International Journal on Advances in Information Sciences and Service Sciences
Volume 2, Number 1, March 2010

Design of SOA-based Grid Computing with Enterprise Service Bus

Alaa M. Riad [1], Ahmed E. Hassan [2], Qusay F. Hassan *[3]
[1] *Head of Information Systems Department, Faculty of Computer Sciences and Information Systems, Mansoura University, Egypt*
[2] *Teacher of Electrical Engineering, Faculty of Engineering, Mansoura University, Egypt*
*[3] *Corresponding Author* *PhD Researcher in Faculty of Computer Sciences and Information Systems, Mansoura University, Egypt*
Amriad2000@yahoo.com, arwaahmed1@gmail.com, qfadhel@usaid.gov
doi: 10.4156/aiss.vol2.issue1.6

Abstract

Due to great advantages that Service-Oriented Architecture (SOA) offers to its adopters in almost all fields, many studies tried to leverage it in grid computing. These studies focused on enabling easy access and flexible management to underlying grid resources. However, none of them allowed access to grid resources through various technologies. Rather, most of them utilized SOA in terms of XML web services without considering its limitations. In this paper, we will introduce a high level architecture that goes beyond traditional efforts in leveraging SOA in grid computing. This architecture uses Enterprise Service Bus (ESB) model by leveraging Windows Communication Foundation (WCF) to offer a number of endpoints for published services. With these endpoints different requirements from both grid clients and executors could be easily and efficiently met. Proposed architecture does not tend to supersede other SOA-based grid computing frameworks and standards. Instead, it offers a new method for wrapping grid computing resources with a set of configurable WCF services that could be utilized by these frameworks.

Keywords: *SOA; Enterprise Service Bus, Grid Computing; .NET; Windows Communication Foundation.*

1. Introduction

Grid computing refers to ability to combine different computing resources to act as a single powerful unit [1]. These resources may include processing cycles, memory, disk spaces, networks, printers, scanners, software licenses, remote devices, etc. Grid computing is widely used to process computationally intensive tasks faster and cheaper [2]. This is usually accomplished by dividing complex computation tasks into a large number of simple subtasks (jobs) that could be processed in parallel using available computing resources [3]. Certainly, grid computing is being utilized to solve different problems such as drugs discovery, economic forecasting, disasters forecasting, climate prediction, data processing, graphics and videos rendering, and complex mathematical problems.

Technically, grid computing has two main forms listed as follows:

- **Dedicated:** A number of computing resources known as executors are dedicated to execute available tasks for a predefined period of time.
- **Non-Dedicated/Voluntary Computing:** Executors donate their idle computing resources to process intensive tasks [4, 5].

Executers of both forms may or may not reside inside organization that uses their computing resources. If executors are not located inside organization, they could be accessed through internet [6]. For example, SETI@home is a project that allows internet users to donate their computers while being idle to analyze extraterrestrial photos [7]. In all cases, it is a must to enable efficient interaction between clients and executors. This interaction is needed to allow clients to send their tasks to grid to be processed by available resources, and on other side, to allow clients to receive returned results.

One way that is gaining momentum in almost all kinds of software applications and systems is Service-Oriented Architecture (SOA) [8, 9]. This method aims to enable clients and servers to communicate with each other through easy-to-access, loosely-coupled, and interoperable services [10]. In this model, service consumers (also known as clients and invokers) can discover and bind to available services through published interfaces [11]. These interfaces are used to describe offered services formally and informally. Formal information may include description for available operations, input and output parameters, and QoS (quality of service) terms. Whereas, informal description may offer information about service creators, contact information, and SLAs (service level agreements).

The unique thing in SOA model is that service consumers are not aware of underling platforms, programming languages, and other technical issues used build up services. In other words, all technical and complicated information are totally hidden from invokers. Thus, the task of service invokers is just to find and use services that meet their business and technical needs.

Due to advantages of SOA model, it has been thought that it could be leveraged efficiently in grid computing field. This would allow clients to have access to grid resources easily and flexibly no matter what technologies are used to construct them. In this paper, we focus on applying SOA concepts to grid computing by offering a high level architecture for an SOA-based grid computing. This architecture is designed to enable sharing of underlying resources and capabilities in terms of configurable, interoperable, and easy-to-use services.

The rest of this paper is organized as follows. The next section presents background and related work. Section 3 provides an introduction to Windows Communication Foundation and related terms. Section 4 describes the computation model of proposed architecture. Section 5 lists the key advantages of proposed architecture. Finally, Section 6 concludes.

2. Background and Related Work

As mentioned, different studies and models tried to leverage SOA concepts in grid computing in different forms including (but not limited to) [12 - 16]:

- **Bayanihan:** It is an SOA-based framework that is built with .NET and uses a number of generic XML web services. These services allow end-users to easily send their tasks to grid computing resources and retrieve returned results while hiding underlying complexities.
- **Alchemi:** It is a programming framework built with .NET technology to allow developers to construct grid-based applications in an easy manner. The core of this framework depends on multithreaded/parallel processing for client requests to make better utilization for available processing cycles.
- **Open Grid Services Architecture (OGSA):** It defines standard mechanisms for creating, naming, and discovering grid services of homogenous resources that are exposed as XML web services.

In fact, all previous studies as well as others have enriched the field of grid computing by defining standards and tools that utilize SOA. However, all of them focused on implementation of SOA from XML Web Services perspectives. To have ultimate utilization for SOA in grid computing field, there should be a way that enables sharing of services without considering underlying technologies. Moreover, designed model must give optimum support to grid clients according to their physical locations. For instance, communications between internet clients/executors and grid resources have different needs and specifications than those of LAN clients/executors. In this respect, depending on traditional XML Web Services in all cases is not the optimum trend. This may lead to slower processing for LAN requests while it might be less secure for internet requests, for example.

Our approach focuses on leveraging SOA in grid computing by using Enterprise Service Bus (ESB) rather than simple XML Web Service. In subsequent sections, we illustrate how ESB would enable meeting the needs of grid clients and executors in an efficient and flexible manner.

3. Windows Communication Foundation (WCF)

Different platforms and technologies are available to enable SOA-based applications such as RPC, Message Queues, CORBA, and XML Web Services. Each of them has an array of capabilities that makes it a suitable realization option for specific needs and cases. However, all of these technologies are mostly used in simple and straightforward scenarios.

Conversely, ESB is a complex framework that allows development and integration of distributed systems in an easy and manageable manner. This is accomplished by offering a set of features such as central administration and configuration; high performance; interoperability; transaction support; security; reliable synchronous/asynchronous communication; serialization facilities; and broad variety of encoding options. ESB model gives service providers ability to plug their services into the bus whenever needed with minimal time and effort. Moreover, it allows service providers to combine services built with different tools to be published and used by consumers regardless to their underlying technologies and protocols [17].

These reasons put ESB on top of the standard technologies that could be used to build SOA applications, leading most of key software vendors tried to exploit ESB capabilities in their products to allow easy realization for SOA in enterprise-scale implementations. One of these vendors is Microsoft that came up with Windows Communication Foundation (WCF) to enable adoption for SOA in large-scale systems. Technical terms of WCF are described in more details in subsequent sections.

3.1. Definition and Architecture

WCF (formerly known as Indigo) is the state-of-the-art technology that is offered by .NET Framework 3.0 (and latter versions) to act as an ESB to enable service-oriented applications and systems on Windows-based machines [18, 19, and 20].

According to SOA concepts, WCF is a message-based architecture that is composed of three main elements listed as follows [21]:

1. **Service:** An exposable unit that contains a set of functions to cover specific business and/or technical requirements. Each service must contain information that allows its consumers to find and use it easily. This information is available through public interfaces known as *Contracts*. WCF contracts may include information about offered operations; input parameters; return types; security requirements; supported programming languages; transport protocols (such as HTTP and TCP); and wire encodings (such as text/XML, binary, and MTOM) being used.
2. **Service Provider:** Constructs and maintains WCF services with all related description and metadata information. These services are created to provide business and/or technical functionalities needed by consumers.
3. **Service Consumer:** Locates and binds to (uses) services that meet his needs. Each service consumer is responsible for building applications that will make use of offered services. These applications may come in different forms such as desktop applications, portals, thin-client applications on portable devices, web services, WCF services, etc.

WCF implements SOAP (Simple Object Access Protocol) and WS-* specifications to enable interoperability and communication between interacting nodes [22]. Thus, loose-coupling is reserved between service providers and consumers.

3.2. Why WCF?

It is known that no single technology can fit all needs especially for complicated scenarios in distributed systems. However, WCF offers a single stack that comprises distributed technologies offered by .NET Framework such as ASP.NET web services (ASMX); .NET Remoting; Enterprise Services; Web Services Enhancements (WSE); COM+; and Microsoft Message Queuing (MSMQ). This unified stack allows service providers to easily offer one

service with different settings and specifications that are compatible with different service consumers. These settings may include communication mechanisms, encodings, transports, security, transactions support, etc.

Defining one service with a number of settings sets could be easily done by offering different *endpoints* to that service. Endpoint refers to a construct that is exposed to enable messages to be sent and/or received. Each of these endpoints is accessible through a unique address (URI), and underlying settings could be configured by service providers either in code or in configuration files. The service provider has the ability to configure offered services using a wide range of system-provided bindings in addition to custom bindings. Bindings are responsible for defining the characteristics of endpoints exposed for available services.

Furthermore, WCF provides higher performance when compared with any of other distributed technologies offered by .NET Framework [23]. High performance is a key requirement in all computer systems especially in those designed to speed up processing times like grid computing.

4. Proposed Architecture

We here propose a high level architecture for an SOA-based grid computing system that uses WCF to build and configure needed services. These services should allow clients to send their jobs, executors to pull jobs from management server to process them, and again clients to retrieve returned result sets. Our work is not focused on introducing a new SOA-based grid computing framework. Rather, it introduces a way for wrapping grid computing resources with set of configurable WCF services that can overcome limitation of traditional XML Web Services.

As illustrated in figure 1, the system exposes a number of services that act as interfaces to management server. In grid computing, management server is usually responsible for preparing tasks, checking returned result sets, and packaging and offering final results to original senders. Each of available services is exposed with a number of different endpoints to meet different needs. For instance, different security settings, transport protocols, serialization techniques, and encoding options.

For basic scenarios, we need three main endpoints described in the following:

- **NetTcpBinding:** This option is suitable for homogenous (.NET-to-.NET) communication over LANs. Although this option is less interoperable, it is the fastest system-provided bindings offered by WCF. NetTcpBinding uses TCP as a communication protocol and binary serialization for data being transmitted between clients and servers and vice versa. Depending on this mechanism exempts data being transmitted from using XML data and HTTP layers resulting in efficient and high performance interaction between nodes [24]. *WS-ReliableMessaging* could be used by setting *reliableSessionEnabled* element to *on* that is set to *off* by default for highest performance possible. This element is offered by WS-* to allow reliable transmission for data between clients and services and vice versa.
- **BasicHttpBinding:** This option allows endpoints to be accessed as if they were traditional XML Web Services (ASMX services in .NET). Inarguably, this binding is slower than NetTcpBinding as it uses a plain-text messages and UTF-8 encoding by default. However, BasicHttpBinding could be configured to use more efficient data formats such as binary or Message-Transmission Optimization Mechanism (MTOM) methods. The interesting thing about BasicHttpBinding endpoints is the conformance to WS-I Basic Profile 1.1 making it more interoperable. This interoperability allows external clients (both homogenous and heterogonous) to have access to grid resources in an easy and flexible manner. Accordingly, we recommend this endpoint to be used by trusted clients who have access to services from the WANs.
- **WSHttpBinding:** This option uses HTTP transport and WS-* specifications, so it is similar to BasicHttpBinding but with more advanced features. This includes support for federation;

distributed transactions; and secure and reliable communication sessions. WSHttpBinding could be upgraded to *WS2007HttpBinding* version that uses the Organization for the Advancement of Structured Information Standards (OASIS) versions of the *ReliableSession*, *Security*, and *TransactionFlow* protocols. This option is recommended for Internet users (both clients and executers) to guarantee security and reliability.

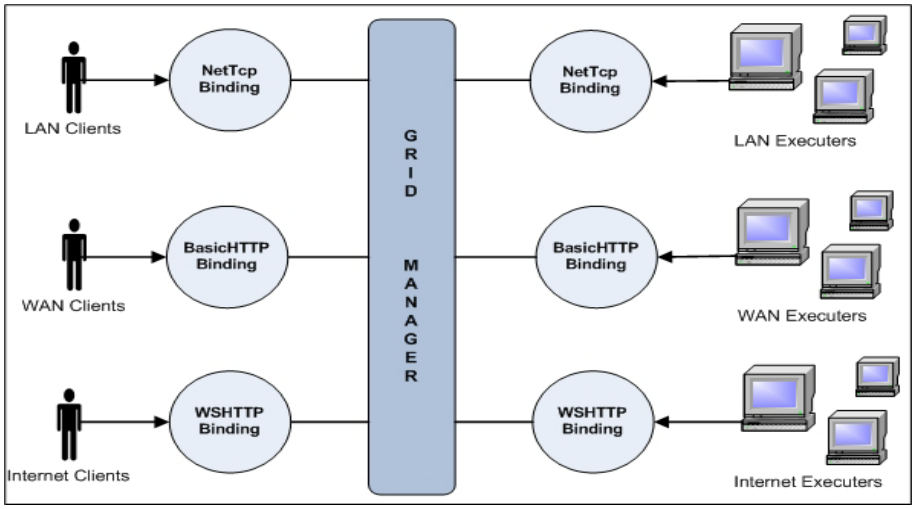

Figure 1. Proposed Architecture

As illustrated in figure 2, our architecture could be easily enhanced by adding more endpoints with other bindings. For example, we may create a new endpoint that uses *WSFederationHttpBinding* (or *WS2007FederationHttpBinding*). This binding uses *WS-Federation*, *WS-Trust*, and *HTTP* protocols to allow federation and integration with trusted partners such as departments, subsidiaries, organizations, universities, etc. Such federation might allow us to integrate our grid with other grids to form a larger and more powerful computing architecture that could be used whenever needed in advanced scenarios.

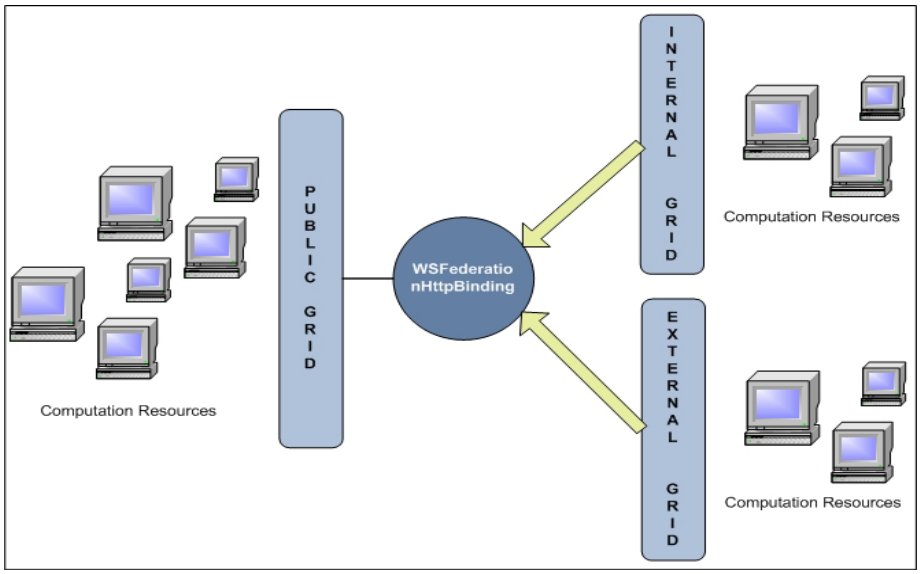

Figure 2. Ability to Federate/Share Grids

As mentioned, all settings and configurations might be made either in code or in configuration files. We prefer the second method as it allows administrators to easily and flexibly define, adjust, and enable/disable endpoints on-the-fly without demanding modification and recompilation for source code.

As illustrated in Figure 3, the *Client* service exposes three main functions that allow grid clients to manage their tasks:

- **AddTask:** Sends a new task to management server to be processed by grid resources. Grid tasks might be implemented in different ways. For example, they might be created as assemblies with number of concurrent threads to be executed in parallel by available executors.
- **KillTask:** Terminates further processing for specified task.
- **GetResult:** Receives results generated from the processing of specified task.

```
Using System;
using System.ServiceModel;

namespace WCFClientService
{
   public class ClientService : IClientService
   {
      public void AddTask()
      {
         //Implementation Code
      }

      public void KillTask()
      {
         //Implementation Code
      }

      public void GetRsult()
      {
         //Implementation Code
      }
   }

   [ServiceContract]
   public interface IClientService
   {
      [OperationContract]
      void AddTask();

      [OperationContract]
      void KillTask();

      [OperationContract]
      void GetRsult();
   }
}
```

Figure 3. Sample C# (C-Sharp) code of WCF-based Client Service

As noted in previous code sample, WCF follows SOA standards as it mandates developers to write both service contracts (interfaces) and implementation code. Service contract is defined by declaring one or more interfaces and marking them with *ServiceContract* attribute. Also,

service operations must be marked with *OperationContract* attribute to allow consumers to use them. Both ServiceContract and OperationContract attributes could be configured with a number of properties to control the structure and behavior of defined services and operations respectively.

Figure 4 illustrates a configuration file that lists settings of a self-hosted version of a WCF-based *Client* service with the three aforementioned endpoints.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <system.serviceModel>
    <bindings />
    <client />
    <services>
      <service behaviorConfiguration="WCFClientService.ClientServiceBehavior"
        name="WCFClientService.ClientService">
        <endpoint
address="http://localhost:8732/Design_Time_Addresses/WCFClientService/ClientService/"
        binding="basicHttpBinding" name="basicHttpBinding"
contract="WCFClient.IClientService" />
        <endpoint
address="http://localhost:8731/Design_Time_Addresses/WCFClientService/ClientService/"
         binding="wsHttpBinding" name="wsHttpBindingEndpoint"
contract="WCFClientService.IClientService">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding" name="mexHttpBindingEndpoint"
         contract="IMetadataExchange" />
        <endpoint
address="net.tcp://localhost:8001/Design_Time_Addresses/WCFClientService/ClientService/"
         binding="netTcpBinding" bindingConfiguration="" name="netTcpBindingEndpoint"
         contract="WCFClientService.IClientService">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <host>
          <baseAddresses>
            <add
baseAddress="http://localhost:8731/Design_Time_Addresses/WCFClientService/ClientService/
" />
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="WCFClientService.ClientServiceBehavior">
          <serviceMetadata httpGetEnabled="True"/>
          <serviceDebug includeExceptionDetailInFaults="False" />
```

```
      </behavior>
    </serviceBehaviors>
  </behaviors>
 </system.serviceModel>
</configuration>
```

Figure 4. Configuration File of One WCF-based Client Service

Figure 5 illustrates a sample code for *Executor* service that allows executors to get jobs to process them and send back returned result sets. Figure 6 illustrates a configuration file that lists settings of a WCF-based executor service.

```
Using System;
using System.ServiceModel;

namespace WCFExecutorService
{
   public class ExecutorService : IExecutorService
   {
      public void GetTask()
      {
         //Implementation Code
      }

      public void SendResult()
      {
         //Implementation Code
      }
   }

   [ServiceContract]
   public interface IExecutorService
   {
      [OperationContract]
      void GetTask();

      [OperationContract]
      void SendResult();
   }
}
```

Figure 5. Sample C# (C-Sharp) Code for a WCF-based Executor Service

It is worth mentioning that configuration files used to store services' settings know nothing about exposed functions. These files are responsible for applying a general schema to available services while being invoked. Settings stored in configuration files may include transport protocol, encodings, security settings, and transaction support. Using configuration files add more flexibility and ease-of use by exempting service providers from exhaustive management for each single operation.

Certainly, the aforementioned functions of both services need more elaboration about their implementation logic, and input/output parameters. However, this study focuses only on introducing the concept of using a flexible SOA-based grid computing architecture. This architecture allows creation and configuration of services with more than one specific technology or protocol. Furthermore, constructed services could be extended by adding more

functions to meet other needs according to implementation scenarios in which they are going to be applied.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <system.serviceModel>
    <bindings />
    <client />
    <services>
      <service behaviorConfiguration="WCFExecutorService. ExecutorServiceBehavior"
name="WCFExecutorService. ExecutorService">
        <endpoint
ddress="http://localhost:8734/Design_Time_Addresses/WCFExecutorService/ExecutorService/"
        binding="basicHttpBinding" name="basicHttpBinding" contract="WCFExecutor.IExecutorService" />
        <endpoint
address="http://localhost:8733/Design_Time_Addresses/WCFExecutorService/ExecutorService/"
         binding="wsHttpBinding" name="wsHttpBindingEndpoint"
contract="WCFExecutorService.IExecutorService">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding" name="mexHttpBindingEndpoint"
         contract="IMetadataExchange" />
        <endpoint
address="net.tcp://localhost:8002/Design_Time_Addresses/WCFExecutorService/ExecutorService/"
         binding="netTcpBinding" bindingConfiguration="" name="netTcpBindingEndpoint"
         contract="WCFExecutorService.IExecutorService">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <host>
          <baseAddresses>
            <add
baseAddress="http://localhost:8733/Design_Time_Addresses/WCFExecutorService/ExecutorService/" />
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="WCFExecutorService. ExecutorServiceBehavior">
          <serviceMetadata httpGetEnabled="True"/>
          <serviceDebug includeExceptionDetailInFaults="False" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Figure 6. Configuration File of WCF-based Executor Service

5. Advantages

The proposed architecture seems to inherit all advantages of grid computing in addition to SOA ones including:

- **Easy to Setup and Administer:** It takes almost nothing to define and manage WCF services in .NET environment. For instance, once the logic of services defined in code, they can be easily managed in configuration files to meet unforeseen needs. In other words, this option offers higher levels of dynamicity that allows implementers to meet different needs.
- **Easy to Use:** Our proposal tries to offer simple services with a set of functions that allow easy access to underlying grid resources.
- **Loose-Coupling:** Because underlying resources are wrapped in terms of services, clients and executors are not obliged to use specific technologies or protocols to access available grids.
- **Interoperability:** Available services are exposed with different endpoints allowing different client applications (both homogonous and heterogonous) to bind to them. Moreover, this model enables consumers to have access to grid resources from both intranet and internet.
- **Flexibility:** Each of offered services uses different technology to cover various needs. For instance, NetTcpBinding enables highest performance possible for .NET-to-.NET communication by using binary format for data being transmitted, whereas, BasicHttpBinding uses XML format to allow interoperability with non .NET clients.
- **Security:** It is guaranteed by ability to construct and offer a specific endpoint that is suitable to each kind of clients. For instance, LAN clients and executers use less secure endpoints because they are already trusted, whereas, those from internet have only access to grid resources through secured endpoints.
- **Scalability:** Other grids could be easily integrated with our architecture through WSFederationHttpBinding endpoints to form a larger and more powerful grid. Such grid might be utilized in complex scenarios that require high performance computing (HPC).
- **Applicability:** As a grid computing architecture, it could be easily utilized in all fields that require HPC. This includes (but not limited to): biomedical engineering; complex mathematical experiments; financial and economic analysis; computational modeling; simulations; and Graphics and Video rendering.
- **Easy to Upgrade:** Using WCF method enables implementers to define new endpoints that use both system-provided and custom-made bindings to meet future requirements. For instance, implementers might decide to allow cross-process communication by defining a new endpoint that uses *NetNamedPipeBinding*. Also, it is easy to allow queued messaging by exposing an endpoint that uses *NetMsmqBinding*.

6. Conclusion and Recommendation

In our paper, we proposed an architecture that leverages SOA concepts in grid computing. This architecture does not stop at XML Web Services like others do as an implementation technology for realizing SOA-based grid computing. Therefore, we utilized capabilities of ESB by using WCF technology to offer different endpoints each with different binding settings. This architecture offers great levels of flexibility, simplicity, and interoperability that are needed to allow integration between grid resources and its clients and executors.

At this end, we would like to remind readers that our work does not mean to supersede other grid computing frameworks or standards. Rather, it means to encourage other researchers including those who built available grid computing frameworks and standards to leverage our methodology in their tools. This would allow grid computing programmers to have stronger frameworks that enable them to build more efficient and powerful grids.

7. References

[1] I. Foster, C. Kesselman, S. Tuecke, “The anatomy of the grid enabling scalable virtual organizations”. The International Journal of High Performance Computing Applications, 2001, 15(3), pp. 200-222.

[2] A. Chien, B. Calder, S. Elbert, and K. Bhatia, Entropia, “Architecture and Performance of an Enterprise Desktop Grid System”, Journal of Parallel and Distributed Computing, Volume 63, Issue 5, Academic Press, USA, May 2003.
[3] B. Jacob, M. Brown, K. Fukui, N. Trivedi, "Introduction to Grid Computing", IBM Redbooks, December 2005.
[4] M. Litzkow, M. Livny, and M. Mutka, Condor, “A Hunter of Idle Workstations”, Proceedings of the 8th International Conference of Distributed Computing Systems, January 1988, San Jose, CA, IEEE CS Press, USA, 1988.
[5] L. F. G. Sarmenta, “Volunteer Computing”, Ph.D. thesis. Massachusetts Institute of Technology, March 2001. http://www.cag.lcs.mit.edu/bayanihan.
[6] N. Nisan, S. London, O. Regev, and N. Camiel, “Globally Distributed Computation Over the Internet: The POPCORN Project”, International Conference on Distributed Computing Systems, May 26 - 29, 1998, Amsterdam, The Netherlands, IEEE CS Press, USA, 1998.
[7] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, SETI@home, “An Experiment in Public-Resource Computing”, Communications of the ACM, Vol. 45 No. 11, ACM Press, USA, November 2002.
[8] A. M. Riad, Q. F. Hassan, “Service-Oriented Architecture (SOA) – A New Alternative to Traditional Integration Methods in B2B Applications”, Journal of Convergence Information Technologies, Volume 3, Number 1, March 2008.
[9] A. M. Riad, A. E. Hassan, Q. F. Hassan, “Leveraging SOA in Banking Systems’ Integration”, Journal of Applied Economics Science”, Volume III, Issue 2 (4), Summer 2008.
[10] Q. F. Hassan, “Aspects of SOA: An Entry Point for Starters”. Annals. Computer Science Series Vol. 7, Issue 2, November 2009, http://anale-informatica.tibiscus.ro/download/lucrari/7-2-12-Hassan.pdf.
[11] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, T. Newling, “Service-Oriented Architecture and Web Services”, IBM Redbooks. April 2004.
[12] L. F. G. Sarmenta, S. J. V. Chua, P. Echevarria, J. M. Mendoza, R. R. Santos, S. Tan, R. P. Lozada, “Bayanihan Computing .NET: Grid Computing with XML Web Services”, pp.434, Second IEEE International Symposium on Cluster Computing and the Grid (CCGRID'02), 2002.
[13] L. T. Yang, M. Guo, “High-Performance Computing: Paradigm and Infrastructure”, L. T. Yang, M. Guo, Chapter 21, pp. 403-429, Wiley, December 2005.
[14] R Buyya, K. Nadiminti, "Enterprise Grid Computing: State-of-the-Art", Enterprise Open Source Journal, Thomas Communications Inc, Dallas, Texas, USA, March/April 2006, pp. 19-22.
[15] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke, “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration”. Draft, June 2002. http://www.globus.org/research/papers/ogsa.pdf.
[16] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, J. Von Reich, “The Open Grid Services Architecture, Version 1.0”, January 2005, Global Grid Forum (GGF), http://forge.gridforum.org/projects/ogsa-wg.
[17] D. A. Chappell, “Enterprise Service Bus”, O’Reilly, 2004.
[18] "Windows Communication Foundation Architecture Overview", Microsoft Corporation, March 2007, http://msdn2.microsoft.com/en-us/library/aa480210.aspx.
[19] C. Vasters, "Introduction to Building Windows Communications Foundation Services", September 2005, http://msdn2.microsoft.com/en-us/library/aa480190.aspx.
[20] C. McMurtry, M. Mercuri, N. Watling, M. Winkler, “Windows Communication Foundation Unleashed”, SAMS Publishing, 2006.
[21] M. P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions", Proceeding of the Fourth International Conference on Web Information Systems Engineering, 2003.
[22] “Global XML Web Services Specifications: Web Services Specifications Index Page”, http://msdn.microsoft.com/en-us/library/ms951274.aspx
[23] S. Gupta, “Performance Comparison of Windows Communication Foundation (WCF) with Existing Distributed Communication Technologies”, February 2007, http://msdn.microsoft.com/en-us/library/bb310550.aspx.

[24] A. M. Riad, A. E. Hassan, Q. F. Hassan, "Investigating Performance of XML Web Services in Real-Time Business Systems", Journal of Computer Science and System Biology, Vol.2, Issue 5, September-October 266-271, doi:10.4172/jcsb.1000041, http://www.omicsonline.com/ArchiveJCSB/2009/October/01/JCSB2.266.pdf.

Biography

Dr. Alaa M. Riad is a Professor and a Head of Information Systems Department in Faculty of Computers and Information Systems, Mansoura University, Egypt. He has received a *PhD*. in Electrical Engineering in 1992 from Mansoura University, Egypt, and an *MS* in Electrical Engineering in 1988 from Mansoura University, Egypt**. Dr. Riad** has authored and coauthored many research papers in published journals. He also has supervised many Master and Doctorate studies.

Dr. Ahmed E. Hassan is an associate professor of Computer Engineering at Mansoura University, Egypt. He has received a *PhD* in Computer Engineering from West Virginia University, USA, and an *M.S* in Computer Engineering from Stevense Institute of Tech NJ, USA; he also got an *M.S* of Artificial Intelligence from Mansoura University, Egypt. **Ahmed E. Hassan** is IEEE Member since 1998, IEEE Computer Society since 1999, IEEE Education Society since 2002, IEEE Computational Intelligence Society since 2002, ACM Member since 2002 and ACM Education
Society since 2002.

Qusay Fadhel Hassan is a senior software engineer and systems' analyst in Department of State-USAID in Egypt. He performs several tasks in mid/large systems including analysis, design, technical architecting, and development. He has a *bachelor's* and *master's* degrees in Information Systems from Faculty of Computer Sciences and Information Systems, Mansoura University, Egypt. Qusay is now a *PhD* student in the Faculty of Computer Sciences and Information Systems, Mansoura University, Egypt. He has authored and coauthored a number of papers that have been published in different refreed magazines and journals. His research interests include SOA, Web Services, Distributed Systems, EAI, Grid Computing, and Cloud Computing.