2005 International Command and Control Research and Technology Symposium

The Future of Command and Control

PEER-TO-PEER DISCOVERY: A KEY TO ENABLING ROBUST, INTEROPERABLE C2 ARCHITECTURES

Topic: C4ISR/C2 Architecture

David Heddle	**Ray C. Prouty**	**Kurt Kalbus**
Chief Engineer	Chief Engineer	Sr. Software Developer
david.heddle@sparta.com	ray@sparta.com	kurt.kalbus@sparta.com

15 March 2005

SPARTA, Inc.

13400 Sabre Springs Pky, #220

San Diego, CA 92128

Phone: (858) 668-3570

PEER-TO-PEER DISCOVERY: A KEY TO ENABLING ROBUST, INTEROPERABLE C2 ARCHITECTURES

Abstract

Net-Centric Operations (NCO) require highly distributed data, applications and personnel across the military Services and agencies. The vision is that data will seamlessly pass between multiple levels of security as warfighters search for and publish/subscribe to services and data. At the center of this new enterprise architecture is the *discovery of services*. Discovery is one of the "core services" identified for the Global Information Grid (GIG) and is an essential element for legacy applications to migrate from stovepipes to services. It also enables runtime integration and self-assembling networks, which are critical for ad hoc communities of interest (COIs). The most challenging problem in Discovery is in the discovery of *services* (vice people or data), which relies on the technology of Universal Description and Discovery Integration (UDDI) registries.

In this paper, we discuss the gap between vision and reality and describe our research and testing of technology options for Discovery. We begin with a description of the three most common discovery methods (e.g., centralized, decentralized and semi-centralized). We then address interoperability among various UDDI vendors and application program interfaces (APIs) and identify the strengths and weaknesses of each discovery method. Finally, we recommend an approach for a near-term, robust system.

1.0 Introduction

The revolution in Net-Centric Operations (NCO) requires ready access to the highly distributed data, applications and personnel across the military services, agencies and coalition partners. The vision for the near-term is that data will seamlessly pass between multiple levels of security as warfighters search and publish/subscribe to services and data. At the center of this new enterprise architecture is the *discovery of services* - the key enabler for a Service Oriented Architecture (SOA). In this paper, we begin to identify options for creating a net-ready Command and Control (C2) architecture with decentralized management and self-adjusting networks - a key enabler to NCO.

2.0 Organization of this Paper

This paper will begin by describing the role of the discovery service in the GIG. We then present various approaches to discovery and discuss the design and implementation of a peer-to-peer foundation. After that, we show results of testing the foundation. Finally, we discuss future development of the peer-to-peer foundation and present an overall summary.

3.0 Discovery's Role in the GIG

One of the central themes of the GIG is the ability to employ Service Oriented Architectures (SOAs)[1]. The use of SOAs enables loosely coupled applications, service reuse, flexibility in system design and the ability to rapidly assemble applications/solutions in an ad-hoc fashion. Some of the benefits that can be achieved by this approach are: shorter development cycles, adaptability to changing missions, and faster access to critical data. The key (or enabling technology) to all of this is the ability to discover *relevant* data and services in a timely fashion.

The five major components of Discovery on the GIG are: discovery of services, data asset discovery, people discovery service, organizations discovery service and the registration service.

The ***discovery of services*** is a UDDI[2]-like service to find net offerings including applications. Just like the people discovery service (below), this service will allow a number of concurrent consumers to create concurrent searches and will support a data store service offering records.

The ***data asset discovery service*** allows one-off data and recurring data to be discovered. (Google is a well-known example of a data asset discovery service.)

---

[1] See, Global Information Grid Core Enterprise Services Strategy, http://www.defenselink.mil/nii/doc/

[2] Universal Description and Discovery Integration (UDDI), http://www.uddi.org/

The ***people discovery service*** searches and retrieves matching Department of Defense (DoD) personnel including civilians- most likely based on the common Lightweight Directory Access Protocol (LDAP)[3].

The ***organizations discovery service*** is similar to the people discovery service; it allows organizations and roles to be discovered – and is also likely to be LDAP based. And, like the data asset discovery service, this service will allow a number of concurrent technical consumers to create concurrent searches. It will also support a data store of indexed organization records.

The ***registration service*** allows consumers to register for dynamic enterprise content. This service will record the consumer's registration for data content in the enterprise within seconds.

4.0 Approaches to Discovery

As part of the description of common methods, we address the DoD's current discovery approach[4] which relies on a centralized discovery method by maintaining an index of resources at a centralized server. It is the easiest to implement, but has a large drawback in that the centralized index server is a single point of failure and requires a common vendor among registries to insure interoperability with the discovery Application Programmer Interfaces (APIs) or requires that all applications support different UDDI vendors.

The other end of the spectrum of discovery approaches is a network of "equal" peers where resources reside (e.g., peer-to-peer (P2P)). Queries for resources are propagated to neighboring nodes until a match is found and the resource is returned to the requesting peer. The Gnutella[5] protocol, used in commercial applications, is a basis for many

---

[3] Lightweight Directory Access Protocol, http://www.gracion.com/server/whatldap.html

[4] Defense Information Systems Agency (DISA) Reports, Available at https://ges.dod.mil.

[5] Gnutella, http://gtk-gnutella.sourceforge.net/index.php?page=faq

decentralized P2P networks. Various music file sharing networks, as well as a general purpose search engine, have been developed based on the Gnutella protocol. The main advantage of a decentralized P2P system is reliability. If some peers on the network are unavailable, the network functions without degradation, except for the loss of resources at the unavailable nodes. The disadvantage of a fully decentralized P2P architecture is that in actuality, the computer hardware running the peer software is not always equal. Treating a slow machine with a dial up network connection equal to a fast machine with a broadband connection can lead to bandwidth problems.

As a potential near-term solution, adopting the best of both extremes, we detail a semi-centralized peer-to-peer (P2P) system, illustrated in Figure 1. We developed this concept in our Net-centric Environment for System Testing and Operational Research (NESTOR) project. Semi-centralized P2P systems consist of networks of peers where some of the more powerful peers are designated as "supernodes". Supernodes are used to cache resources from less powerful peers, called "leafs". When a request for a resource is propagated through the network, attempts are made to find and return a resource from a supernode to the requesting peer. This is a potentially attractive solution to the bandwidth problem encountered with fully decentralized P2P networks.

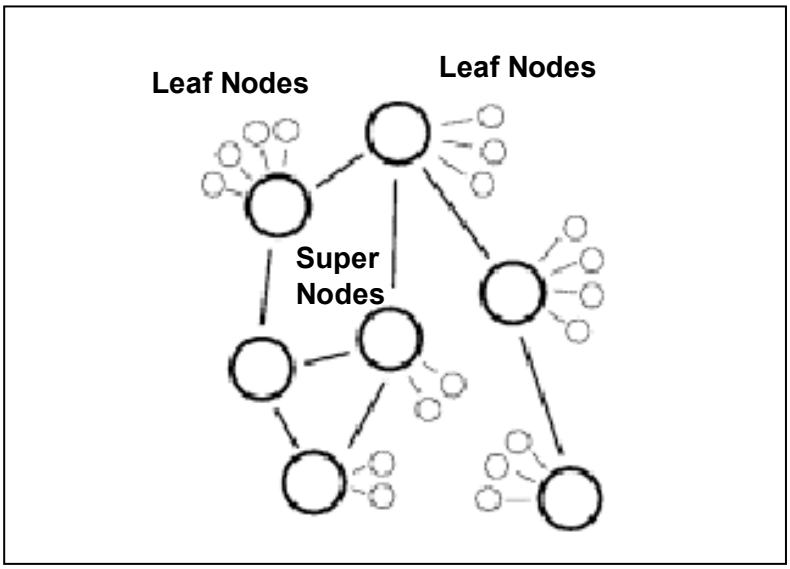

Figure 1: Semi-Centralized Peer-to-Peer Network

5.0 Designing a Peer-to-Peer Discovery Foundation

Our NESTOR discovery architecture is based on the LimeWire[6] open source code. Enhancements were made to perform keyword queries on distributed UDDI registries for Web Services. In our NESTOR implementation, illustrated in Figure 2, each node in the network is aware of one or more UDDI registry servers. When a peer receives a UDDI discovery request, it searches its known UDDI servers for a matching Web Service. If a match is found, information about the discovered Web Service (such as its Web Services Description Language (WSDL)) is returned to the requesting peer. In addition, the discovery request is further propagated to other peers on the network to search for more matches. One major benefit of this approach is that the discovery peers interface with their known UDDI servers via a vendor-independent API. In this way, UDDI servers from different vendors can be added to the network without change to the peer program code – a huge cost savings for the military. This attribute is highly desirable for interoperability among various US military organizations and, we contend, a requirement for interoperability with our coalition and allied partners.

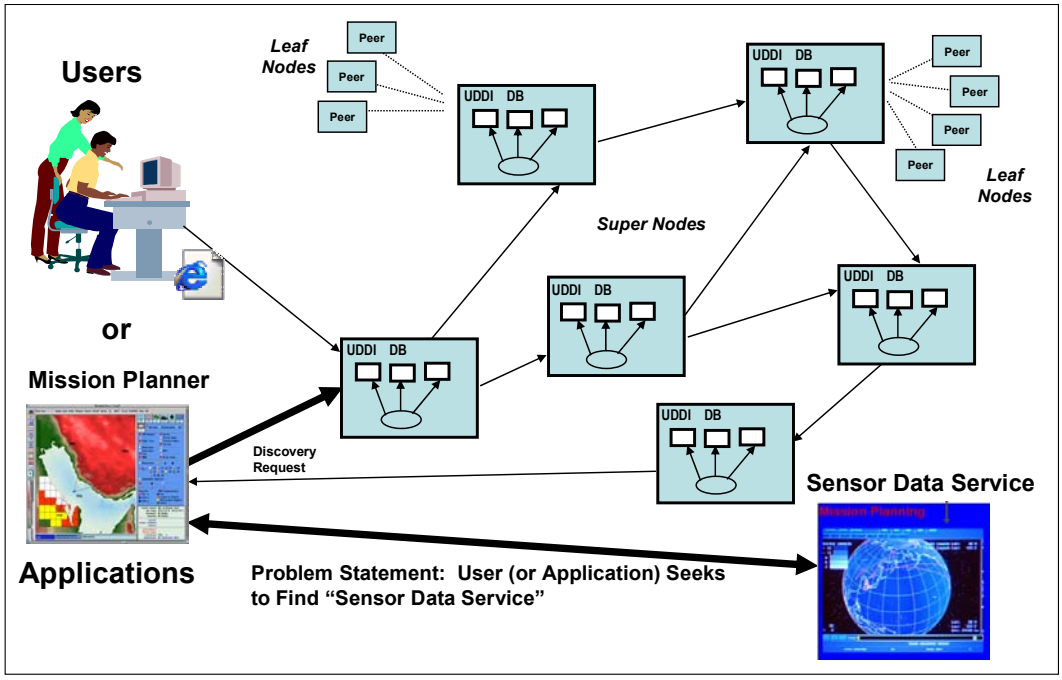

Figure 2: NESTOR Peer-to-Peer Discovery Network

[6] Limewire, www.limewire.com

The LimeWire approach was chosen for two primary reasons. First SPARTA, Inc has successfully used this code in a previous application. Secondly, the basic LimeWire code provides a well-tested, robust infrastructure for peer-to-peer communication inside and outside firewalls, as well as implementing the concepts of super-nodes and leaf nodes in a platform independent manner.

6.0 Implementing a Peer-to-Peer Discovery Foundation

Version 4.0.6 of the LimeWire Java open-source code was used as a starting point. The code consists of two main packages: **core** and **gui**. The **core** package implements the peer-to-peer infrastructure and handles the network file search and transfer. The **gui** package provides a graphical user front end and interfaces with the **core** back end. For our purposes, we dispensed with the **gui** package and imported the **core** package into an Eclipse (v2.1.3) project.

The first change to the **core** package consisted of removing some of the peer 'bootstrap' functionality to handle the requirement that any peer-to-peer system must know about neighboring peers in order to start communication with the network. The LimeWire **core** package uses both a configuration file and a list of internally specified Uniform Resource Locators (URLs) in order to 'bootstrap' the communication. We stripped out the internal URLs and modified the configuration file to eliminate the "well-known" Gnutella caches from consideration since those peers will be generic LimeWire peers and will not know how to handle Discovery request messages.

The second change to the code involved implementing a Discovery request message. The LimeWire code defines a *QueryRequest* message class which contains, among other things, keywords for network file searches. It is the information in the *QueryRequest* that is propogated throughout the peer-to-peer network in order to search for matching file names on peer nodes. The new *DiscoveryRequest* class is basically identical to the *QueryRequest* with a different functional header. In this way, *DiscoveryRequests* will take advantage of the LimeWire infrastructure for propagating messages throughout the

peer-to-peer network. The real difference is what happens when a *DiscoveryRequest* is received by a peer and what happens when the originating peer receives the reply back from its original *DiscoveryRequest*.

Changes were made to the base LimeWire code to recognize an incoming *DiscoveryRequest* message. When such a message is detected, instead of searching a directory for a filename (as in the case of a *QueryRequest*), the peer invokes a vendor *independent* UDDI API to search for Web Services matching the provided keywords. (See section 7.0). If a Web Service is found, a *DiscoveryReply* message is formulated and sent back to the originating peer. This reply currently contains the Web Service name, description and WSDL Uniform Resource Identifier (URI). Whether or not Web Services are found, the *DiscoveryRequest* is then forwarded to neighboring peers.

When the originating peer receives the *DiscoveryReply* message a new query reply handler is invoked to parse the *DiscoveryReply* message and extract the Web Service names, descriptions and WDSL URIs.

7.0 Vendor Independent UDDI API

There are many UDDI registry products available from several vendors. Some UDDI registries, such as those from Systinet[7] and Acumen Technologies[8], are standalone products. Other vendors, such as WebLogic and Sun, integrate a UDDI server into their Application Server products.

Most vendors provide a Java-based UDDI client API so that Java applications can be written to access their UDDI registry. In addition, there are standards such as JAXR that define Java UDDI APIs. Vendors generally try to make their UDDI registries conform to standards such as JAXR, so that an implementation of JAXR, such as that from Sun, should theoretically interface to a particular vendor's UDDI.

---

[7] Systinet UDDI Registry, http://www.systinet.com/products/sr/overview

[8] UDDI, www.acumentechnologies.com, http://www.uddi.org/solutions.html#Acumen_Technology

Despite vendor claims, our research and experimentation showed that no single UDDI API would always successfully interface with a given UDDI registry. Depending on the release level of the API, the release level of the UDDI registry and even the version of Java used to compile and run the client application, the client might or might not be able to publish to or query a particular UDDI. The results of our experimentation are shown below in Figure 3.

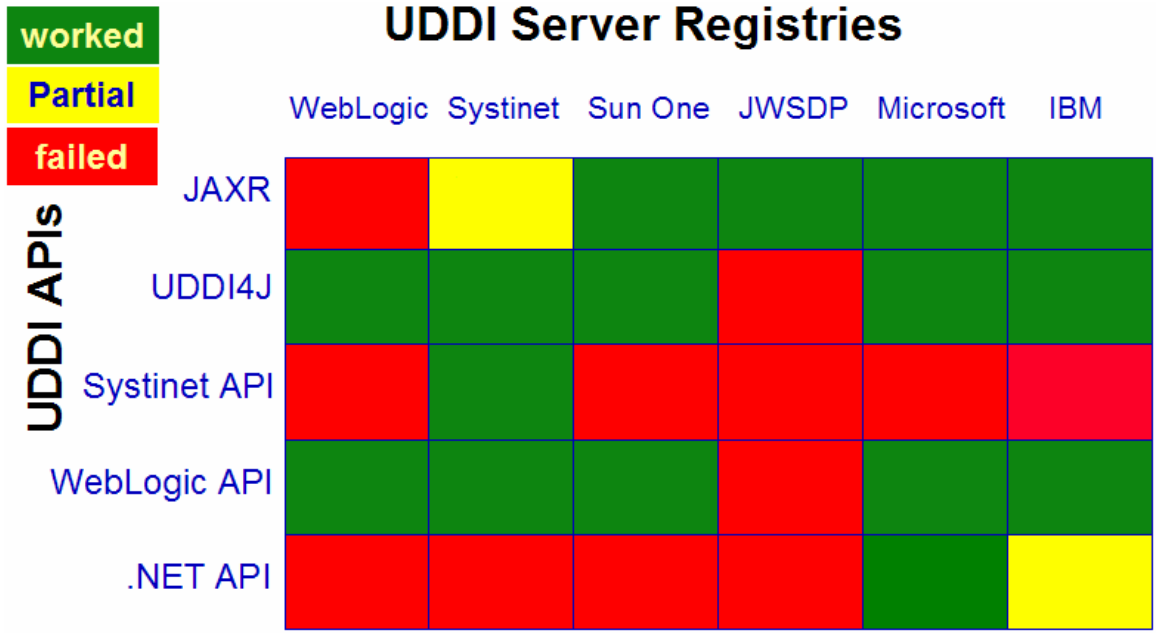

Figure 3: Vendor Interoperability with UDDI APIs and Server Registries

One of the key goals of our Discovery service is to be able to discover Web Services from any brand of UDDI registry – enabling a net-ready key capability. For this reason, we developed a vendor independent Java-based UDDI API. The API is essentially a "wrapper" around lower level APIs that have been proven to work with various UDDI registries. This vendor independent API is invoked by the Discovery peers (as described in section 6.0) and has been tested with WebLogic, Sun and Systinet UDDI registries.

8.0 Testing the Peer-to-Peer Foundation

As part of our NESTOR project, a small test network of Discovery peers and UDDI registries was setup. The architecture for this test is shown in Figure 4. The following table summarizes the underlying network:

Hostname	IP address	Operating System (OS)	UDDI registry	UDDI Host
Potato (San Diego)	157.185.24.29	Win XP Pro	Sun	Potato
Chakotay (San Diego)	157.185.24.253	Win 2003 Serv	WebLogic	Chakotay
Kirk (Hampton)	157.185.52.20	Win 2003 Serv	Systinet	Kirk
Watergate (Rosslyn)	157.185.86.236	Win 2000	Sun	Potato

The first two machines reside behind the same firewall in SPARTA San Diego. The other two machines reside behind two separate firewalls at locations on the East Coast (i.e., Rosslyn/Arlington and Hampton, Virginia). There are three different UDDI registries running on the network. The Watergate machine does not have a UDDI registry, but the Discovery Peer running on that machine has been configured to query the Sun UDDI registry on one of the SPARTA San Diego machines. Several Web Services were registered in each of the UDDI registries. In particular, a Web Service with the keyword "missile" was registered in all three UDDI registries.

Discovery peers were installed and started on each of the four machines in the network. A simple Java GUI front end was written to interface with the Discovery peer on the

computer known as 'Potato'. (This GUI front end logically replaces the **gui** package provided with the default LimeWire code, as described in section 6.0).

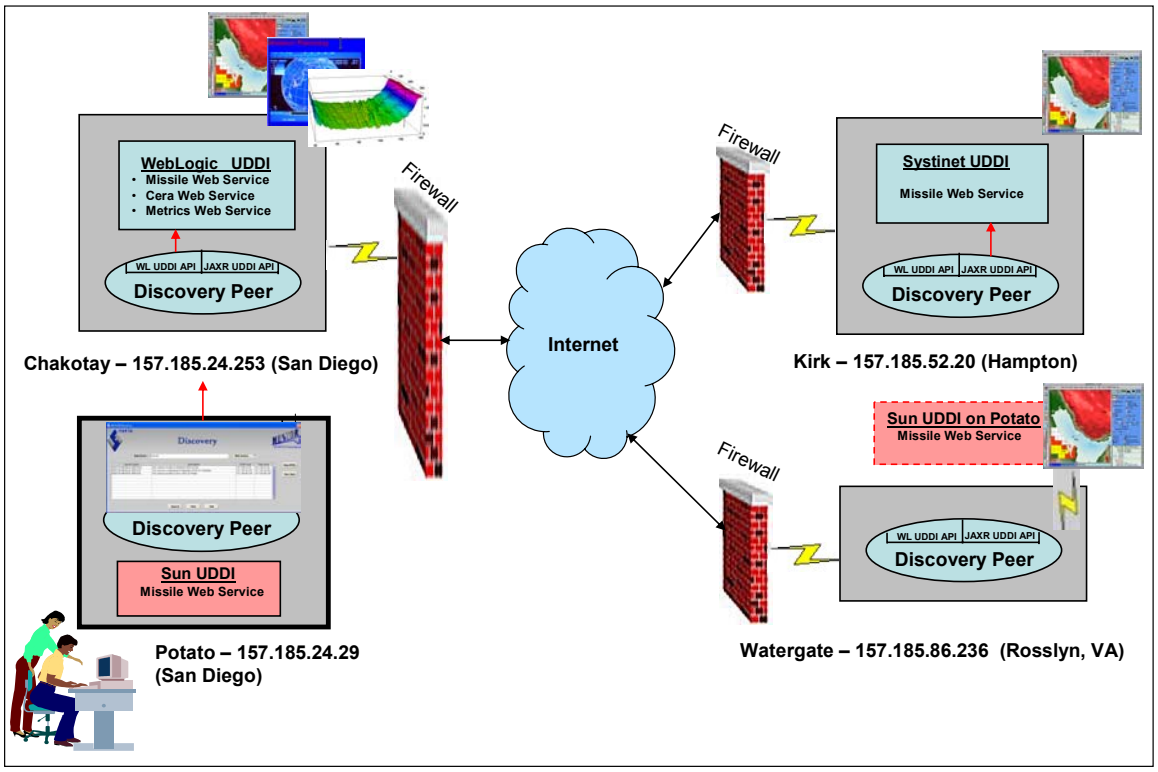

Figure 4: Discovery Proof-of-Concept Test

The LimeWire code checks a bootstrap configuration file (gnutella.net) to initiate communication with the peer-to-peer network. In this test network, initially the gnutella.net file for the Discovery peer on Potato was empty. The gnutella.net file on Chakotay was initialized to connect to the peer on Watergate. The peer on Watergate was initialized to connect to the peer on Kirk. The important point here is that the originating peer on Potato initially had no direct knowledge of any of the other peers running on the test network.

The keyword "missile" was entered into the GUI client on Potato. The GUI client triggered the Discovery peer on Potato to initiate a *DiscoveryRequest* for Web Services containing the keyword "missile".

The LimeWire infrastructure first sends out the *DiscoveryRequest* to all the peers on the same subnet via multicast. Thus, the Chakotay peer receives the *DiscoveryRequest* from

Potato and searches its WebLogic UDDI registry and finds the "missile" Web Service. It sends back a *DiscoveryReply* to the Potato peer.

The peer on Chakotay then forwards the *DiscoveryRequest* to it's known peers – in this case, the peer on Watergate, which discovers the "missile" Web Service in it's Sun UDDI registry (which actually resides on a different machine). The Watergate peer also sends a *DiscoveryReply* to the Potato peer. The same process occurs as the Watergate peer forwards the *DiscoveryRequest* to the Kirk peer.

Thus the *DiscoveryRequest* for "missile" which was initiated by the Potato peer resulted in the discovery of the three "missile" Web Services registered in the three UDDI registries on the test network, even though the gnutella.net file on Potato initially contained no information whatsoever about neighboring peers.

It should be noted that after receiving the three hits back from the peer network, the LimeWire code will record the IP addresses of the responding peers in the local gnutella.net file. Therefore subsequent requests will be sent directly to the peers that have been previously found to be "friendly".

This test proved several key points: (1) a heterogeneous UDDI registry system can be built to allow different Services, Agencies and Coalition enclaves to quickly join the net, (2) a semi-centralized peer-to-peer system can support Discovery using the LimeWire approach and (3) discovery requests can be handled across firewalls in a timely fashion.

9.0 Future Development of the Peer-to-Peer Discovery Foundation

The following areas have been identified as items for future Discovery foundation enhancements:

1) Development of a more robust browser-based front end to the Discovery service, similar to familiar Web search engines, such as Google.

2) Enhancements to the Discovery API, so that other applications can be more easily written to take advantage of the Discovery service.
3) Testing the network with a more heterogeneous mix of super-nodes and leaf-nodes, for example using machines that are connected to the network via dial-up.
4) Implementing a mechanism to gather metrics about performance and coverage of the Discovery peer-to-peer network.
5) Expanding the Discovery functionality to include more than just finding Web Services, such as querying LDAP registries for information about people.
6) Implement additional security mechanisms, such as group security protocols.

10.0 Summary

Discovery is recognized as a core service for the Global Information Grid. It is also one of the least understood of the core services. Our research demonstrates a concept that is able to negotiate firewalls and find actual, viable services based on a keyword search. The semi-centralized peer-to-peer architecture mitigates both the single point of failure problem of centralized architectures and the bandwidth bottlenecks that arise in heterogeneous fully decentralized systems. Finally, we describe a path for expanding and enhancing our foundation for enterprise-level testing and to demonstrate how this discovery implementation will provide a bridge among COIs and a foundation for secure cross-service and allied interoperability. This approach allows heterogeneous vendor solutions (i.e., no single vendor for UDDI must be chosen), which is very advantageous in acquisition and promotes rapid interoperability as we move to a complete net-ready GIG across all COIs.