A Taxonomy of Discovery Services and Gap Analysis for Ultra-Large Scale Systems*

Joe Hoffert, Shanshan Jiang, and Douglas C. Schmidt
Institute for Software Integrated Systems
Vanderbilt University
2015 Terrace Place
Nashville, TN 37212
{joseph.w.hoffert,shanshan.jiang,d.schmidt}@vanderbilt.edu

ABSTRACT

Timely discovery of services in ultra-large scale (ULS) systems plays a vital role in critical areas, such as national power grids, homeland security, and health care. This paper develops a taxonomy for classifying discovery services and presents an overview of existing discovery service technologies. It then classifies these discovery services using the taxonomy and performs a gap analysis for discovery services with respect to emerging ULS systems. Our results show that while discovery services are fairly mature and broadly applicable to today's systems much R&D remains to support emerging systems of ultra-large scale effectively.

Keywords

Discovery Services, Taxonomy, ULS Systems, Gap Analysis

1. INTRODUCTION

As distributed systems grow to encompass large numbers of components that provide different types of services, it becomes increasingly important to discover these services in a scalable and dependable way. Discovery services are part of many distributed computing technologies, such as CORBA [11], Jini [16], Web Services [3], and Universal Plug and Play [19]. This paper provides a taxonomy of various discovery services, evaluates existing discovery services using this taxonomy, and identifies gaps in the ability of current discovery services to meet the needs of emerging *ultra-large scale (ULS) systems*, such as the Global Information Grid (GIG) [1].

ULS systems are an emerging area of research and development [6]. They are characterized by such properties as decentralization; inherently conflicting, unknowable, and diverse requirements; continuous evolution and deployment; heterogeneous, inconsistent, and changing elements; erosion of the people/system boundary; normal failures; and new paradigms for acquisition and policy. Although these characteristics are evident in today's large-scale systems (such as the Internet, air traffic management systems, and power grid control systems), scale and inherent decentralized control will dominate in ULS systems.

As ULS systems continue to grow and mature the importance of discovery services that can handle the characteristics of these systems increases dramatically. ULS systems need to build upon existing legacy systems, which may not support the latest discovery service technology. Moreover, many discovery services have closed architectures that do not support other types of discovery services, which is inadequate for ULS systems.

ULS systems also require a great deal of flexibility in service discovery. For example, failures and disconnectivity will be commonplace, so alternatives for services must be located and negotiation of services must be conducted to use available resources effectively and support safety- and mission-critical systems. Quality of service (QoS) support will also be essential for ULS systems since *when* a service is found is often as critical as *which* service is found, *i.e.*, a less-than-ideal service found in a bounded amount of time may be better than an ideal service that is located too late. Research is therefore needed to understand the implications of the many dimensions in which ULS systems need support from discovery services.

The remainder of the paper is organized as follows: Section 2 describes a taxonomy of different categories of discovery service properties; Section 3 summarizes various discovery service technologies and maps the surveyed discovery services to the taxonomy categories in Section 2; Section 4 performs a gap analysis of discovery services needed for ULS systems running in mobile ad hoc networks; and Section 5 presents concluding remarks that describe the research needed to address the gaps described in Section 4.

2. A TAXONOMY OF DISCOVERY SERVICES

Discovery services can be used for many purposes and in many contexts, such as powering up a laptop and looking for a nearby printer, networking small wireless devices relevant to a single person, finding music on the Internet, or locating the closest emergency room in a disaster recovery operation. This section presents a taxonomy that categorizes the properties common between discovery services and the properties that differentiate them so that existing discovery services can be compared and contrasted meaningfully.

*This work is supported in part by the AFRL/IF Pollux project and NSF TRUST.

ACMSE 2007, March 23-24, 2007, Winston-Salem, N. Carolina, USA

Table 1 outlines the categories in this taxonomy.

Categories	Alternatives
Request granularity	fine- vs. coarse-grained
Structure discovery	richly vs. simply structured
Resource constraints	constrained vs. unconstrained
Service lookup strategy	named vs. property-based
Network scope and type	LAN vs. WAN
Heterogeneity	supported vs. not supported
Discovery QoS	supported vs. not supported
Service negotiation	supported vs. not supported

Table 1: Taxonomy of Discovery Service Categories

The categories in this taxonomy are described as follows:

• **Request granularity.** This category includes support for fine-grained requests and coarse-grained requests. Fine-grained discovery allows for very detailed requests, which can include dynamic properties of the desired service itself. An example for this property would be discovering the closest color laserjet printer with a certain DPI resolution. Coarse-grained querying, in contrast, only allows specification of general high-level capabilities, such as simply locating a printer. It has the disadvantage of decreased flexibility compared to fine-grained queries, but the advantage of easier development and use.

• **Structure discovery.** This category distinguishes between discovery of objects that are richly structured versus discovery of objects that have a simple structure. Richly structured objects can provide functionality that users can access programmatically via an API. An example is object references returned by distributed computing middleware, such as CORBA or Java Remote Method Invocation (RMI). Supporting the discovery of richly structured objects simplifies the development of distributed applications. This functionality can increase the overhead and complexity of a discovery service, however, since it must manage information about the kinds of objects it supports, as well as how it presents these objects so users can interact with them appropriately.

Conversely, simple structured objects, such as text or bitmapped graphical images, are homogeneous and provide little or no functionality themselves, which greatly limits user manipulation of these objects. These limits are not a concern, however, for certain types of applications, such as a web browser on a PDA. The advantage of simple structured objects is that they can be supported easily and the discovery service itself need know little or nothing about the objects themselves.

• **Resource constraints.** This category differentiates discovery services based on the typical resources that users of the service will have. Some discovery services are designed for highly resource constrained platforms, whereas others are not. Examples of resource constrained nodes include PDAs, cell phones, and remote sensors, as opposed to relatively resource-rich nodes, such as laptops, desktops, and servers. Whether a user is resource constrained and/or is deployed in an environment that is resource constrained affects the development of the discovery service since the discovery service will, in part, need to participate in the same environment or otherwise be constrained itself.

For instance, if memory is at a premium then coarse-grained queries might only be supported since fine-grained queries imply the use of some type of database. At the very least, fine-grained queries must be highly optimized for a particular environment and would not provide much flexibility in the types of services they support. These optimizations, in turn, limit the flexibility of queries supported.

• **Service lookup strategy.** Another differentiation between discovery services involves how users identify the service they want to discover. Some discovery services use names to identify services, which is analogous to the "White Pages" where telephone numbers are indexed according to people or businesses names. This type of discovery can be implemented efficiently by using the name as a key to quickly locate the appropriate service. For example, a laptop could lookup a printer by its name.

Other discovery services offer property-based service lookup akin to the "Yellow Pages" that are indexed by properties rather than names. In this case, users need not know the name of a particular service, but instead provide particular properties. For example, a laptop could also look up a printer by its properties, such as a color printer with 600 DPI and more than 200 sheets in its paper tray.

Name-based lookup is generally more efficient than property-based lookup. It is also more limited, however, since desired properties of the retrieved service can only be implied by some means outside of the discovery service itself. Moreover, the requester of a named service must be "bootstrapped" in some fashion so that it somehow knows the name.

• **Network scope and type.** Some discovery services are designed for local area networks (LANs). In this case, custom network protocols can be used to communicate between users and the discovery service. For example, a discovery protocol developed atop hardware multicast is often an efficient way to locate services on a LAN.

Other discovery services are designed for wide area networks (WANs). In this case, more sophisticated protocols must be used to locate services globally, which requires larger, more complex global addresses and more state to keep track of efficient routes through large network topologies. For example, a WAN-based discovery service may need to develop spanning trees to minimize hops through the WAN. While discovery services developed for WANs will necessarily be more complex to handle a wide array of concerns, they can scale to a broader scope.

• **Degree of heterogeneity.** Some discovery services support heterogeneous discovery service technologies and bridge the differences between them. For instance, the client could issue a request with one discovery service technology (*e.g.*, the CORBA Trading Service) and the discovery service would be able to retrieve an appropriate service that registered itself with a different discovery service technology (*e.g.*, UDDI).

Some discovery services are designed to work with just one technology. For example, Jini is designed to work with Java RMI and leverages its ability to download bytecode from a service provider into the requesting client. The advantage of such a homogeneous design is that a discovery service can be optimized to take advantage of particular characteristics of that technology.

• **Discovery QoS.** Most discovery services provide clients with"best effort" strategies that provide no assurance when a service will be discovered and provided to clients. A discovery service that supports QoS, in contrast, could specify service level agreements for the services it provides. For ex-

ample, a discovery service client could designate that if a requested service was not located within a specified amount of time the "not-found" status would be returned.

• **Service negotiation.** A client of a service typically requests a particular service and the discovery service either returns (a reference to) that service or no service at all (*e.g.*, via the "not-found" status). A discovery service that supports negotiation of services could interact with the client to determine alternatives to the initial request should that request not be available.

The categories presented above extend earlier work by Vanthournout *et al.* [20], whose taxonomy helps system developers choose a discovery service appropriate for their applications. This paper, however, considers some discovery services different from theirs, including some that are still being researched (such as the Service Discovery Broker Engine [7] that provides a bridge between disparate discovery service technologies). We also enumerate different taxonomy categories that we use to classify discovery services, including service negotiation and QoS support for discovery, and evaluate discovery services in light of the requirements manifested by ULS systems, as discussed in Section 4.

3. DISCOVERY SERVICES OVERVIEW AND CLASSIFICATION

This section describes the discovery services that we surveyed and taxonomizes these services based on the categories presented in Section 2. We first briefly describe the discovery services we considered for this paper.

• **CORBA Naming Service.** The CORBA Naming Service [12] supports discovery of objects that implement various services keyed by names, akin to the "White Pages." After a reference to an object is obtained, operations on the object can be invoked to access the designated services.

• **CORBA Trading Service.** The CORBA Trading Service [10] supports discovery of objects that implement various services based upon various static or dynamic properties, akin to the "Yellow Pages." As with the CORBA Naming Service, a reference to an object is obtained and operations can be invoked on the discovered object.

• **Jini Lookup Service.** The Jini Lookup Service [17] provides a federated way for Java clients to discover services using Java RMI. The Java client makes a request to the Jini Lookup Service and specifies an interface. The lookup service then returns a Java object matching the interface or a proxy for a non-Java service. Since this discovery service is based on Java RMI a requested object (including its bytecode and state) can be downloaded to the client. Jini's use of Java RMI enables optimizations (such as caching of objects) and flexibility (such as downloading smart proxies to the client). It can also have undesirable side effects, however, such as increased latency and jitter when first transferring the object.

• **Data Distribution Service (DDS) discovery services.** The Real Time Innovation (RTI) DDS and PrismTech OpenSplice implementations of the OMG DDS [13] specification provide highly configurable QoS parameters for anonymous real-time publish/subscribe. These DDS implementations also provide peer-to-peer discovery services whereby *topics* (which provide a unique identifier for particular data items within the global data space) can be discovered by entities that are interested in those topics (*i.e.*, *data writers* and *data readers* which respectively write and read data). Discovery occurs in two steps: (1) the *domain participants* (which include the local entities that are grouped by a common *domain* or communication enclave) send out messages to discover each other using best-effort communication and (2) the domain participants exchange information about their data writers and data readers using reliable communication.

• **Simple Service Discovery Protocol (SSDP).** SSDP [19] is used by UPnP to allow controllers (*a.k.a. control points*) to find devices and learn about device capabilities. When devices first join a network they send out short messages advertising themselves that are multicast to a well-known address and port. The control points listen at this address and port and receive the synopsis of the device's capabilities. More detailed information can be retrieved via a URL that is supplied. Likewise, when a control point first joins the network it sends out messages that include a target or pattern used to match devices or services.

• **Service Location Protocol (SLP).** SLP [18] is a service discovery protocol that allows computers and other devices to find services in a LAN without prior configuration. SLP has been designed to scale from small, unmanaged networks to large enterprise networks. In SLP, a *user agent*, which is a software entity looking for appropriate services, emits a request message to query the types of services available. This message can be unicast or multicast and may be received by either a *service agent* or a *directory agent*. The service agent is the software entity that knows the location of one or more services and can therefore reply to the request. A dialog can occur between the user agent and the service agent if a service agent knows the location of a desired service. The optional directory agent is used as a centralized repository for the location of services, which can enhance performance since a user agent can unicast messages to a known directory agent to request a service as opposed to multicasting requests when no directory agent is available or known.

• **Bluetooth Service Discovery Protocol (SDP).** Bluetooth [2] supports the interconnection of a broad selection of wireless devices (*e.g.*, mobile phones, computers, and PDAs) using short-range wireless connections. The Bluetooth SDP creates boot up connections for devices via the *Logical Link Control and Adaptation Protocol (L2CAP)* layer and is considered orthogonal to the discovery protocol. Users can then send request messages either to a particular device (if it is known) or to the unknown devices in the *piconet* (which is the short-range, ad-hoc network automatically created, modified, and deleted as devices enter and leave radio range). These request messages can relay various types of requests (*i.e.*, listing of available services, query for a particular service with or without specific attributes).

• **Universal Description, Discovery, and Integration (UDDI).** UDDI [9] is a platform-independent, XML-based registry for businesses to list themselves and their services on the Internet. At its inception the targeted industry was business-to-business (B2B) service delivery. UDDI creates various service description templates and registers them in the *UDDI Business Registry* (UBR). Businesses then instantiate and fill in applicable templates with the services they support. These service descriptions are registered with the UBR, which gives each service and business registration a unique ID. Users can query the registry to discover desired services. As part of registration, businesses store in-

Discovery Service	Request Granularity		Structured Discovery?		Resource Constrained?		Named Service Lookup?		Network Scope		Heterogeneity?		Discovery QoS?		Service Negotiation?	
Options	**Fine**	**Coarse**	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**LAN**	**WAN**	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
CORBA Naming Service		X	X			X	X			X		X		X		X
CORBA Trading Service	X		X			X		X		X		X		X		X
Jini Lookup Service	X		X			X		X	X			X		X		X
Surveyed DDS Implementations	X[1]	X	X			X		X		X		X	X[2]			X
SSDP	X		X			X		X	X			X		X		X
SLP	X		X			X		X	X			X		X		X
Bluetooth SDP		X	X		X			X	X			X		X		X
UDDI	X		X			X		X		X		X		X		X
JXTA	X		X		X			X		X		X		X		X
Gnutella		X		X		X	X			X		X		X		X
Napster		X		X		X	X			X		X		X		X

[1] The DDS implementations surveyed provide coarse-grained request granularity when discovering participants and fine-grained granularity when discovering data readers and data writers.
[2] The DDS implementations surveyed provide limited quality of service support for discovery by specifying metatraffic transport priorities and restricting nodes that are able to connect.

Figure 1: Classification of Discovery Services

formation about themselves in the "White Pages," information about their category of business in the "Yellow Pages," and information about how other businesses should conduct business with them in the "Green Pages." UBRs can either be public (where any business can query for the service of another business) or private (which are used within organizations to advertise services).

• **JXTA.** JXTA [14] is a set of open-source XML-based protocols that allow connected devices on the network to communicate and collaborate with other connected devices. The intended devices range from cell phones and wireless PDAs to PCs and servers. JXTA supports a peer-to-peer (P2P) paradigm where a network overlay is formed to allow direct communication even if a peer is behind a firewall or a network address translation (NAT) service. Two classifications of peers are used when developing the network overlay: (1) an *edge peer*, such as a low-bandwidth device with transient connectivity, and (2) a *super-peer*, which is a proxy for other peers that would not otherwise communicate (*e.g.*, edge peers on different subnets or peers hidden behind a firewall or NAT service).

• **Gnutella.** Gnutella [5] is a resource sharing network used primarily to exchange files. It uses a P2P architecture that enables clients to also be servers. When a client initiates a query for a resource it contacts the list of peers that it knows. With the network overlay scheme Gnutella employs, if the contacted peers do not have the resource they forward the request onto the peers which they know. This forwarding continues until a hop count (*a.k.a.* time-to-live) value is reached. If the resource is found the provider peer contacts the client peer directly and the two can initiate the transfer of the resource.

• **Napster.** Napster [8] uses a P2P architecture for discovering songs that can be played with an MP3 device. Napster differs from Gnutella in that it uses a centralized registry with a well known IP address and port number that users query for song selections. Once the registry matches the queries the locations of the peers supplying the requested songs are provided to the client. Direct P2P transfer of audio files can then commence, with the client requesting files from peers (via Napster transfer or by some other means).

Figure 1 shows the classification of the discovery services outlined above using the categories described in Section 2.

4. ANALYZING GAPS IN DISCOVERY SERVICE SUPPORT FOR ULS SYSTEMS

This section presents a gap analysis of the properties summarized in Table 2 to indicate which discovery services capabilities must be enhanced to support the needs of emerging ULS systems. Figure 1 shows that the discovery services we surveyed in Section 3 have little or no support for three categories in our discovery service taxonomy: (1) heterogeneity, (2) discovery QoS, and (3) service negotiation. Although many conventional distributed systems do not need these

Categories	Need	Status
Heterogeneity	Support legacy services	Initial R&D
Discovery QoS	Mission-critical systems	Some COTS support
Service negotiation	Flexibility	Needs scaling-up R&D
Network scope and type	ULS systems	Needs scaling-up and intermittent connectivity R&D

Table 2: Gap Analysis Properties

discovery service capabilities, they are critically important for emerging ULS systems. Moreover, the requirements of ULS systems push certain properties (such as network scope and dynamic services) of conventional discovery service implementations beyond what they can currently handle, as discussed below.

• **Heterogeneity.** Many discovery services today only work with a single technology (*e.g.*, CORBA or Java) and do not support different types of underlying discovery protocols. Heterogeneity plays an important role for ULS systems, however, which must integrate legacy applications and technologies that use existing discovery services. The effort to migrate these systems to a single common discovery service or to reimplement existing functionality would be monumental and cost-prohibitive. Discovery services for ULS systems will therefore need to accommodate currently existing discovery services. The use of federations and gateways have been used in other areas (*e.g.*, networking protocols, enterprise service buses) and can provide architectural guidance to incorporate heterogeneous discovery services.

Some examples of work done to support heterogeneous discovery services include the Service Discovery Broker Engine [7], which provides (1) a gateway that locates services across heterogeneous discovery services (*e.g.*, Jini Lookup Service and SLP) and (2) federation of Service Discovery Brokers for scalability. Friday *et al.* [4] identify key limitations of existing discovery services for ubiquitous computing applications, enumerate requirements for effective support, summarize the design of an appropriate discovery architecture, and present the lessons learned from prototype development. We found no industry standards or commercial products, however, that support heterogeneity. The work that has been done can be categorized as initial research into this area.

• **Discovery QoS.** In ULS systems the right answer delivered too late will be the wrong answer. Discovery services for ULS systems will therefore need to support QoS capabilities for finding services. This QoS is distinct from the QoS that a service might provide once it is discovered.

For example, consider the QoS requirements needed for discovery services in disaster recovery, such as in the aftermath of a hurricane hitting a large metropolitan area. Critical services would need to be found in a timely manner, *e.g.*, information from monitors on levee walls during floods from excessive rains, since late responses could be disastrous. This scenario is just one of many examples of how discovery QoS can be crucial to acceptable performance. Other examples involve different kinds of QoS, such as specifying when and at what rate a discovery service might on its own search for new, updated, or removed services.

The DDS discovery services in our survey provide limited QoS for discovery by specifying metatraffic transport priorities and restricting nodes that are able to connect. While this provides some initial support of rudimentary QoS for discovery services it does not enable solutions for the disaster recover scenario outlined above. Much more work is needed, therefore, to identify and address discovery QoS requirements for ULS systems.

• **Service negotiation.** Most discovery services only retrieve services as requested by clients, *i.e.*, the clients either get what they asked for in whole or they get nothing, but there is no negotiation of services. An example of negotiation could be a client looking for a particular service with particular attributes that is not available. The discovery service might negotiate with the client to see if there are other available services that are close to what the client wants. The provided service might not be ideal, but it might be better than no service at all.

Negotiation of services is essential in ULS systems. Again, consider the hurricane disaster recovery described above. Rescuers are tasked with finding people stranded by high water and bringing them to safety. As the rescuers find survivors they realize some need immediate medical attention, so they seek to discover the appropriate health care facility. Unfortunately, due to the wide-scale impact of the disaster the appropriate health care facility is inundated with patients and has no capacity. In this instance, the next best plan would be to find a health care facility (*e.g.*, a paramedic facility at a fire station) that might not be ideal but is better than no facility at all.

In cases like this, finding a less optimal service in time is better than finding an ideal service too late. Research on service negotiation may therefore be able to leverage work on QoS of the discovery service itself, as described above. Although there is a large body of literature on negotiation strategies for multi-agent systems [15], this work must be scaled up to the scope of ULS systems and integrated into discovery service implementations.

• **Network scope and type.** An important property of discovery services for ULS systems will be support for networks that are broader in scope and more dynamic than today's conventional fixed LANs. By their very nature, ULS systems will run in WANs, and increasingly will involve mobile ad hoc networks (MANETs). As a result, significant R&D will be required on discovery services that can operate robustly in environments characterized by distributed control, configuration, and administration, *i.e.*, that exhibit (1) lack of centralized control, (2) self-organization and self-restoration, (3) transmission through multiple hops, and (4) frequent link failures and changes of network topologies.

While most discovery services can discover services dynamically, ULS systems require even greater dynamism in the WANs and MANETs outlined above. Services must be discovered in a timely and reliable manner and the window of time that a service is available may be very short and/or sporadic. For example, an aircraft or satellite might only be in contact with a discovery service on the ground briefly due to inclement weather conditions or terrain obstructions.

For some transport protocols (*e.g.*, TCP) this actually creates more problems than it solves. While Internet Protocol version 6 (IPv6) supports a plethora of self-discovery and re-configuration support new R&D is needed on discovery services for ULS systems that can operate robustly even in the face of discontinuities in time or availability in the network. Just as the Internet Protocol was a breakthrough in connecting disjoint LANs, innovations will be needed to connect together the discontinuous pieces of a sporadic network with limited availability to provide the needed connectivity.

5. CONCLUDING REMARKS

Technologies that support ULS systems are gaining attention in the research community, as evidenced from the ULS systems report [6] and workshops at ACM OOPSLA 2006 and IEEE/ACM ICSE 2007. This paper describes several categories within a taxonomy of existing discovery services that require further research and experimentation. Four categories particularly relevant to ULS technologies include (1) heterogeneity, (2) discovery QoS, (3) service negotiation, and (4) network scope and type, as shown in Table 2.

The most mature of these four technologies is the support for wide-area networks (WANs) since several discovery services in our survey (*i.e.*, the CORBA Naming and Trading Services, DDS implementations, UDDI, JXTA, Gnutella, and Napster) are designed for and support WANs. ULS systems, however, will push the envelope for size, scope, and granularity of availability. As ULS systems become more prevalent, however, they will include more entities over a wider geographical area and a broader range of networks (such as MANETs), so the services they provide will be available more sporadically and in smaller time slots. The scale needed for ULS systems will therefore force rethinking of the current WAN support for these services.

Existing discovery service implementations and R&D activities provide little or no support for the other three categories. There has been some work on discovery QoS, and the discovery services for DDS in our survey address this to a limited extent. The level of support must be enhanced greatly, however, to support the scale of ULS systems. We could find no existing discovery services that supported service negotiation. Current work should therefore be enhanced and new research be initiated to address ULS system needs for federating existing discovery services to collaborate and interact, as well as adding global properties to any such federation, *e.g.*, QoS-enabled federation of discovery services.

Our research group at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University is using the taxonomy and the resulting gap analysis presented in this paper to investigate policies and mechanisms for developing discovery services that can meet the needs of ULS systems. In the near-term, support for federations of discovery services to enable heterogeneity (*a.k.a.* open scalability) will be a research focus for us. Moreover, we have started researching the addition of QoS (*i.e.*, real-time, fault tolerance, and security) support for discovery in anonymous publish/subscribe middleware. We plan to transfer this knowledge and experience into the broader context of discovery services for ULS systems.

Acknowledgments

The authors would like to thank Prof. Aniruddha Gokhale for his assistance in developing this paper and Prof. Larry Dowdy for his encouragement in writing this paper.

6. REFERENCES

[1] Quadrennial Defense Review Report. www.defenselink.mil/pubs/pdfs/qdr20060203.pdf, February 2006. p. 70.

[2] Bluetooth Special Interest Group. The Official Bluetooth Wireless Info Site. www.bluetooth.com/bluetooth/, 2007.

[3] World Wide Web Consortium. Web Services. www.w3.org/2002/ws, 2002.

[4] Adrian Friday, Nigel Davies, Nat Wallbank, Elaine Catterall, and Stephen Pink. Supporting Service Discovery, Querying, and Interaction in Ubiquitous Computing Environments. *Wireless Networks*, 10(6):631–641, November 2004.

[5] Gnutella.com. Gnutella. www.gnutella.com, 2001.

[6] Software Engineering Institute. Ultra-Large-Scale Systems: Software Challenge of the Future. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, Jun 2006.

[7] Teemu Koponen and Teemupekka Virtanen. A Service Discovery: A Service Broker Approach. In *Proceedings of the 37th Hawaii International Conference on System Sciences*, pages 284–290, Washington, DC, USA, January 2004. IEEE Computer Society.

[8] Napster. Napster. www.napster.com, 2007.

[9] OASIS UDDI. Universal Description, Discovery, and Integration Protocol. www.uddi.org, 2007.

[10] Object Management Group. *Trading Object Service Specification Version 1.0*, OMG Document formal/2000-06-27 edition, June 2000.

[11] Object Management Group. *Common Object Request Broker Architecture Version 1.3*, OMG Document formal/2004-03-12 edition, March 2004.

[12] Object Management Group. *Naming Service Specification Version 1.3*, OMG Document formal/2004-10-03 edition, October 2004.

[13] Object Management Group. *Data Distribution Service for Real-time Systems Specification*, 1.2 edition, January 2007.

[14] Project JXTA. JXTA. www.jxta.org, 2006.

[15] Tuomas Sandholm. Distributed Rational Decision Making. In G. Weiss, editor, *Multiagent Systems: A Modern Introduction to Distributed Artificial Intelligence*. MIT Press, 1999.

[16] Sun Microsystems. Jini Connection Technology Version 1.2. www.sun.com/software/jini, December 2001.

[17] Sun Microsystems. Jini Lookup Service Specification Version 1.1. www.sun.com/software/jini/specs/jini1.2html/discovery-spec.html, September 2006.

[18] The OpenSLP Project. Service Location Protocol Version 1.2.1. www.openslp.org, March 2005.

[19] UPnP Forum. UPnP Device Architecture Version 1.0. www.upnp.org, June 2000.

[20] Koen Vanthournout, Geert Deconinck, and Ronnie Belmans. A Taxonomy for Resource Discovery. *Personal and Ubiquitous Computing*, 9(2):81–89, March 2005.