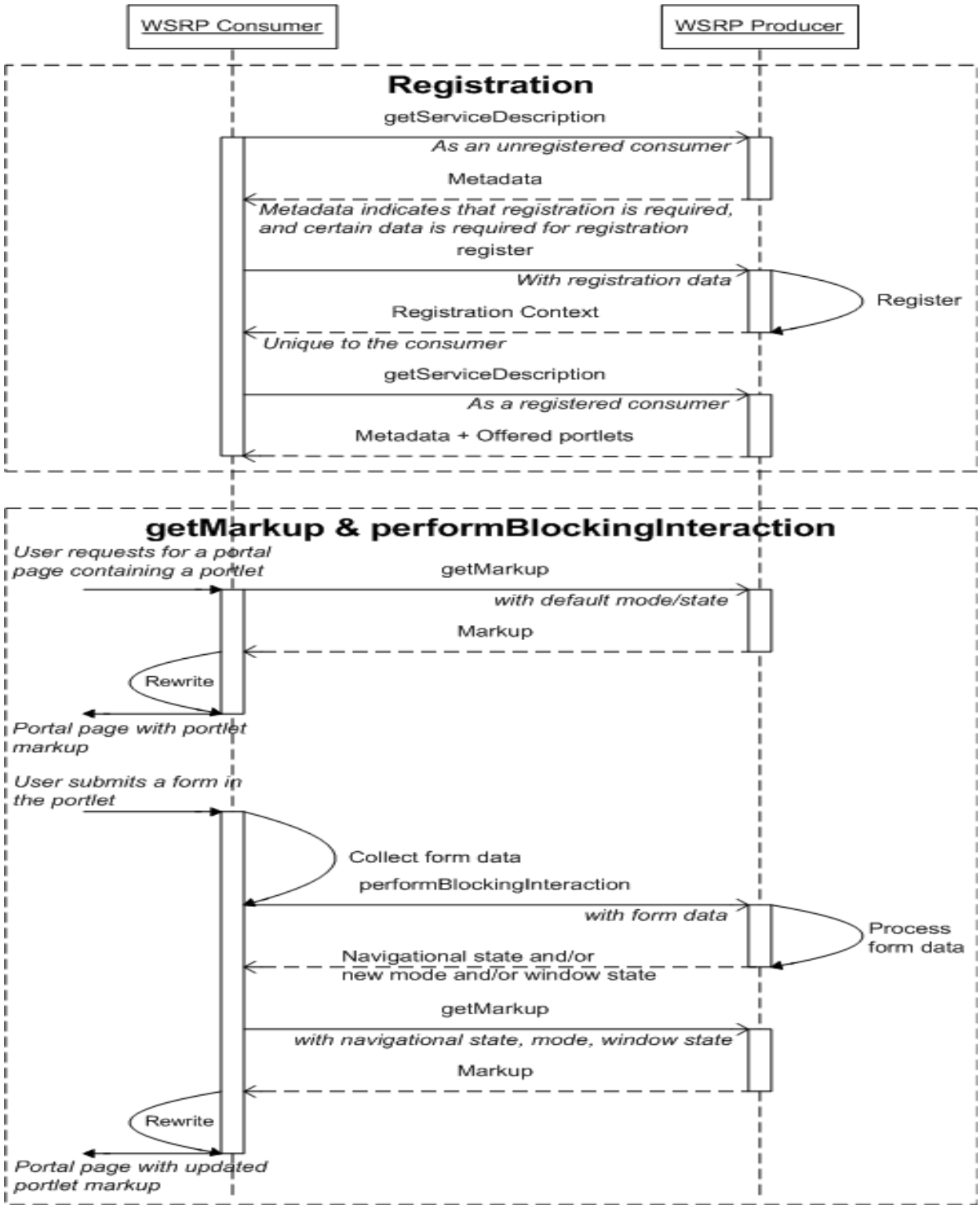


Figure 3.12: **Sequence Diagram (WSRP Protocol)** [28]

3.3.2.3 Activity Diagram

The activity diagram depicts the workflow of activities within the system. It graphically represents the flow of performance of various actions by the system entities. The flow of the system activities is summarized in figure 3.13 below.

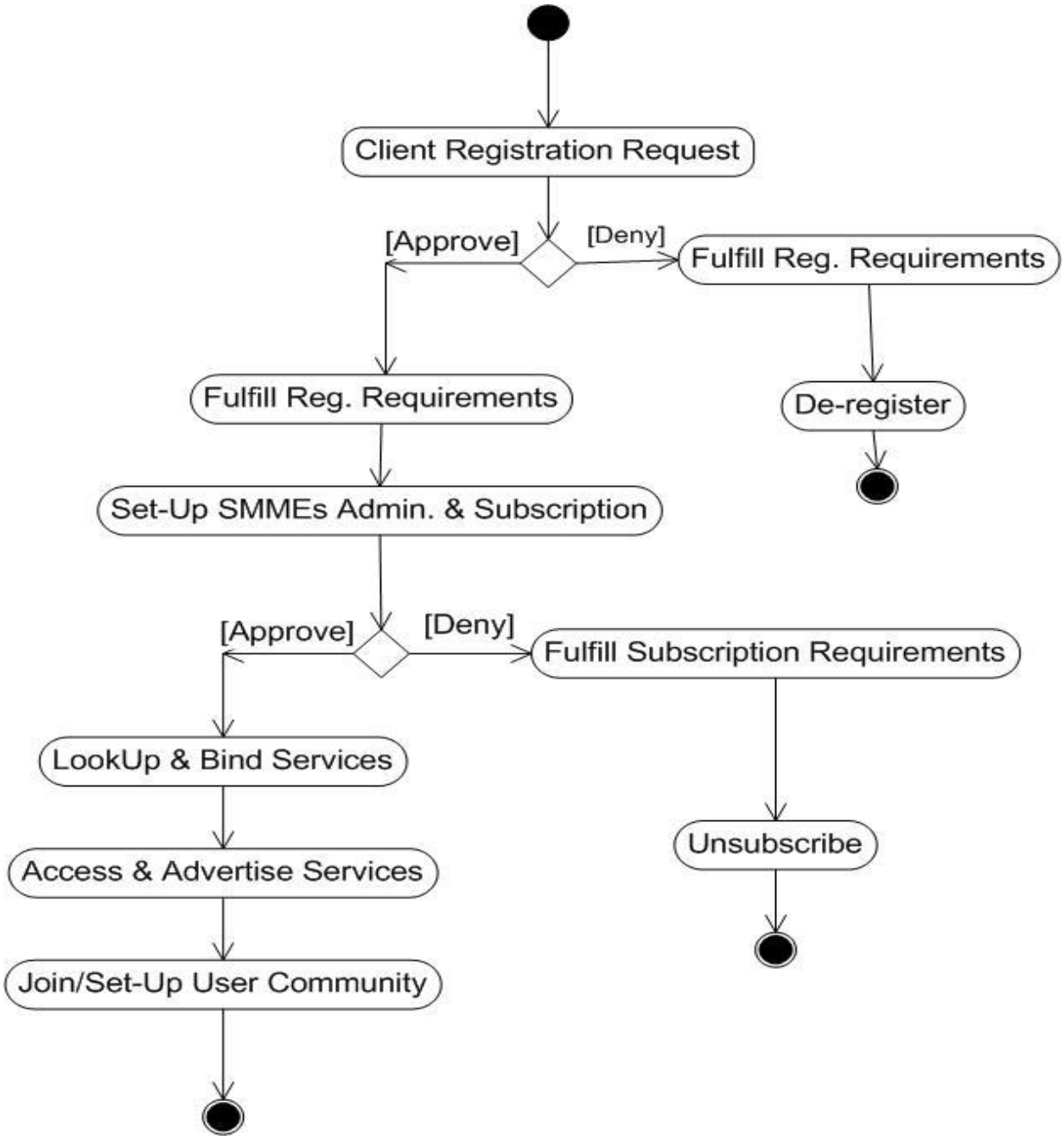


Figure 3.13: **Activity Diagram**

3.3.2.4 Collaboration Diagram

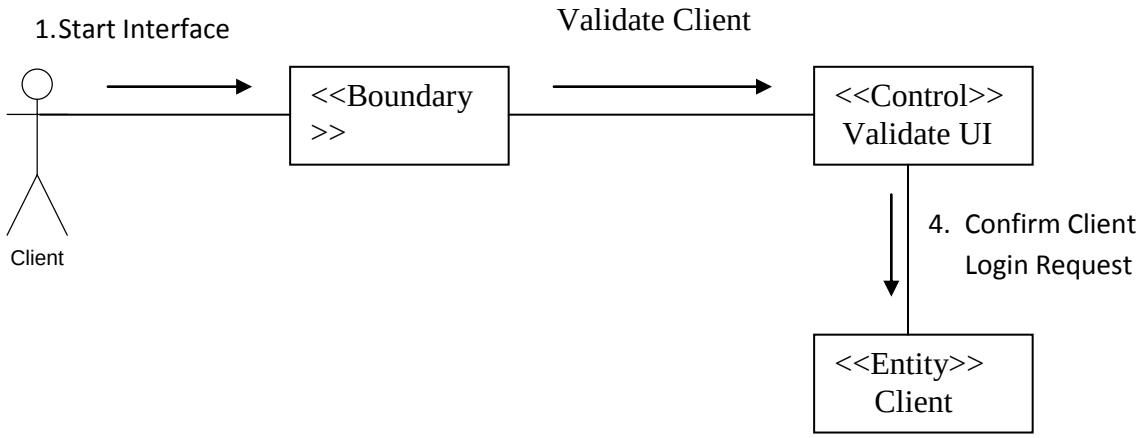


Figure 3.14a: **Validate Subscriber's Login Collaboration Diagram**

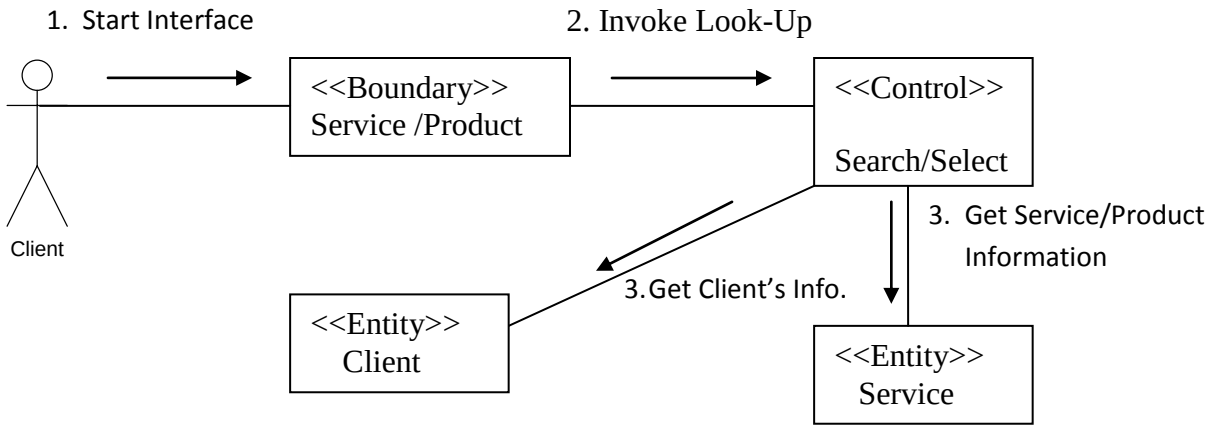


Figure 3.14b: **Service/Product Lookup Collaboration Diagram**

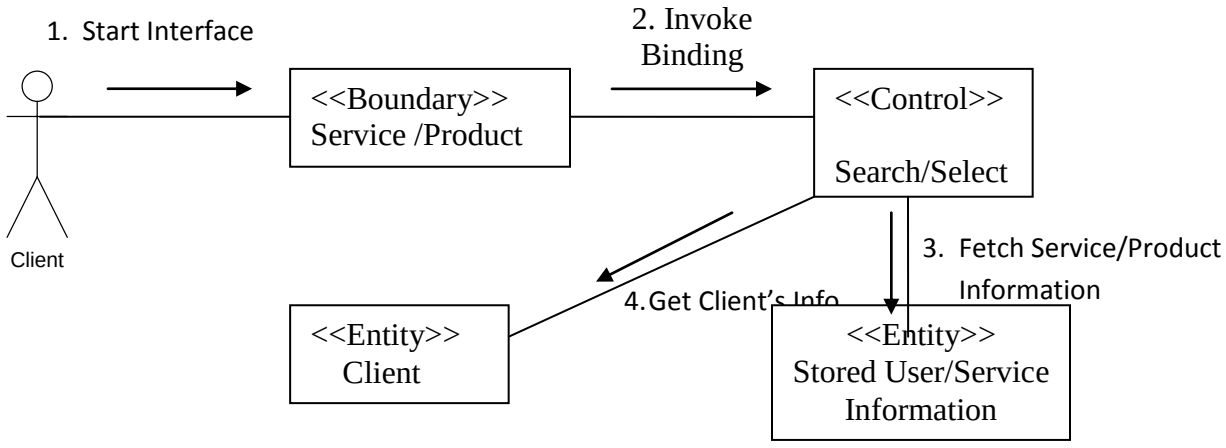


Figure 3.14c: **Service/Product Request Collaboration Diagram**

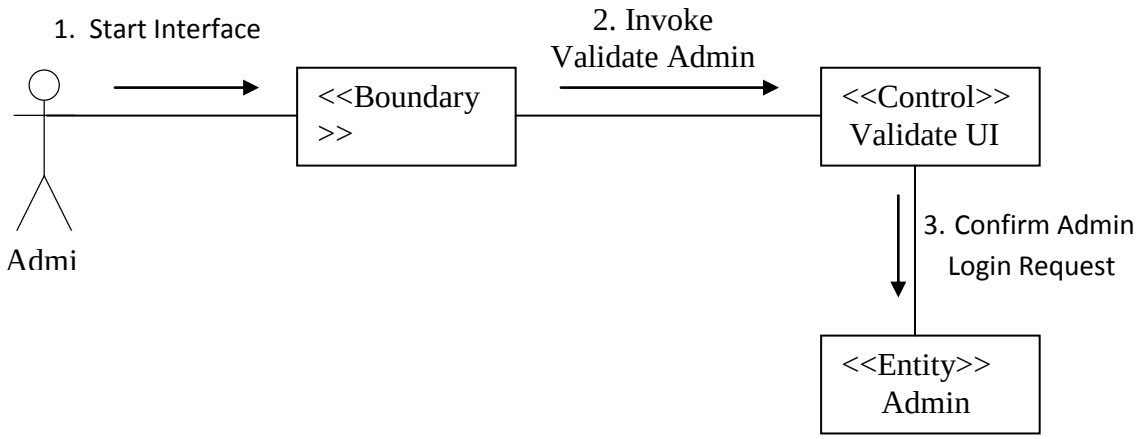


Figure 3.14d**: Validate Admin Login Collaboration Diagram**

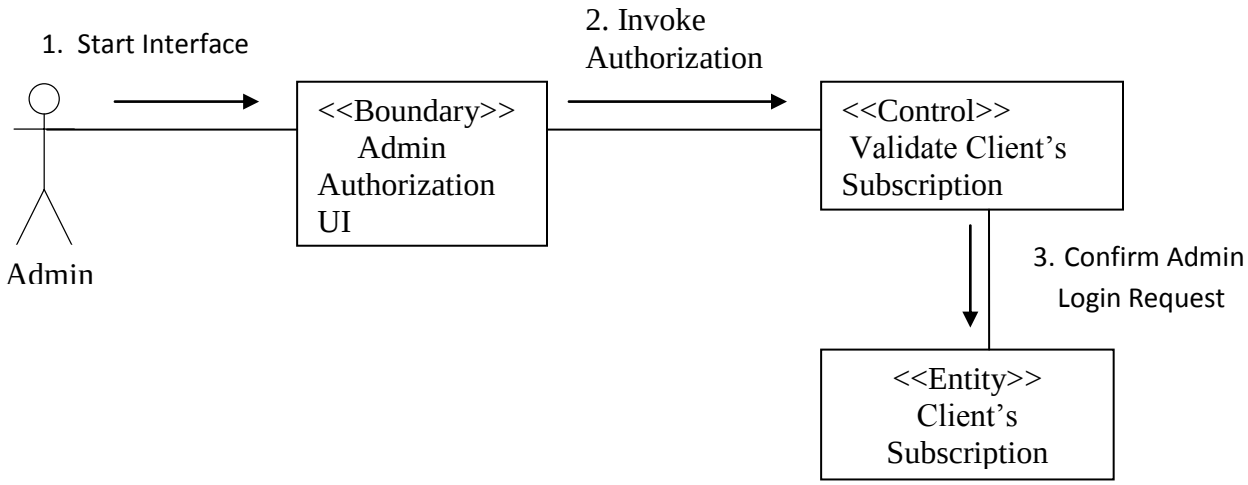


Figure 3.14e: **Admin Authorization Collaboration Diagram**

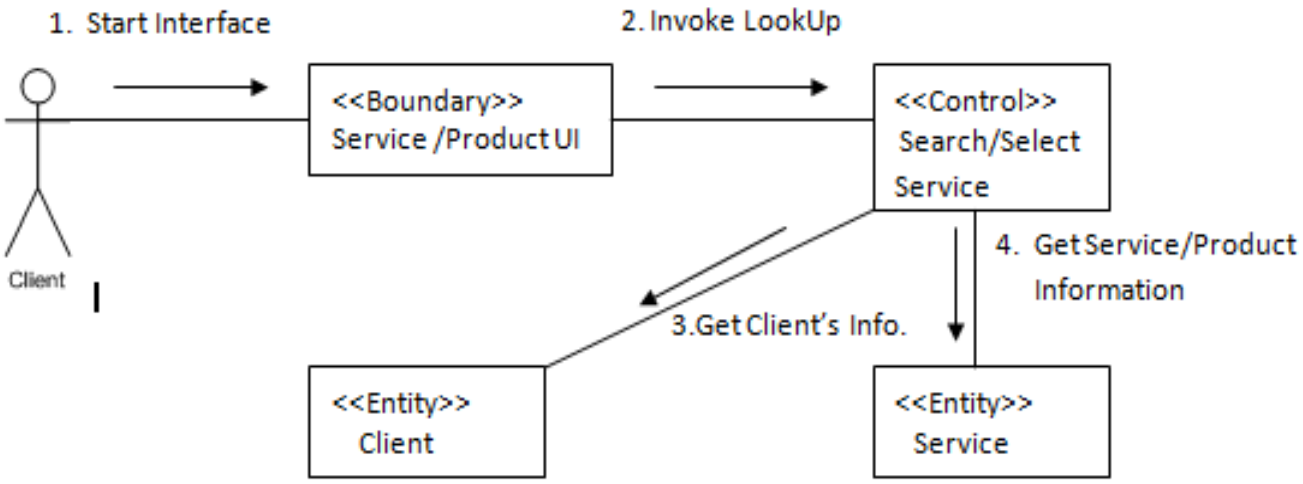


Figure 3.14f: **Service/Product Look-Up Collaboration Diagram**

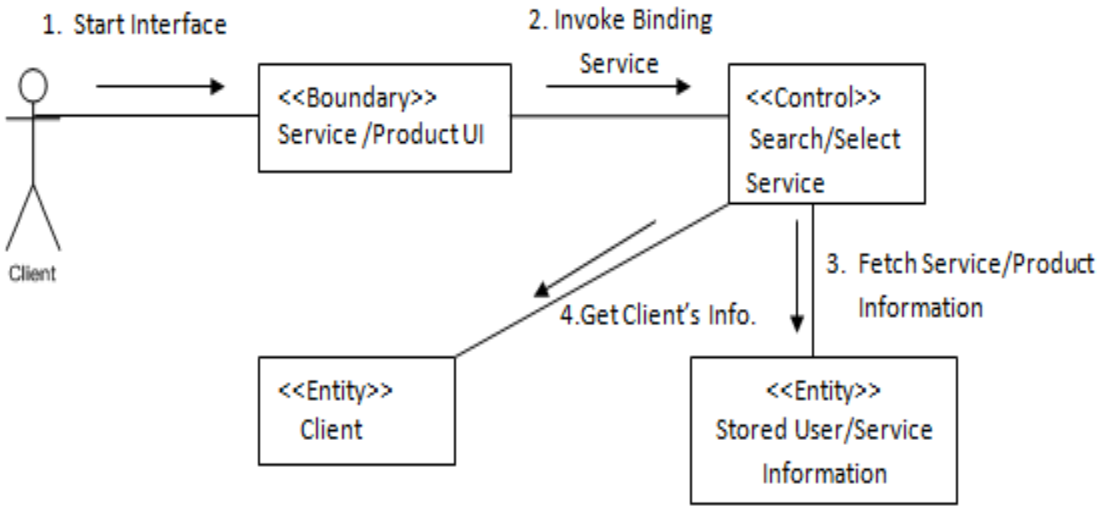


Figure 3.14g: **Service/Product Request Collaboration Diagram**

3.3.2.5 Entity Class Diagram

The entity class diagram depicts the relationships among the various entity classes. This is illustrated in figure 3.15 below.

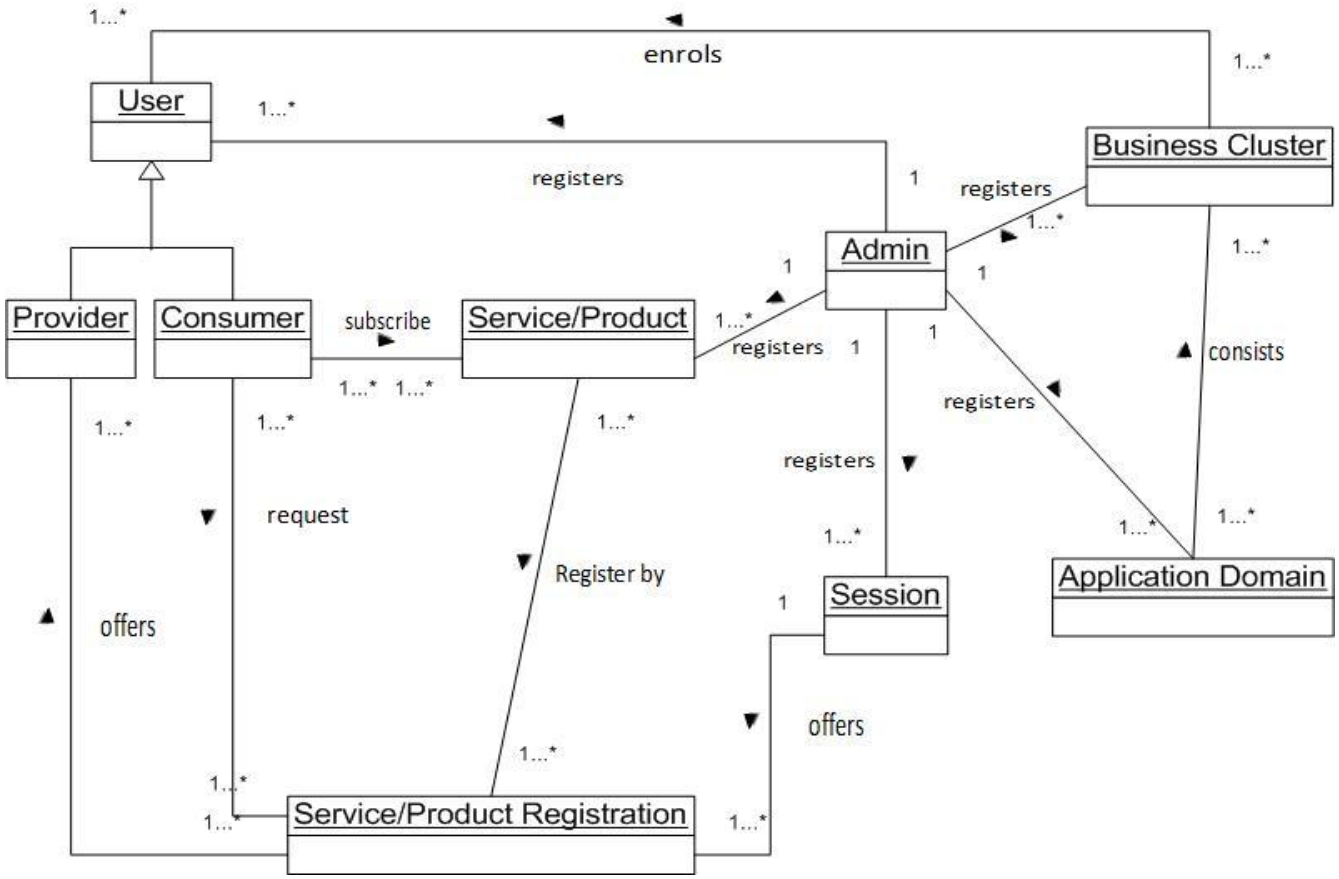


Figure 3.15: **Entity Class Diagram**

3.4 THE USER INTERFACE DESIGNS

A Graphical User Interface (GUI) is a medium through which the system users interact with the system. It is a medium through which the system users send in their request into the system, displays the outputs to the users alongside various options available to the user.

Some of the various interfaces showing the different aspects of the system are as follows:

1. The GUISET Portal Home Page.
2. The Create User Account Interface.
3. The Portal Administrator's Registration Interface.
4. The Secured User Login Interface (User Authentication).
5. The Administrator's Home Page.

3.4.1 The GUISET Portal Home Page

The GUISET portal home page is the first user interface displays at logon to both the existing users and the guest users. It presents the general overview of the portal system, showing the various tabs, links, and login section; create user account link, etc. The GUISET Portal home page is shown in figure 3.16 below.

3.4.2 The Create User Account Interface

Every prospective user that is not registered yet are redirected to the create user account page from the home page where the user's details are supplied to the system in order to create a new user account. The create user account interface is shown in figure 3.17 below.

3.4.3 The Portal Administrator's Registration Interface

The portal administrator's registration interface is the medium through which the system administrator is first registered with the system before he can assume the responsibility of the overall administration and management of the entire portal system. The administrator's registration interface is shown in figure 3.18 below.

Figure 3.16: **The GUISET Portal Home Page**

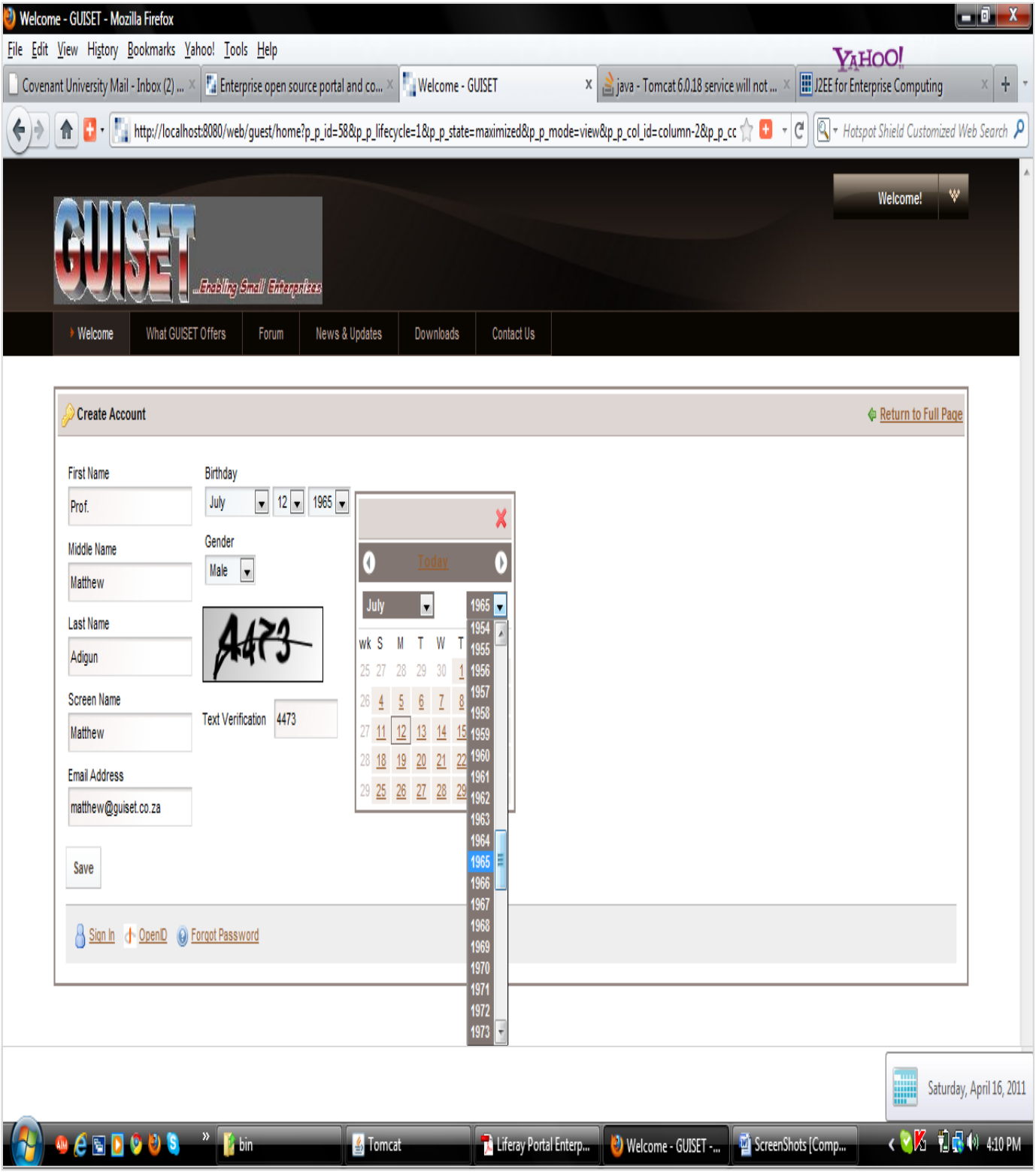


Figure 3.17: **The Create User Account Interface**

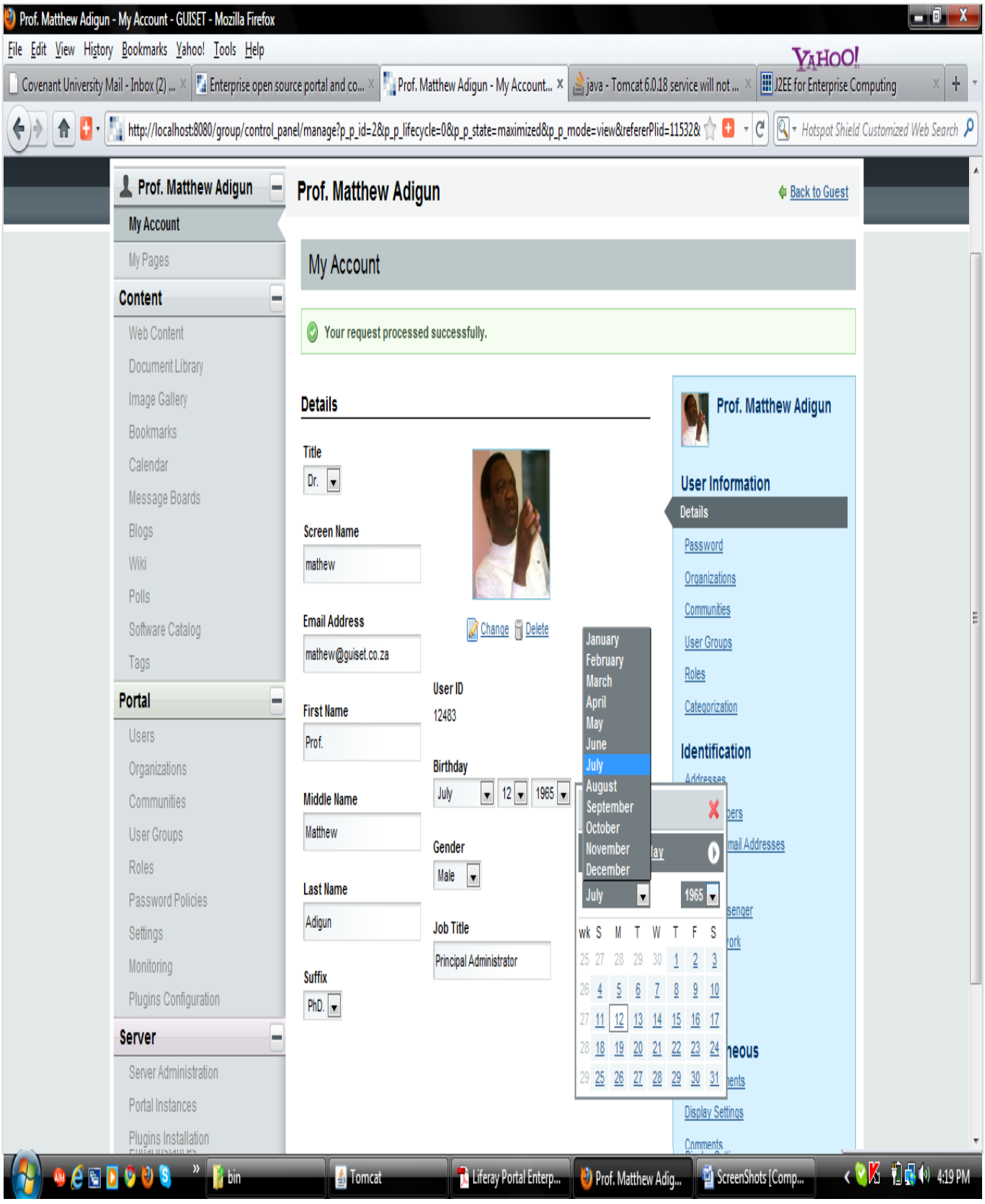


Figure 3.18: **The Portal Administrator's Registration Interface**

3.4.4 The User Authentication Interface

The user authentication interface is a medium of achieving secured user login. The user (administrator, registered users, etc) is expected to login in with valid email address and password. In a case where the email address or password supplied by the user, the system issues a login error message. This is as shown in figure 3.19 below.

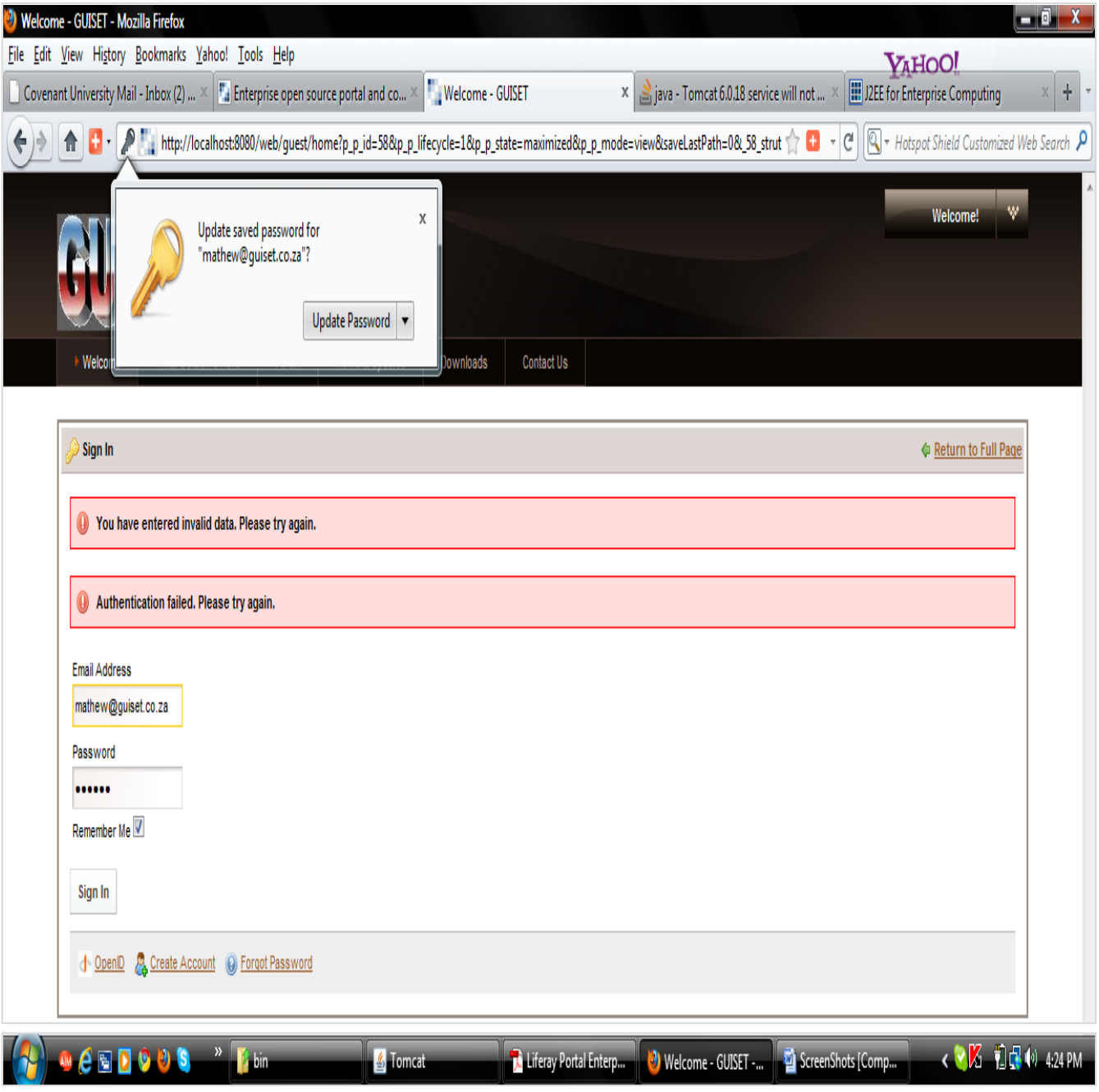


Figure 3.19: **The User Authentication Interface**

3.4.5 The Administrator's Home Page

The administrator's home page is the user interface the administrator is taken to after he has successfully login into the system. The administrator is welcome to the home page from where he has access to several options through the various tabs and links. The administrator's home page is shown in figure 3.20 below.

Figure 3.20: **The Administrator's Home Page**

CHAPTER FOUR

4.0 SYSTEM IMPLEMENTATION

4.1 THE GRID-BASED PORTAL DEVELOPMENT TOOLS USED

A prototype of the portal system based on the proposed framework is built on Liferay 5.2.3 portal tool kit [10, 25, 26] bundled with Tomcat 6.0.18. Liferay is more than just a portal container; which comes with lot of helpful features like Content Management System (CMS) [91], WSRP compliant producer and consumer, Single Sign-On (SSO), support for Aspect-Oriented Programming (AOP) [61, 64], and many other latest technologies. Liferay has a very clean architectural design based on best practices of J2EE, which allows it to be used with a variety of containers ranging from lightweight servlets containers like Tomcat and Jetty, to fully fledged J2EE-compliant servers like Borland ES, JBoss, JOnAS, JRun, Oracle9iAS, Orion, Pramati, RexIP, Sun JSAS, WebLogic, and WebSphere [26, 51].

The Flexibility in its design allows implementation of business logic in any suitable and appropriate technology like Struts [92], EJB [19] etc., which in turn can be based on Hibernate [93], Java Messaging Service (JMS) [94], JavaMail and Web Services. Liferay makes it possible to give Portal Presentation to any type of Java application with no or minimum changes. The Customization of portlets and portal pages in Liferay and the layout management are very easy. Liferay Portal has a Web-based Graphical User Interface for user interaction to design the layout of Portal Pages without modifying any configuration files, which is similar to Stringbeans [10, 25]. Liferay Portal Enterprise comes with many useful portlets, and in fact Liferay portal has maximum utility portlets as compared to other open source Portal Frameworks, which are JSR 168 compliant and can be used in any portal framework with little changes.

Liferay supports WSRP specification as long as both WSRP consumer and WSRP producer are a Liferay portal instance and like most of the other Portal Frameworks, Liferay uses a default database, Hypersonic 1.7, which is fine for development purposes. Liferay can also be used with any database with minimum efforts due to the use of Hibernate [93] in its design. Liferay has JSP portal tag libraries and lot of utility classes in different packages to assist programmer in developing the portlets/portals.

4.2 THE PORTAL PROTOTYPE AND USER INTERFACES

The portal prototype is a proof of concept built by using Liferay 5.2.3 portal tool kit and some the various GUIs through which different users interact with the system are highlighted below.

4.2.1 The Enterprise Portal Configuration Panel Interface

The portal configuration panel is a medium through which the GUISET portal parameters are set. Some of these parameters include authentication parameters such as SSO, email parameters, default user credentials and associations, portal display parameters, etc. This is shown in figure 4.1 below.

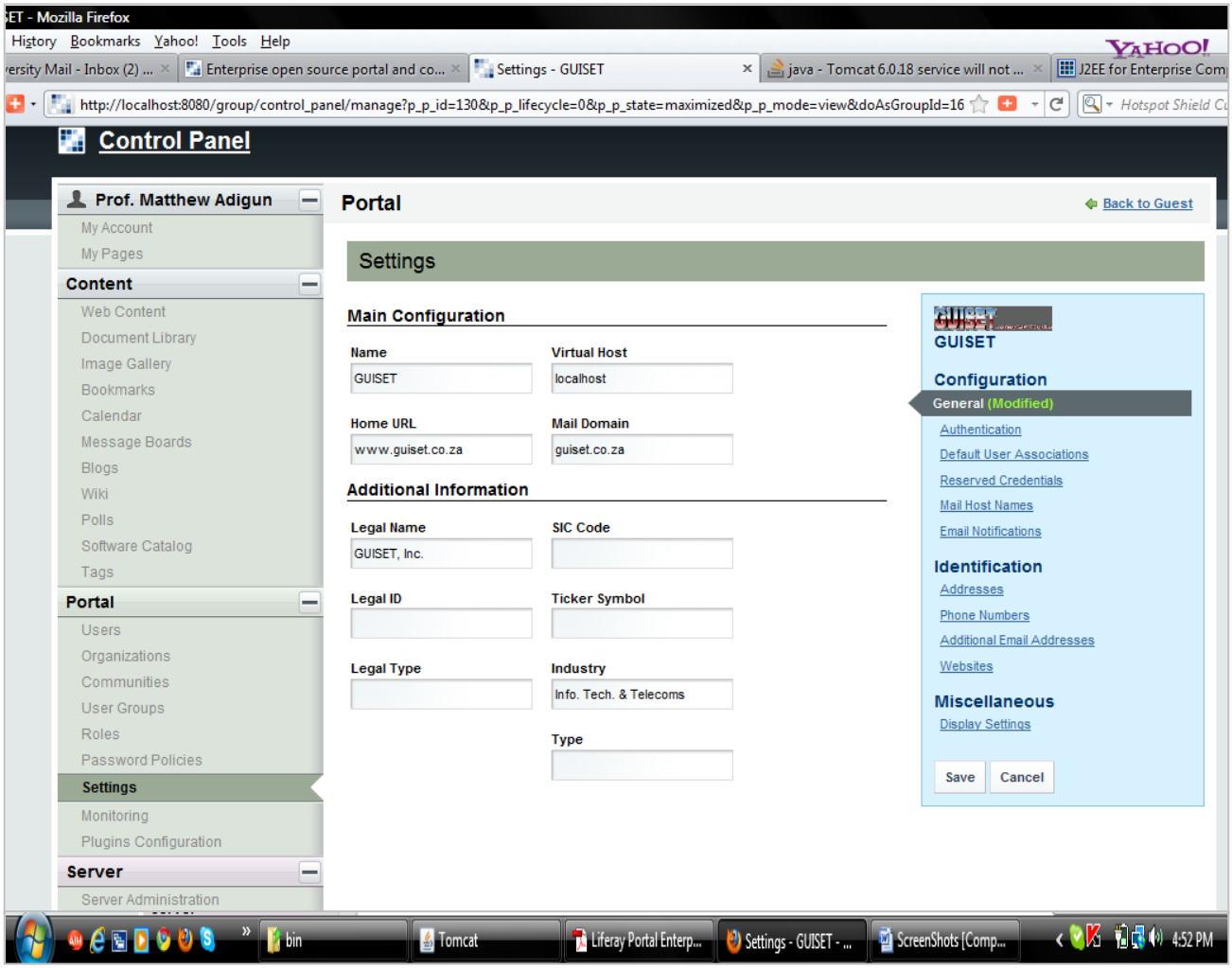


Figure 4.1: **The Enterprise Portal Configuration Panel Interface**

4.2.2 The Portal Authentication Setting Interface

The portal authentication parameters are set through this interface. These parameters include: Lightweight Directory Access Protocol (LDAP), Central Authentication Service (CAS), Open Single Sign-On (SSO), etc. This is shown in the figure 4.2 below.

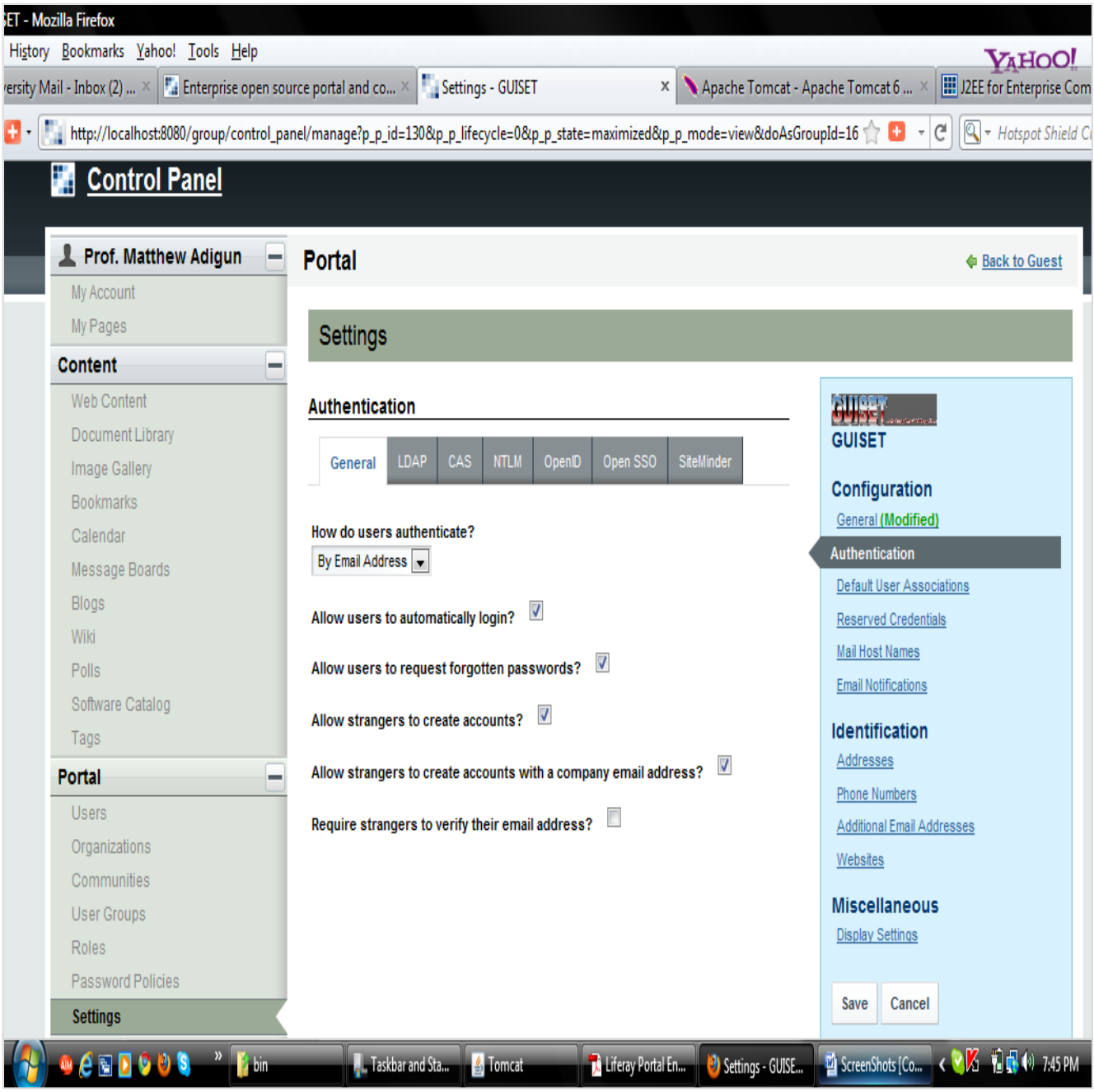


Figure 4.2: **The Portal Authentication Setting Interface**

4.2.3 The Portal Single Sign-On (SSO) Setting Interface

This presents the setting of one of the authentication parameters, the Open SSO. The associated parameters are set as shown in figure 4.3 below.

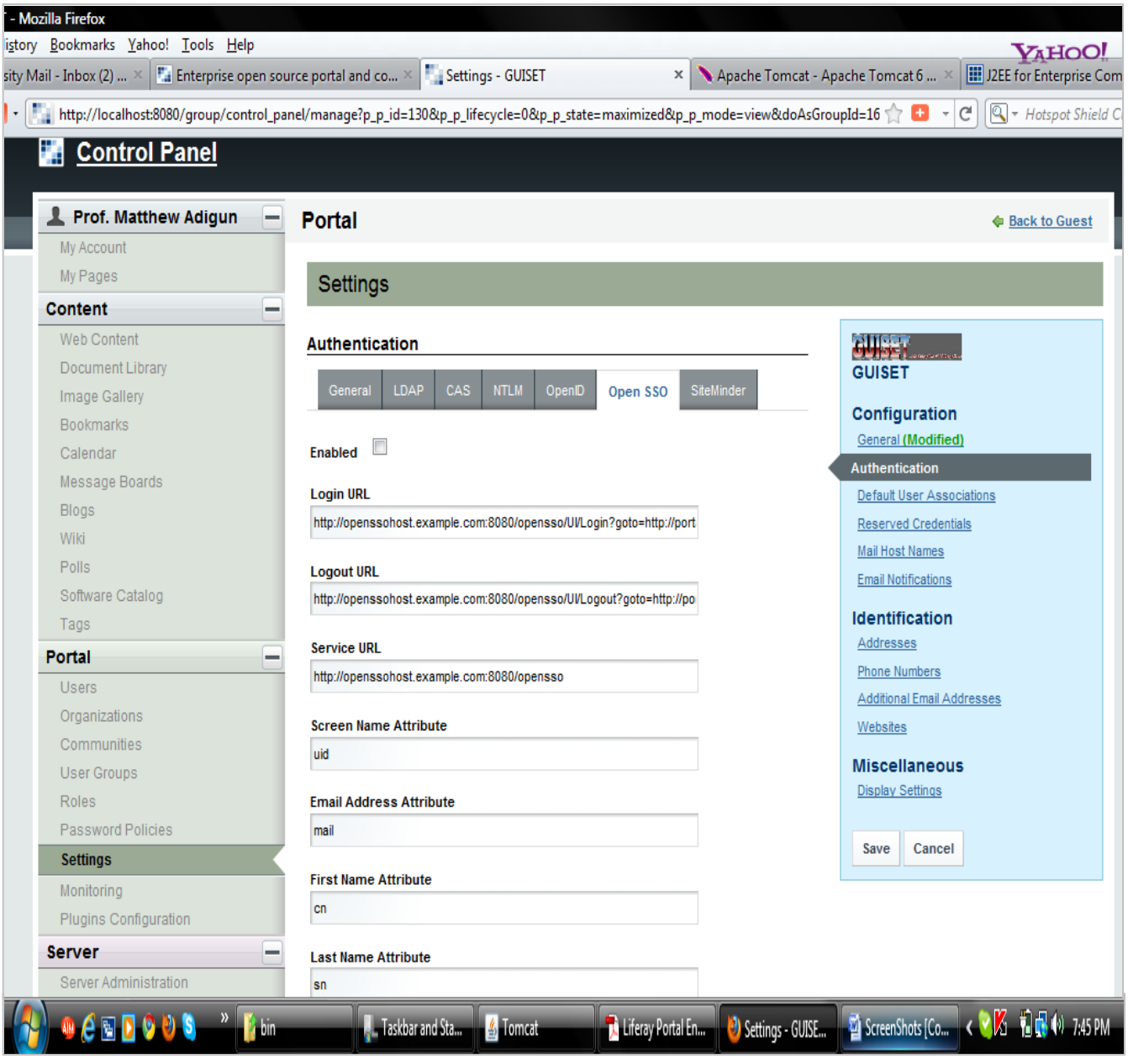


Figure 4.3: **The Portal Single Sign-On Setting Interface**

4.2.4 The Create Membership Account Interface

The create membership account interface presents a new prospective member the medium through which a new membership account can be create. The new user is expected to supply valid data in order to have an account opened. This is shown in figure 4.4 below.

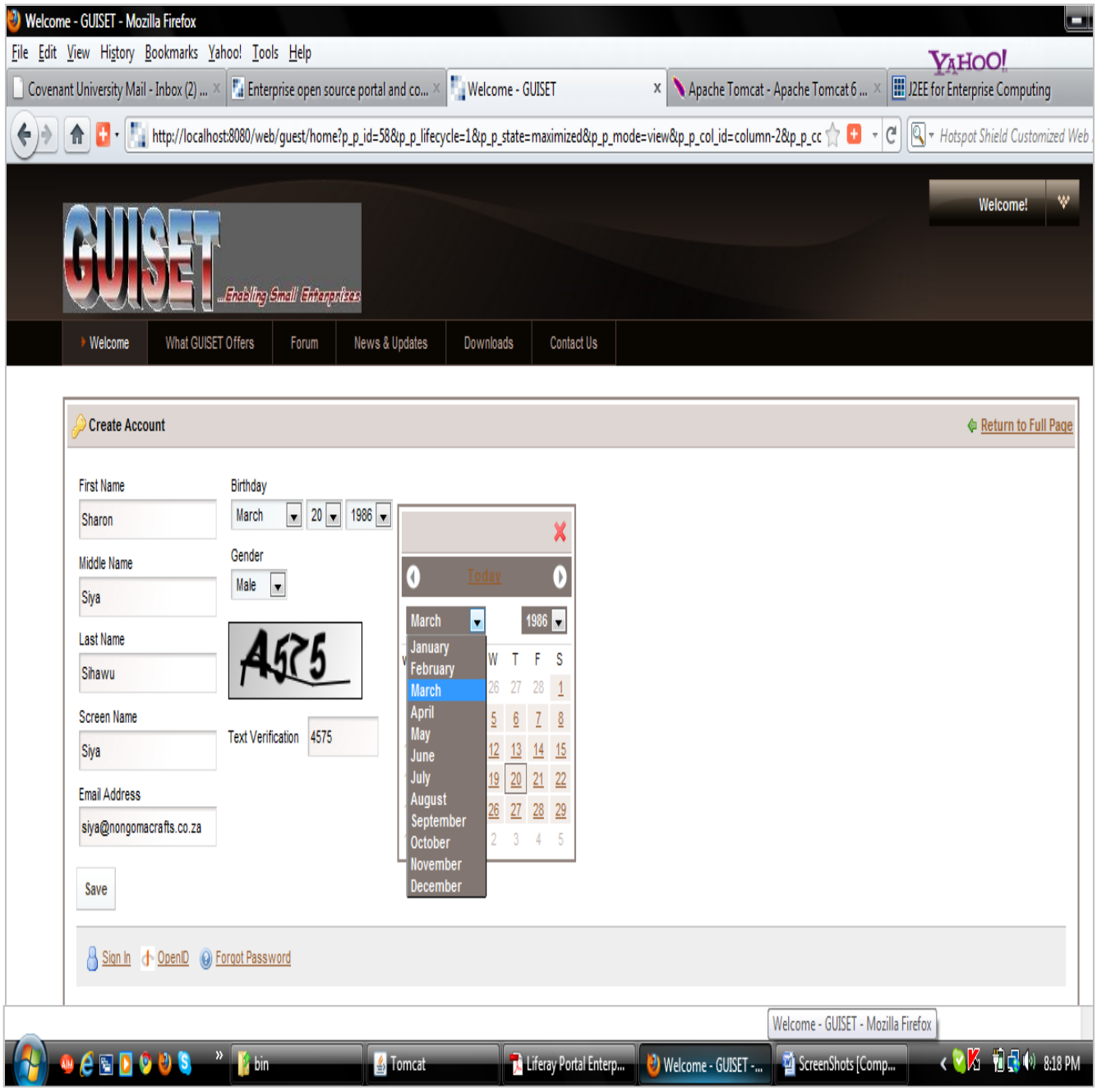


Figure 4.4: **The Create Membership Account Interface**

4.2.5 The User Membership Registration Interface

The new user after successfully creating a membership account can then loon to the portal from where the detailed membership registration of the user can be completed. The various options available to the user through this interface are shown in figure 4.5 below.

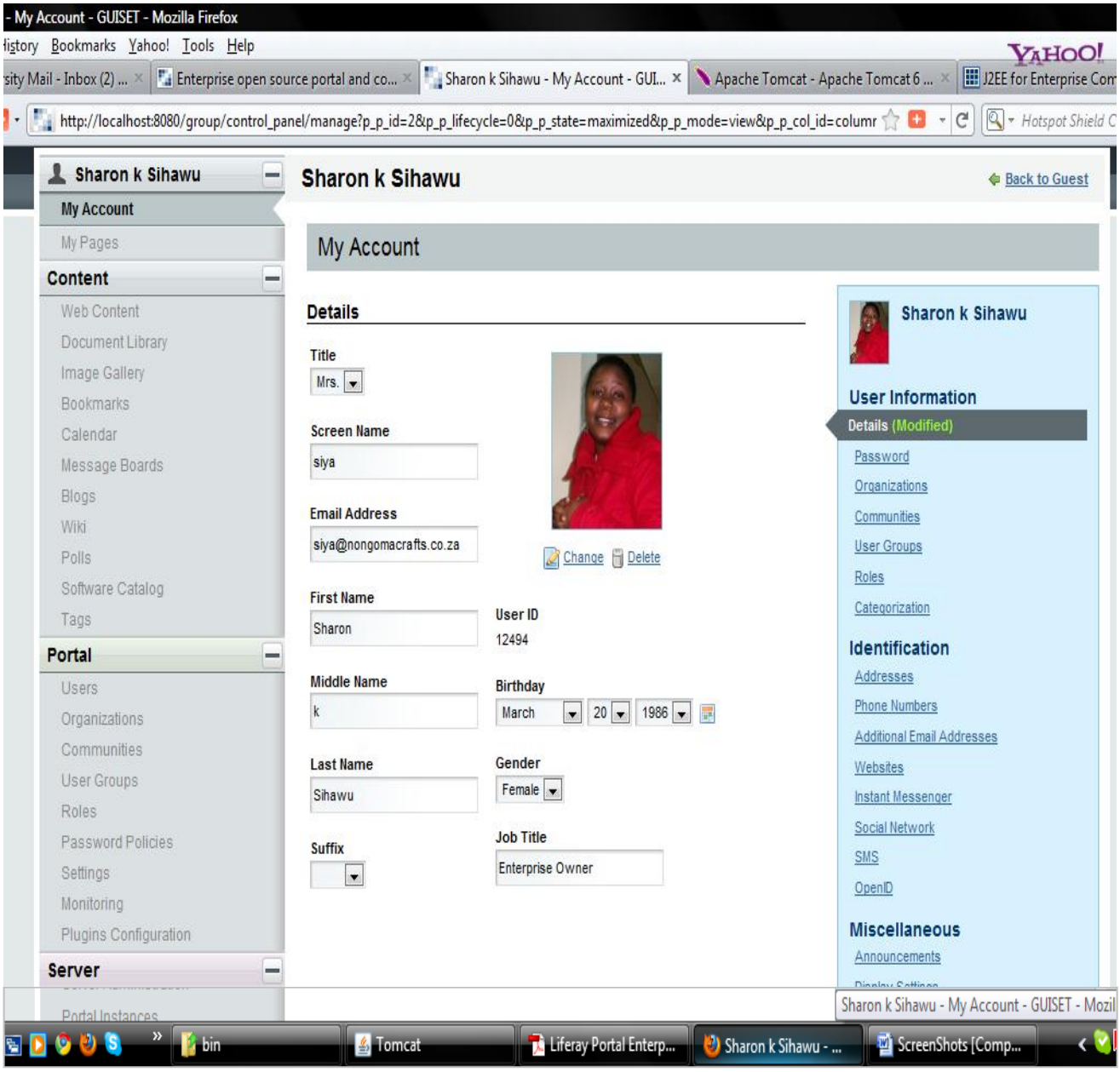


Figure 4.5: **The User Membership Registration Interface**

4.3 THE SYSTEM REQUIREMENTS

The system requirements consist of the various tools required from the point of design and development to the eventual deployment of the portal system. These requirement are presented in tabular form in the tables below.

TABLE 4.1: **The Software Requirements**

Requirements	***Software***
Operating System	Microsoft Windows XP, Windows Vista, etc.
Grid-enabled Portal Tool Kits	Liferay 5.2.3 bundled with Tomcat 6.0.18
Database Management System	Liferay Default Database, Hypersonic 1.7
Model Design Tools (UML Modeling)	Microsoft Office Visio 2007
Underlying Grid System	Globus Tool kits 4.0

TABLE 4.2: **The Web Client Software Requirements**

Requirements	***Software***
Operating System	Microsoft Windows XP, Vista, etc.
Internet Browser	Internet Explorer 6+; Mozilla Firefox 4;

4.4 THE HARDWARE REQUIREMENTS

The Hardware requirements are also presented in tabular form in the table 4.3 below.

TABLE 4.3: **The Hardware Requirements**

Minimum Requirement
Pentium IV, 2.5GHz, CPU
Minimum of 1GB, Random Access Memory (RAM)
Minimum 14” Color Monitor
Minimum 32 Bit Video Graphics Adapter (VGA)
Minimum 32 Bit Sound Card
Modem or Ethernet Card
Keyboard and Mouse
Uninterruptible Power Supply (UPS)

4.5 THE EVALUATION

One of the key objective of the research work is the evaluation of the grid-enabled portal prototype for GUISET. The objective of the evaluation however, is to find out if the portal prototype fulfills the minimum requirements to be suitable for the utility context. In a broader sense, we categorize the evaluation into two types namely analytic and emperical evaluation. Analytic evaluation deals with modeling and analysis of system functional requirements empirical evaluation deals with data collection techniques such as questionnaires and interviews from system users during the evaluation process.

4.5.1 Functional Requirements

A most critical question to ask is which, if any, of the traditional and the identified relevant requirements in this study does the grid-enabled portal prototype satisfy? An attempt to answer this question was made by drawing a chart and noting what requirements are met or not. Below is list of important items from the design requirements document and noting whether the portal meets these requirements.

Table 4.4**: The Security Requirements**

A. SECURITY REQUIREMENTS		
SCOPE	REQUIREMENTS	YES/NO/PARTIAL
1. Secured Login	• Single Sign-On to Portal & Portal Services	Yes
2. Authentication and Authorization	• Valid user details required to access the portal	Yes
	• Resources Information protected from unauthorized user access.	Yes
	• System Information Protected from Unauthorized user access.	Yes
3. Privacy	• User Information protected from unauthorized access.	Yes

4. User Interaction and Communication	• Well Secured Transaction	No
5. Credential Management	• Integration with Liferay's in-built Security. • Locally Stored Proxy Certificates	Yes No

Table 4.5: **The Membership & User Profile Management Requirements**

B. MEMBERSHIP & USER PROFILE MANAGEMENT REQUIREMENTS		
SCOPE	REQUIREMENTS	YES/NO/PARTIAL
1. User/Member Registration	• User Account Creation. • Member Registration.	Yes Yes
2. Membership Subscription and Accounts.	• Management of Members' Accounts & Subscriptions.	No
3. Accounting & Billing	• Accounting & Billing Service of Members.	No

Table 4.6: **The Users' Collaboration Requirements**

C. USERS' COLLABORATION REQUIREMENTS		
SCOPE	REQUIREMENTS	YES/NO/PARTIAL
1. Collaboration & User Community Enablement	• Online Chat	Yes
	• User Group Creation & Management.	Yes
	• Discussion Forum & user Communities.	Yes
	• Bulletin Boards & Blogs.	Yes
	• Shared Updates, News feeds, Calendar, etc	Yes
2. Messaging	• E-mails, Texts & Alerts	No
	• Videoconferencing	No

4.5.2 The Usability Evaluation

The usuability evaluation of the grid-enabled portal prototype was designed in a way that it can be achieved as effectively as possible. A set of relevant criteria for evaluating the usability of a grid-enabled portals were identified in literature. These criteria are itemized in section 4.5.3.

A total of twelve (12) users consisting of seven (7) final year undergraduate students and five (5) postgraduates students were selected from the Department of Computer and Information Sciences, College of Science and Technology, Covenant University, Nigeria to carry out the evaluation.

The selected participants were taken through a pre-experiment orientation, by being given various explanations on grid and utility computing concepts, the workings of grid-enabled portals and how to use them to achieve specified tasks. The participants were also taken through a brief work-through training on what the various requirements and expectations are. The purpose of conducting this was to make sure that participants have no difficulties to understand all steps of each task. This

really helped to make the task description simpler and easily understandable. After a substantial level of understanding of each participant was ascertained, they were then assigned a set of tasks to perform on the portal before they filled the usability evaluation questionnaires.

The stastitics on the background of the participants is presented in the table below:

Table 4.7: **The Background of Participants**

Background of Participants	**No. of Participants**	**Very High**	**High**	**Medium**	**Low**	**Poor**
Level of Understanding of Grid-based Technologies.	12	8.33%	25%	41.67%	16.67%	8.33%
Level of Experience with Use of Grid-enabled Portals	12	8.33%	25%	33.33%	25%	8.33%

Table 4.7: **The Background of Participants**

4.5.3 Questionnaire Results

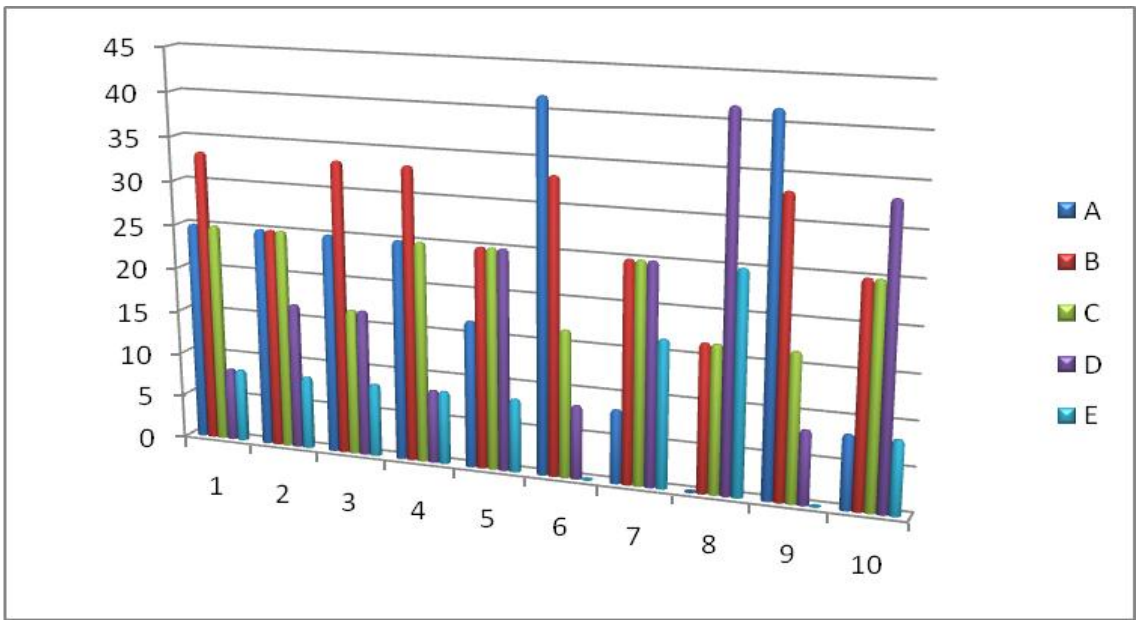
The questionnaire is designed by the author on the base of finding from usability test and guidelines for usability evaluation of the web sites provided by IS&T Department, MIT [95]. The result of the usability evaluation is presented in the table 5.5 below.

Table 4.8: **The Participants' Response**

S/N	Criteria	Questions	No. of Responses	Strongly Agree (A)	Agree (B)	Indifferent (C)	Disagree (E)	Strongly Disagree (E)
1.	**Simplicity**	i, ii	12	25%	33.33%	25%	8.33%	8.33%
2.	**Satisfaction**	iii, iv	12	25%	25%	25%	16.67%	8.33%
3.	**Aesthetics**	v, vi	12	25%	33.33%	16.67%	16.67%	8.33%
4.	**Memorability**	vii, viii	12	25%	33.33%	25%	8.33%	8.33%
5.	**Hypertext Structure**	ix, x	12	16.67%	25%	25%	25%	8.33%
6.	**Security**	xi, xii	12	41.67%	33.33%	16.67%	8.33%	0%
7.	**Resourcefulness and Job Management**	xiii, xiv	12	8.33%	25%	25%	25%	16.67%
8.	**Accounting**	xv, xvi	12	0%	16.67%	16.67%	41.67%	25%
9.	**Collaboration**	xvii, xviii	12	41.65%	33.33%	16.67%	8.33%	0%
10.	**Messaging**	xix, xx	12	8.33%	25%	25%	33.33%	8.33%
	AVERAGE			**21.67%**	**28.33 %**	**21.67%**	**19.16%**	**9.17%**

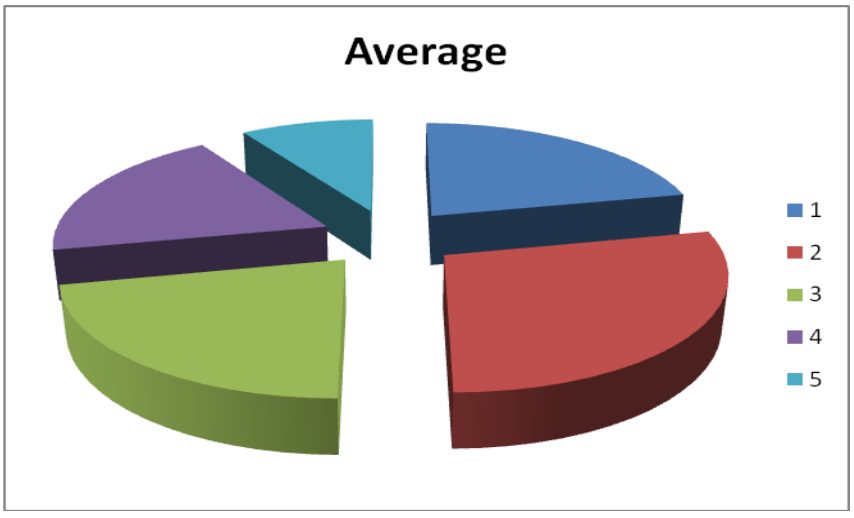
Key: **A:** Strongly Agree **B:** Agree **C:** No comments **D:** Disagree **E:** Strongly Disagree

The graphical representation of the above result is shown in figure 5.1 below. On x-axis, each criterion is described by numeric values from 1 to 10. Y-axis shows the percentages of responses. Response of each criterion is represented by alphabetic values from A to E depicting from strongly agree to strongly disagree.



Key: **A:** Strongly Agree **B:** Agree **C:** No comments **D:** Disagree **E:** Strongly Disagree

Figure 4.6: **Graphical Representation of Participants' Response**



Key: **1:** Strongly Agree **2:** Agree **3:** No comments **4:** Disagree **5:** Strongly Disagree

Figure 4.7: **Overall Result of the Evaluation**

This figure 5.2 depicts the overall response of the participants concerning the questionnaires. An average value of 21.67% participants is strongly agreed according to the questionnaire results. 28.33 % participants are agreed and the 21.67% was indifferent in the questionnaires. Average 19.16% participants are disagreed and 9.17% participants are strongly disagreed according to the results.

CHAPTER FIVE

5.0 SUMMARY, RECOMMENDATIONS AND CONCLUSION

This chapter gives a summary of this work. A number of recommendations for future works in this area were made and consequently the conclusion.

5.1 SUMMARY

The research work investigated and identified the relevant additional requirements that can be catered for in the design and development of grid-enabled portals for utility computing contexts.

The portal prototype implemented for GUISET serves as an interface meant to provide an enabling operating environment for every prospective utility service providers and customers willing to form or join any user business cluster or community [7]. It provides a street level entrance into a *bring-and-share* mode of utility computing [7] for every member or prospective member of the business community. GUISET is not an application but, an infrastructure that accommodates various services as a suite of service-oriented on-Demand Applications such as applications developed elsewhere by different service providers which could be e-Commerce, e-Agriculture, e-Health, e-Tourism, e-Government, etc [7]. GUISET therefore aims at technologically enabling the business activities of SMMEs by facilitating an affordable access to relevant technologies on a *pay-as-you-use* basis.

In the course of carrying out this work, an exploration of various key concepts was done. Key concepts such as: portlet-oriented architecture, service-oriented architecture (SOA), and its approaches to portal development, component-based concepts and approaches, web 2.0 etc. An in depth evaluation of existing grid-based portal tool kits was report and this helped to inform the choice of Liferay 5.2.3 portal tool kit [10, 25, and 26] bundled with Tomcat 6.0.18 as the tool with which a prototype of the portal system was built. The implementation of the GUISET portal offers a usable prototype that facilitates the realization of ODC platform for improved wealth creation and affordable access to scarce and expensive computing, particularly among SMMEs and rural-based businesses.

An evaluation of the grid-enabled portal prototype was conducted using a set of benchmark requirement standards, and also the usability evaluation of the portal prototype in a controlled experiment by a group of experienced users.

5.2 RECOMMENDATIONS AND FURTHER WORKS

This research work is part of a bigger on-going research endeavor embarked on by the Center of Excellence for Mobile e-Service, Department of Computer Science, University of Zululand, South Africa, which aims at building an evolutionary system - GUISET infrastructure. In this work not all identifiable additional requirements for design of grid-enabled portals have been considered. The scope of this work is limited to those that do not require the expensive third party infrastructure and usage access rights such e-Billing and e-Payment. Therefore, only a selected set of additional requirements have been considered and not all that is possible.

These requirements include: Content Management requirements, Portlets Management requirements, Service Registry Management requirements and a number of the unrealized requirements as specified in the functional requirements evaluation in 4.5.1 These can be further catered for in future works.

The GUISET portal is conceptualized as two separate web interfaces or sites [7]. These are:

3. GUISET Infrastructure Portal;

4. GUISET Service-driven e-Commerce on-Demand (SEConD) Portal.

The former is the primary focus of this research work. However, the latter represents the complementary portal interface which provides e-Commerce services on-demand. It is conceptualized as a portal use case into which future services can be plugged. It would serve as a common or uniform point of entry for service integration and provisioning - a “bring-and-share” mode of utility computing; advertisement and deployment of available services and products, service categorization and cataloging. It would provide a template for customer specification of service parameter – service lookup and binding. The administration of membership (SMMEs) accounts subscription and billing aspect of the portal would be addressed.

Other portal functionalities also envisaged are: Business-to-Consumer portal capabilities - product or service information and ordering capabilities. Portlets are provided for specific product or

service categories. These portlets are activated as needed by the portlet container (one portlet for each category of products or service information). Services exposed on the GUISET portal have either a Native Component or Portlets as their backend. Therefore, contributed services are exposed on a Hardware-As-A-Service Basis.

Thus, as the research work continues, it is intended that each of these requirements would be achieved afterwards as new research findings would be exploited to make the system fully realizable.

5.3 CONCLUSION

As the Grid & Utility Computing technologies mature, small, micro and medium enterprises (SMMEs) and organizations would be able to meet their IT infrastructure needs thereby reducing their investment on IT infrastructure. The GUISET Infrastructure is based on adopting the utility approach of service-oriented architecture (SOA) for service delivery; and it is conceptualized to support SMMEs providing a Grid-based Utility Infrastructural Services that can be requested on-Demand and paid for per every usage.

GUISET is not an application as it were, but, an infrastructure that accommodates various services as a suite of service-oriented on-demand applications such as applications developed elsewhere by different service providers which could be e-Commerce, e-Agriculture, e-Health, e-Tourism, e-Government, etc. However, as a grid-based utility infrastructure, GUISET is meant to provide an enabling operating environment through a *portal system* for every prospective utility service provider and customers willing to form or join any user business cluster or community.

The GUISET portal therefore is an interface meant to provide a street level entrance into a *bring-and-share* mode of utility computing for every member or prospective member of the business community. Therefore, it is believed that this work will have an indelible impact towards bringing the entire GUISET architecture alive. This will eventually enhance the various SMMEs that would be part of the infrastructure.

REFERENCES

1. Adigun, M.O, bEmuoyibofarhe, O. J, cMigiro, S.O, "Challenges to Access and Opportunity to use SMME enabling Technologies in Africa," a presentation at 1st All Africa Technology Diffusion Conference, Johannesburg South Africa, June 12-14, 2006.

2. David Sprott, "Service Oriented Architecture: An Introduction for Managers," *http://www*.roadmap.cbdiforum.com/reports/soa/managersintro.php, 2004.

3. Khosrow-Pour, M., "Encyclopedia of E-commerce, E-Government and Mobile Commerce," Information Resources Management Association, USA, 2006.

4. Eilam, T. et al, "Using a Utility Computing Framework to Develop Utility Systems," IBM Sys. Journal, Vol. 43(1), 2004.

5. Pagden, E. "The IT Utility Model-Part I, Sun Professional Services," Sun BluePrintsTM Online, July 2003

6. Migiro, S.O, Adigun, M.O, "ICTs, e-Commerce and rural development:the case of arts and crafts SMEs in rural KwaZulu-Natal," Commonwealth Youth and Development, (3)2, pp 65-83, 2005.

7. Adigun, M.O, "GUISET: An Architecture and A Vision," an Idina Hour Presentation at the Center of Excellence, e-Service, Unizul, KZN, South Africa. August 20th 2008.

8. Xavier, M.P., Belén, B.M., Miguel, V.L., "Grids Portals: Frameworks, Middleware or Toolkit," IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 4, May 2010 ISSN (Online): 1694-0784, ISSN (Print): 1694-0814.

9. Baker, M., Ong, H., Allan, R.J, Wang, X.D, "Virtual Research in the UK: Advanced Portal Services," University of Portsmouth, http://esc.dl.ac.uk/GridWG. 2005.

10. Akram, A., Dharmesh, C., Xiao, W.D, Yang, X., Allan, R.J., "A Software-Oriented Architecture for Portals Using Portlets," CCLRC e-Science Centre, CCLRC Daresbury Laboratory Warrington WA4 4AD, UK, 2005.

11. Russell, M., Novotny, J., Oliver, W., "GridSphere's Grid Portlets," Computational Methods in Science and Technology 12(1), 89-97 (2006).

12. Dharmesh, C., Akram, A., Allan, R. J., "Grid Middleware Portal Infrastructure," UK e-Science, ACM 1-58113-000-0/00/2004., 2004.

13. WSRP, Web Services for Remote Portlets. http://www.oasisopen.org/committees/tc_home.php?wg_abbrev=wsrphttp://www.oasis-open. org and http://www-106.ibm.com/developerworks/webservices/library/ws-wsrp/?Open &ca= daw-ws-dr

14. Yanli, C., Jian C., Minglu, L., Lei, C., "Portlet-based Portal Design for Grid Systems," Proceedings of the Fifth IEEE International Conference on Grid and Cooperative Computing Workshops (GCCW'06) 0-7695-2695-0/06, www.ieee.com 2006.

15. Arsanjani, A., "Service-Oriented Modeling and Architecture," IBM Developerworks, www.ibm.com, 2004.

16. Allan, R.J, Chris, A.C., Baker, M., Fish, A., "Portals and Portlets 2003," NESC Workshop, 14-17, April, 2004. http://www.nesc.ac.uk/technical_papers/UKeS-2004--XX.pdf .

17. Sprott, D., and Wilkes, L., "Understanding Service-Oriented Architecture," http://www.developer.com, 2004.

18. Introduction to JSR-168. Last accessed on July 25, 2010 http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/

19. Yang, X.D, Akram, A., Allan, R.J, "Developing portal/portlets using Enterprise JavaBeans for Grid users," Concurrency and Computation: Practice and Experience. Concurrency Computat.: Pract. Exper. 2005; 00:1-7 Prepared using cpeauth.cls [Version: 2002/09/19 v2.02].

20. Paul, A., "What is Web 2.0? Ideas, Technologies and Implications for Education," JISC Technology and Standards Watch, February, 2007.

21. Foster, I., Kesselman, C., "The Grid: Blueprint for a New Computing Infrastructure," Morgan-Kaufman, 1999.

22. Rappa, M. A. "The Utility Business Model and The Future Of Computing Services," IBM Sys, Jrnl., Vol. 43, NO 1, 2004.

23. Tom Phillip: "Utility Computing - Identifying the Application Domain and Its Boundaries," an MSc. thesis in Computer Science and Business Administration, submitted to the department of Informatics, University of Zurich, Switzerland, Dec. 2004

24. Michal, B., Bartalos, P., Bielikova, M., Filkorn, R., Tvarozek, M., "Adaptive portal framework for Semantic Web applications," Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava, Slovakia, www.fiit.stuba.sk, 2007.

25. Yang, X., Martin, T. D., Mark, H., Mark, C., Ligang, H., Peter M.R., "Survey of Major Tools and Technologies for Grid-enabled Portal Development," Proceedings of the UK e-Science All Hands Meeting 2006, © NeSC 2006, ISBN 0-9553988-0-0*UK, University of Cambridge, Cambridge, UK*, 2006.

26. Liferay Portal Enterprise, http://www.liferay.com , Oct. 2010.

27. Fang, J., et al., "Grid Portal System Based on GPIR," presented at the Second International Conference on Semantics, Knowledge, and Grid, IEEE, 2006.

28. Oscar, D., Salvador, T., Sandy, P., "Turning Portlets into Services: The Consumer Profile," International World Wide Web Conference Committee (IW3C2), WWW 2007/Track: Web Engineering, May 8–12, 2007, Banff, Alberta, Canada. ACM 9781595936547/ 07/0005.

29. Novotny, J., Michael, R., Oliver, W., "GridSphere: A Portal Framework for Building Collaborations," www.gridsphere.org, 2004.

30. Allen, G., Daues, G., Foster, I., Laszewski, G., Novotny, J., Russell, M., Seidel, E., Shalf, J., "The Astrophysics Simulation Collaboratory Portal: A Science Portal Enabling Community Software Development," Proc. of the 10th IEEE Intl. Symposium on High Performance. Dist. Comp 2001;

31. UNICORE. Last accessed September 2, 2010. http://unicore.sourceforge.net/.

32. Novotny, S. Tuecke, V. Welch. "An Online Credential Repository for the Grid: MyProxy," Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), 2001.

33. OGCE. Last accessed September 2, 2010. http://www.collab-ogce.org/ogce/index.php/Main_Page

34. Ge He, Zhiwei, Xu, "Design and Implementation of a Web-Based Computational Grid Portal," Web Intelligence (WI), pp.478, 2003 IEEE/WIC International Conference on Web Intelligence (WI'03), 2003

35. Adabala, S., et al., "From virtualized resources to virtual computing grids: the In-VIGO system," Advanced Computing and Information Systems (ACIS) Laboratory, 2004.

36. Chen, X., et al., "A Multilayer Portal Model for Information Grid," in the Third China Grid Conference, 2008, p. 8.

37. Globus Toolkits. Last accessed September 3, 2010. http://www.globus.org/toolkit/.

38. Li, M., et al., "PortalLab: A Web Services Toolkit for Building Semantic Grid Portals," in the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003, p. 8.

39. Wang, X.D, et al., "Flexible Grid Portlets to Access Multi Globus Toolkits," in The Fifth International Conference on Grid and Cooperative Computing Workshops, 2006, p. 6.

40. Foster, I., Kesselman, C., Nick, J, and Tuecke, SW., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG," Global Grid Forum, June 22, 2002.

41. Dahan, M., et al., "Grid Portal Toolkit 3.0 (GridPort)," in Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, 2004, p. 2.

42. Natis, Y. V., "Service-Oriented Architecture Scenario," Gartner Publication, www.gartner.com/resources/114300/114358/114358.pdf, 2003

43. W3C, Simple Object Access Protocol (SOAP) Version 1.2, W3C Recommendation 24th June 2003.

44. Lee, S.P., Chan, L.P. Lee, E.W., "Web Services Implementation Methodology for SOA Applications," 1-4244-9701-0/06, www.ieee.com 2006.

45. W3C, Web Services Description Language (WSDL) Version 2.0, W3C Candidate Recommendation 27th, March 2006.

46. Akram, A., Dharmesh, C., Wang, X.D., Yang, X., Allan, R.J., "WSRP Reincarnate of SOA," CCLRC e-Science Centre, CCLRC Daresbury Laboratory Warrington WA4 4AD, UK, 2005.

47. Evolution of UDDI, The Stencil Group White Paper published 19th July 2002 at UDDI.org

48. Pluto Project. Last accessed September 8, 2010. http://portals.apache.org/pluto/.

49. Stevens, M., "Service-Oriented Architecture Introduction, Part 2," http:// softwaredev.earthweb.com/msnet/article/ 0,,10527_1014371,00.html, 2004.

50. Sheldon, F.T, Jerath, K., Pilskalns, O., Kwon, Y., Kim, W., and Chung, H., "Case Study: B2B E-commerce System Specification and Implementation Employing Use-Case Diagrams, Digital Signatures and XML," Multimedia Software Engineering, Proceedings, pp. 106 – 113, 2002.

51. IBM WebSphere MQ. Last accesses September 8, 2010. http://www.redbooks.ibm.com/.

52. Martin, K., Susan, B., Alan, H., Sven, M., Chris, N., Rick, R., Jonathan, A., Paul, V., "Patterns: Implementing an SOA with the Enterprise Service Bus," Redbook SG24-6346-00, www.ibm.com/redbook, August 2004.

53. Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., and Newling, T., "Patterns: Service-Oriented Architecture and Web Services," www.ibm.com/redbook, 2004.

54. Common Object Request Broker Architecture, CORBA. Last accessed September 10, 2010. http://www.corba.org/.

55. Remote Method Invocation, RMI http://java.sun.com/j2se/1.5/pdf/rmi-spec-1.5.0.pdf, 2010.

56. Distributed Component Object Model (DCOM) Remote Protocol Specification", Microsoft Corporation, [MS-DCOM] — v20101001, Oct 1, 2010.

57. David, B., Haas H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D., "Web Services Architecture. W3C Working Group Note 11," http://www.w3.org/TR/2004/NOTE-ws-arch-20040211, 2004.

58. Fiorano Software, Inc., "Service Oriented Architecture Implementation Frameworks," www.fiorano.com.

59. Michael, C., "Web Services Architecture W3C Working Draft 14 2002," http://www.w3.org / TR/2002/WD-ws-arch-20021114, 2002.

60. Qureshi, M.R., "Reuse and Component Based Development," COMSATS Institute of Information Technology, www.ciitlahore.edu.pk 2006.

61. Kaisler, S.H., "Software Paradigms," John Wileys & Son, 2005, pp99-100.

62. Hunt, J., "Agile Software Construction," Springer-Verlag London 2006, pp193-204.

63. Highsmith, J., "Extreme Programming," Cutter Consortium White Paper, www.cutter.com, Sept. 2010.

64. IBM Rational Unified Process (RUP), http://www.redbooks.ibm.com/, Sept. 2002

65. Luiz Fernando Capetz, Miriam A.M. Capretz, Dahai Li, "Component-Based Software Development," IEEE Industrial Electronics Society, 2001, pp. 1834-1837.

66. Jim Q. Ning, "A Component-Based Software Development Model," IEEE Software, September 1996, pp. 390-391.

67. Paleologo, G. A. "Price – at – risk: A Methodology for Pricing Utility Computing Services," IBM Sys. Jrnl, Vol. 43, NO 1, 2004

68. Global Grid User Support System, GGUS, https://gus.fzk.de/pages/home.php, 2010.

69. MD-web, http://desktop.psnc.pl/, 2010.

70. Barbera, R., "The GENIUS Grid Portal, (Genius-web)" Computing in High Energy and Nuclear Physics, 24-28 March 2003.https://www.genius.ct.infn.it/, 2010

71. ACESSGRID-web, www.accessgrid.org/ 2010.

72. The P-Grade Portal: PG-web, www.lpds.sztaki.hu/pgportal/ , 2010.

73. LUNARC Application Portal: LUNARC-web, https://grid.lunarc.lu.se/lap/, 2010.

74. Batch Object Submission System, BOSS http://boss.bo.infn.it/, 2010.

75. EnginFrame (EF-web), http://www.it-tude.com/engineframeportal.html, Oct. 2010.

76. gLite grid middlewares, Last accessed September 8, 2010, http://glite.web.cern.ch/glite/default.asp Oct. 2010.

77. Torterolo, Livia, "GENIUS Portal, EnginFrame Framework: Features, Architecture and Future Perspectives," GridKa School, Forschungs ZentrumKarlsruhe, livia.torterolo@nice-italy.com, http://www.nice-italy.com/web/nice/home Sept. 11-15, 2006.

78. uPortal by JA-SIG. Last accessed September 10, 2010. www.uportal.org.

79. Novotny, J., Russell, M., Wehrens, O. "GridSphere: A Portal Framework," GlobusWorld, www.gridsphere.com , 2004.

80. Novotny, J., "The Grid Portal Development Kit," IEEE Concurrency: Practice and Experience, 2002.

81. Sajjad, A., Jameel, H., Kalim, U., Lee, Y.K. and Lee, S. (2005), "A Component-based Architecture for an Autonomic Middleware Enabling Mobile Access to Grid Infrastructure," LNCS, Vol. 3823, Springer, Berlin, pp. 1225-34.

82. Kiani, S.L., Riaz, M., Lee, S., Jeon, T. and Kim, H. (2005), "Grid access middleware for hand held devices, advances in grid computing – EGC 2005", in Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A. and

Bubak, M. (Eds), European Grid Conference, Amsterdam, The Netherlands, LNCS, Vol. 3470, 14-16 February, p. 1002.

83. Czajkowski K., Ferguson D., Foster I., Frey J., Graham S., Maguire T., Snelling D., Tuecke S., "From Open Grid Services Infrastructure to WSResource Framework: Refactoring & Evolution", Version 1.1, 2004.

84. Java Community Process: JSR 152 Java Server Pages 2.0 Specification, Last accessed September 12, 2010, http://www.jcp.org/en/jsr/detail?id=152

85. Jetspeed-2: Last accessed September 8, 2010, http://portals.apache.org/jetspeed-2/

86. Chang, S.H., and Kim, S.D, "A Service-Oriented Analysis and Design Approach to Developing Adaptable Services," Proceedings of IEEE International Conference on Services Computing, SCC 0-7695-2925-9/07, 2007.

87. Jan Schulz. H., "Web Service Middleware – An Infrastructure for Near Future Real Life Web Service Ecosystems," proceedings of the IEEE international conference on service-oriented computing and applications (SOCA'07), 0-7695-2861-9/07. www.ieee.com, 2007.

88. Open Grid Services Infrastructure (OGSI) Version 1.0, June 2003.

89. Tomislav Urban, "GridPort: Using Globus for Grid- Enabled Web Portals", GlobusWorld, 2004.

90. Whitten J. L., Bentley L. D., Dittman K. C., "Systems Analysis and Design Methods", McGraw Hill, 6th Edition, New York, 2004.

91. Content Management System, Last accessed March 10, 2011, http://www.jcp.org/en/jsr/detail?id=170.

92. Struts, Last accessed March 10, 2011, http://struts.apache.org/.

93. Hibernate, Last accessed March 10, 2011, http://www.hibernate.org/.

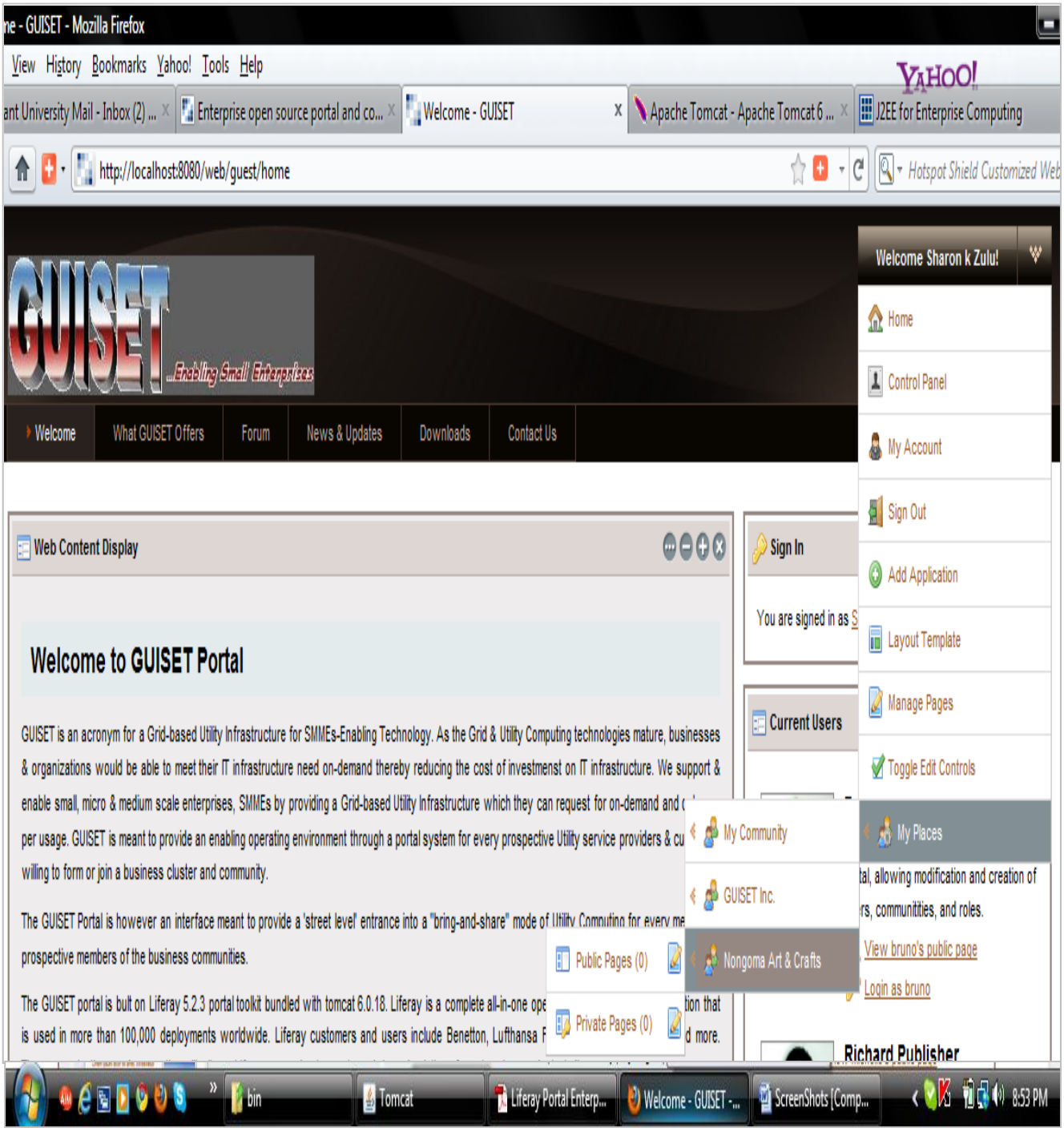
94. Java Messaging Service, Last accessed March 10, 2011, http://java.sun.com/products/jms/.

95. Usability Guidelines.(2007),Information services and technology-MIT, http://web.mit.edu/ist/usability/usability-guidelines.html, Last accessed June 23, 2011.

APPENDIX I

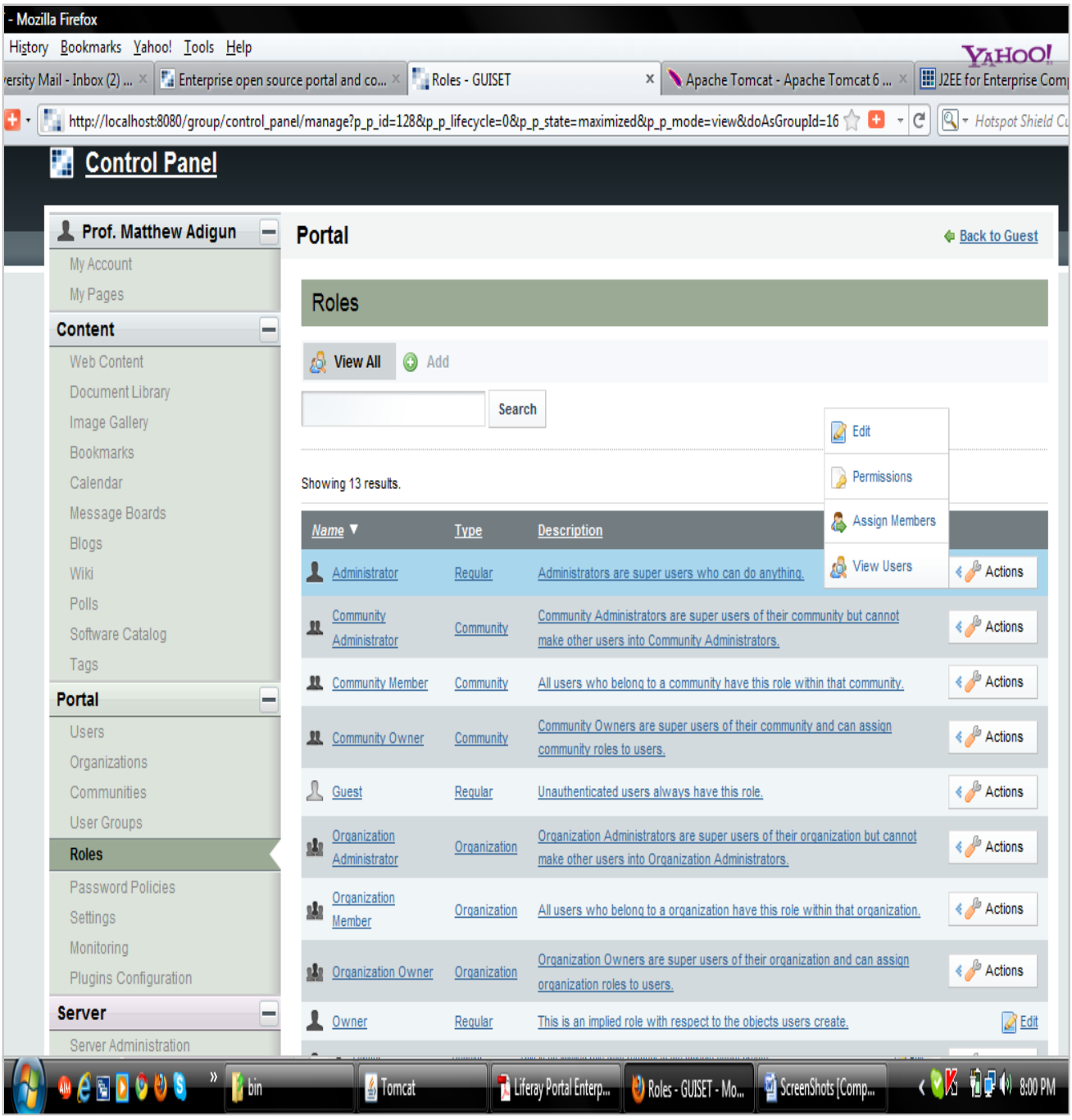
A. The Registered User General Home Page

Whenever a newly registered member logs into the portal, the general home page is displayed, to welcome the member. The member is presented with various options through the tabs and portlets from which he can decide whatever choices to make.



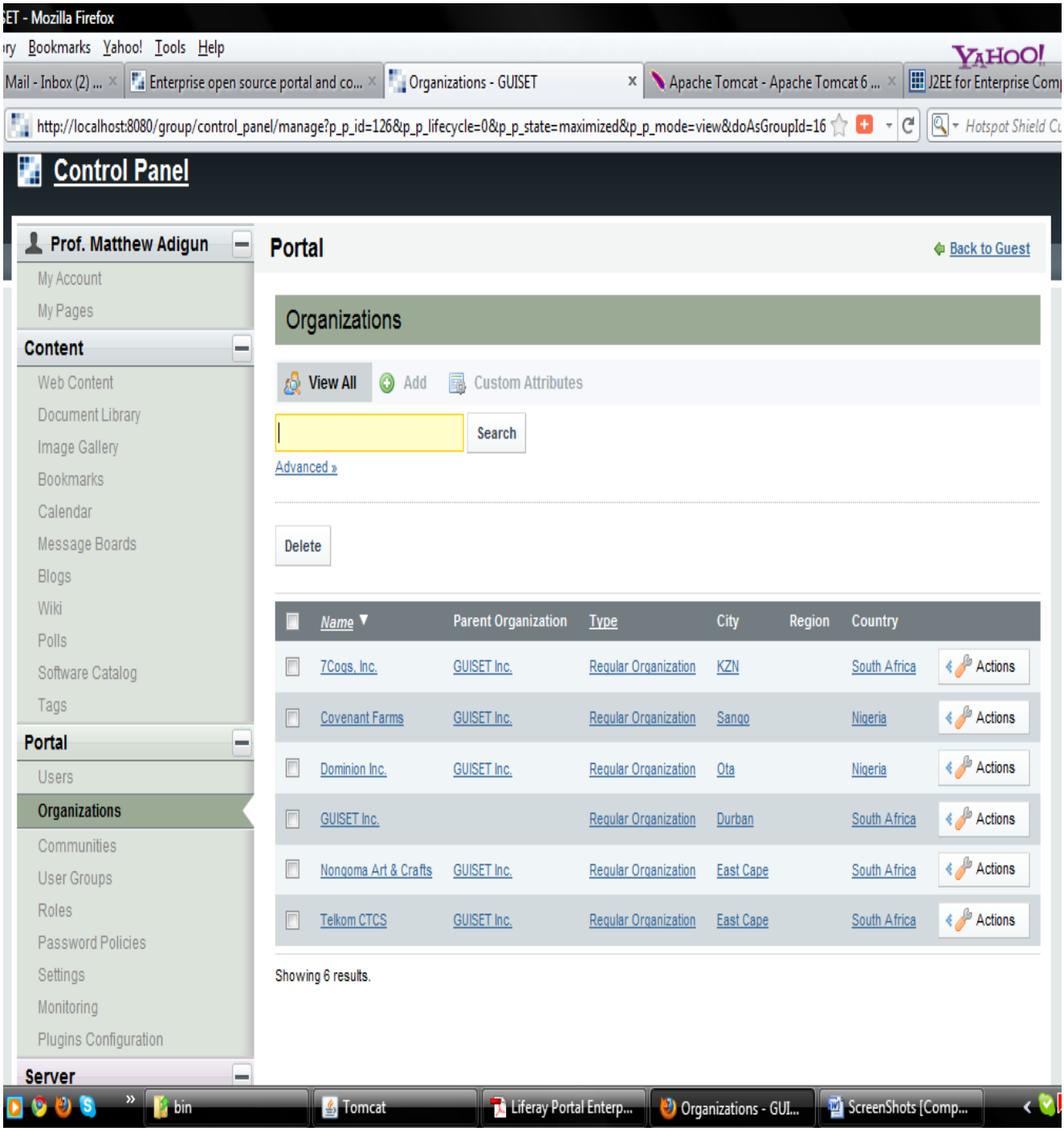
B. The Administrator's Assign Roles Interface

One of the responsibilities of the portal administrator is the management and assigning of appropriate roles to different users. The various roles alongside their respective description are shown below.



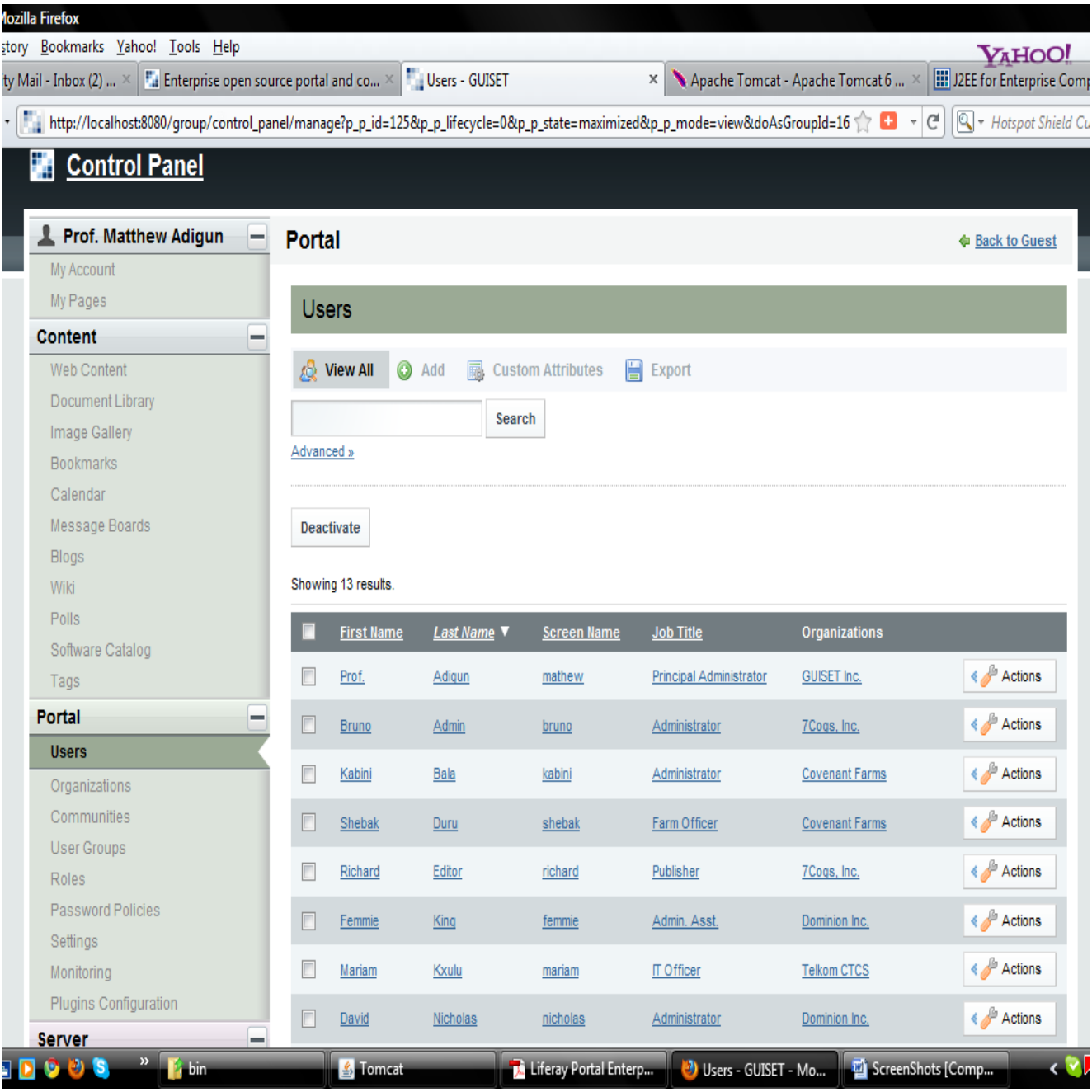
C. The Registered Enterprises Interface

The registered enterprises interface presents a view of the registered enterprises on GUISET portal. It also presents a medium for the administrator to add more enterprises to the system.



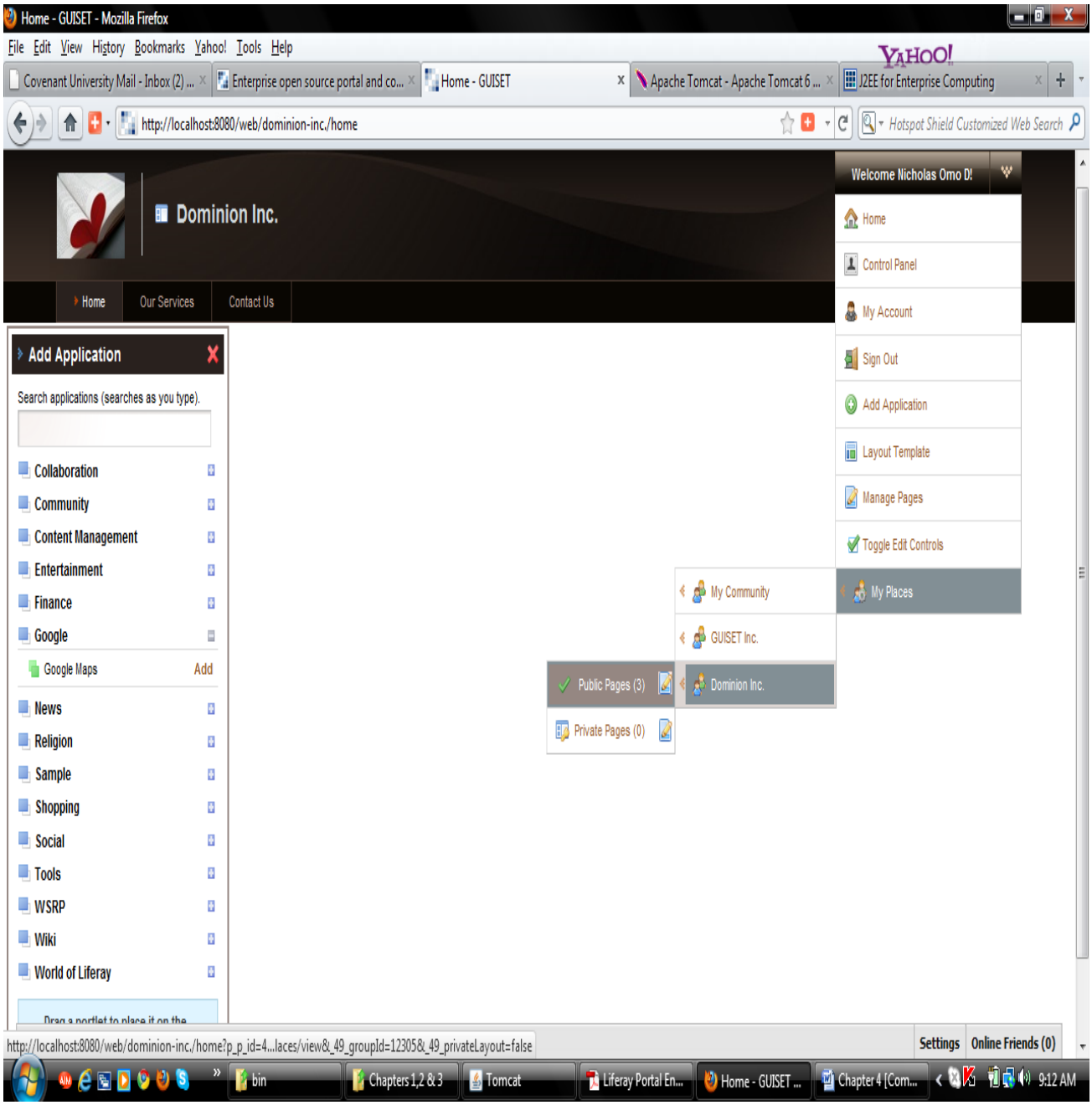
D. The Registered Users' Interface

The registered users' interface presents a view of the registered users on GUISET portal, showing the various organizations they belong to. It also presents a medium for the administrator to add or assign more users to the registered organizations.



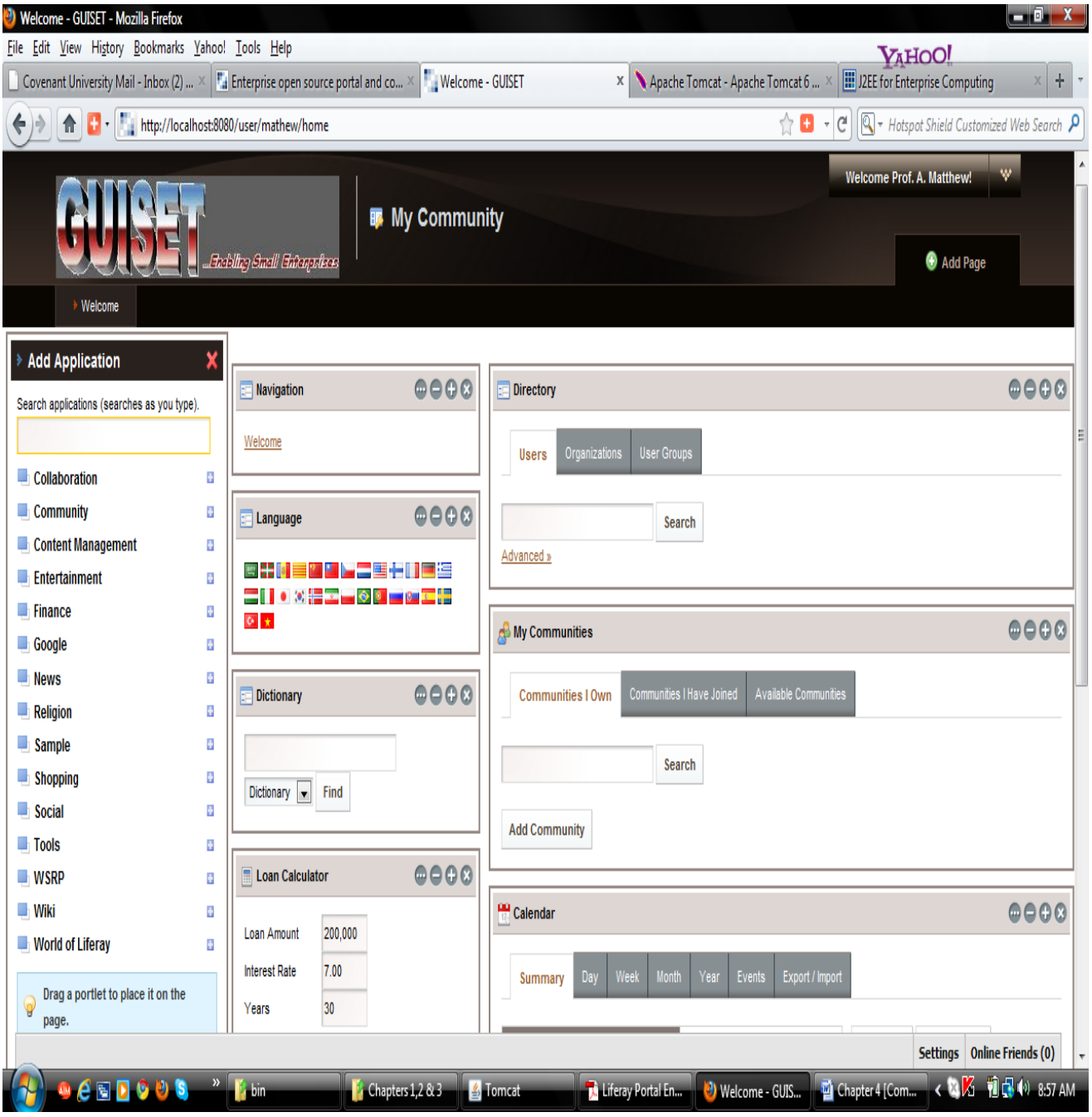
E. Enterprise' Personalized Web Pages

Aside the general home page for the registered members, every enterprise is presented with the medium on the GUISET through which a personalized website can be built easily and with minimum development cost. For example, a sample personalized website for one of the registered enterprises, Dominion Inc. is shown below. The user also has the opportunity to add various application encapsulated as portlets as desired.



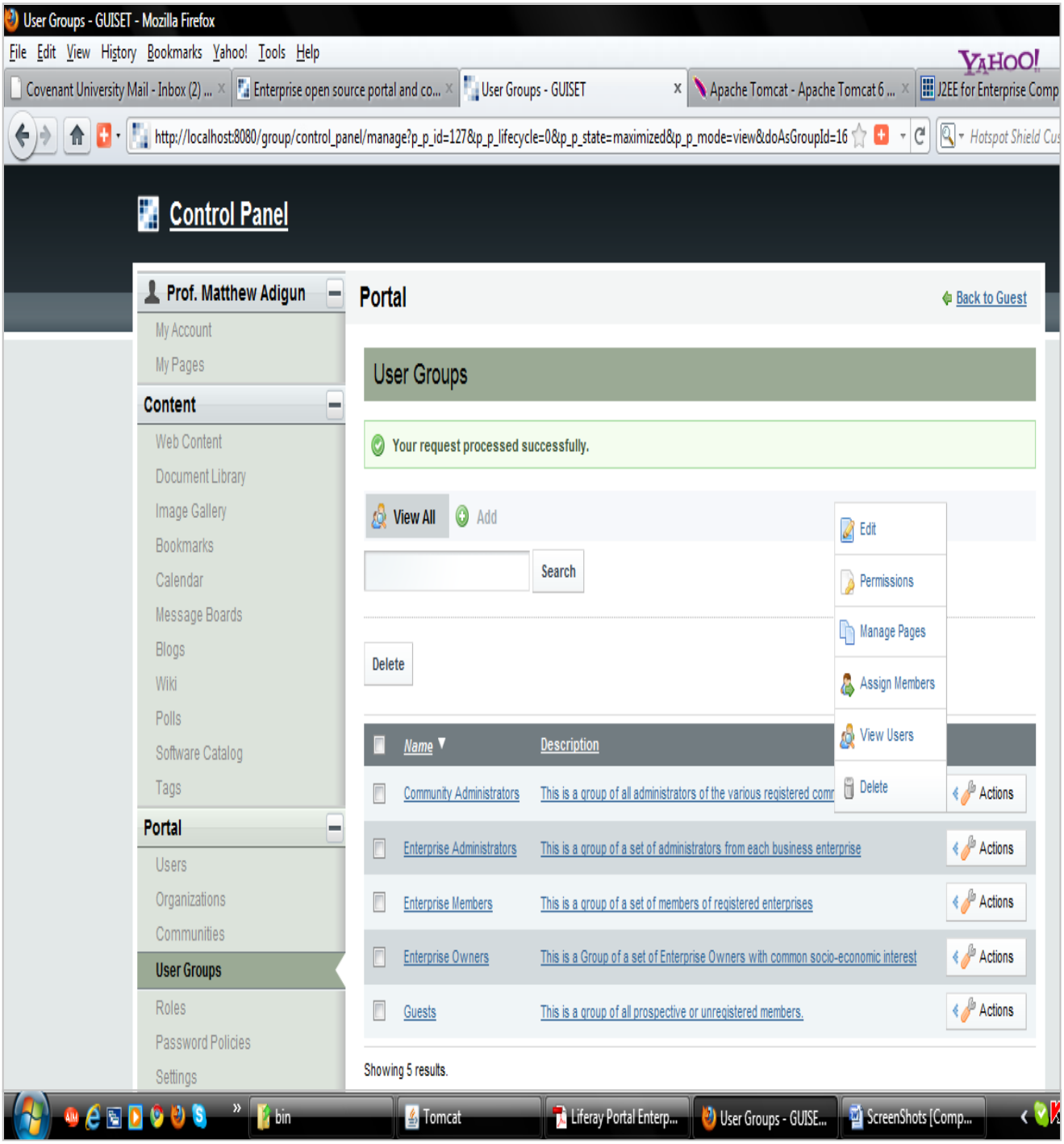
F. A User's Personal Community Page

Every registered member belonging to an enterprise is designed to have a personalized community page. The user can belong to a number of user communities where he can share common interests with other members and also relates with other communities. With the personal community page, the user has a various options on how he relates with the communities he belongs to.



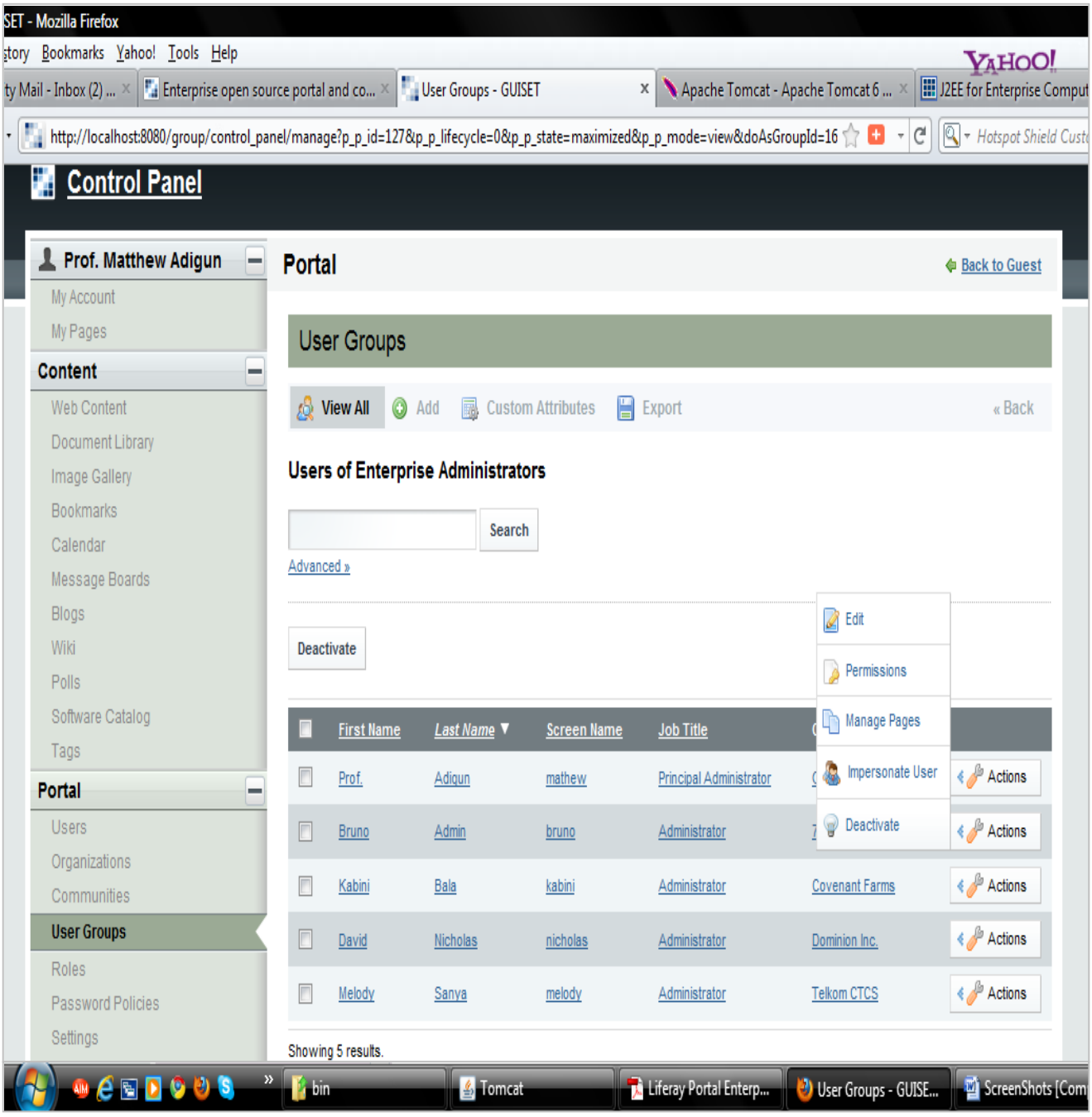
G. User Groups Interface

A user group is a special group which may have a set of associated users based on similar characteristics. The various existing user groups are presented through this interface, For instance, Enterprise Owners, Enterprise Administrators, etc.



H. A Sample User Group- Enterprise Administrators

A sample existing user groups called Enterprise Administrator is made up of some or all the administrators from the different enterprises.



QUESTIONNAIRE FOR EVALUATION

This questionnaire is designed to evaluate the usability level of the GUISET Portal infrastructure. Your objective and sincere response would be highly appreciated. Kindly Tick (√) as appropriate.

Section A:

1. **Level of Understanding of Grid-based Technologies**: Very High ( ); High ( ); Medium ( ); Low ( ); Poor ( );
2. **Level of Experience with Use of Grid-enabled Portals**: Very High ( ); High ( ); Medium ( ); Low ( ); Poor ( );

Section B:

S/N	Questions	Strongly Agree (A)	Agree (B)	Indifferent (C)	Disagree (E)	Strongly Disagree (E)
1.	**Simplicity**					
i.	The GUISET Portal is understandable and very easy to use.					
ii.	The Portal generally is simple to browse without any difficulty.					
2.	**Satisfaction**					
iii.	The GUISET Portal requires few steps to complete any task.					
iv.	The Portal saves my time in accomplishing any task.					
3.	**Aesthetics**					
v.	The GUISET Portal has well designed pages easy to navigate.					
vi.	The Portal highlights most important contents on the main page.					
4.	**Memorability**					
vii.	How to Use the GUISET Portal can easily be remembered.					
viii.	I would like to revisit the portal as often as I could.					
5.	**Hypertext Structure**					
ix.	Information about GUISET Portal services is well structured.					
x.	There're active links to various portal features & Services.					
6.	**Security**					
xi.	The portal supports single sign-on to GUISET information and services					
xii.	The Portal denies you unauthorized access to user and resource information on the grid infrastructure.					
7.	**Resourcefulness & Job Management**					
xiii.	The GUISET portal provides adequate information and access to available resources and services.					
xiv.	User Job requests are well managed and executed.					
8.	**Accounting**					
xv.	The GUISET portal properly manages & administers users' accounts and subscription.					
xvi	The Billing & QoS requirements are well taken care of.					
9.	**Collaboration**					
xvii	The GUISET supports user groups, forums and communities					
xviii	The Portal allows for shared updates, news, alerts and RSS feeds					
10.	**Messaging**					
xix	The GUISET portal supports online messaging: e-mails, chats, etc					
x	The Portal is multimedia-enabled: Videoconferencing, Teleconferencing, etc.					