-01THIRD QUARTER 2005, VOLUME 7, NO. 3

IEEE COMMUNICATIONS SURVEYS
The Electronic Magazine of Original Peer-Reviewed Survey Articles

www.comsoc.org/pubs/surveys

SERVICE NAMING IN LARGE-SCALE AND MULTI-DOMAIN NETWORKS

REAZ AHMED, RAOUF BOUTABA , FERNANDO CUERVO, YOUSSEF IRAQI, TIANSHU LI, NOURA LIMAM, JIN XIAO, JOANNA ZIEMBICKI

ABSTRACT

The increasing availability of high-performance network resources creates a rich breeding ground for widely-distributed applications that span multiple network domains or administrative domains. Such applications provide services that can be accessed by remote users. Discovery and management of these systems require the ability to name the provided services. In light of these requirements, we distill a set of criteria for comparison of naming schemes: readability, extensibility, namespace size, naming authority, name resolution architecture, name persistence, and standardization. Based on these criteria, we summarize and compare a representative set of existing naming and name resolution approaches. We analyze the approaches based on our criteria, and select a number of candidate technologies for the design of a naming and name resolution mechanism suitable to a multi-domain, Internet-scale environment.

With the increasing need for networked applications and distributed resource sharing, there is a strong incentive for an open, large-scale service infrastructure that operates over multi-domain networks. Ongoing research in Web Service [1] and Grid technologies [2] is pushing toward global access of services. However, most of these works define only standards, protocols, and unified service access interfaces. We have yet to see any concrete platform specification that can support the envisioned service infrastructure. In this work we will concentrate specifically on the naming aspect of the infrastructure.

To support service invocation and access to resources, a service or resource must be uniquely identifiable and addressable. We denote as *naming* the process of assigning names to services and resources. Accessing a service or resource requires name resolution, which maps a name to an address. Naming and name resolution provide essential support of service discovery and service invocation since they allow for uniquely identifying entities. Moreover, a naming scheme can allow for abstracting the identity of a service or resource from its location and access mechanism. This survey summarizes and compares the most significant and representative existing approaches in naming services and resources in large-scale applications, and in resolving the names into addresses that can be used to access those entities. To evaluate the existing approaches in a clear and systematic way, we select a set of criteria that are crucial for a naming approach to be widely accepted, and appropriate for multi-domain applications. Each naming approach is then assessed in terms of the selected criteria: readability, extensibility, namespace size, naming authority, name resolution architecture, name persistence, and level of standardization (including available implementation).

We have chosen to focus our discussion on requirements specific to application-layer services that may be discovered and invoked across administrative domain boundaries in large (Internet-scale) networks. These requirements are different from those of applications in local-area networks, since they must consider heterogeneity, flexibility, and a greater degree of scalability. The main thrust behind our selection is therefore the suitability of a naming scheme to an Internet-scale environment that spans several administrative domains and provides many different types of services.

The rest of this article is organized as follows. We define the terminology used in this article. We expand on these definitions to formalize the concept of naming. We present the issues and challenges in naming and name resolution in the context of a multi-domain service infrastructure, while we give a brief overview of the existing approaches we have selected for study. We then use the presented issues to analyze and compare existing approaches. We put all the criteria together and recommend the characteristics of a combination of candidate approaches that, in our opinion, would best meet the

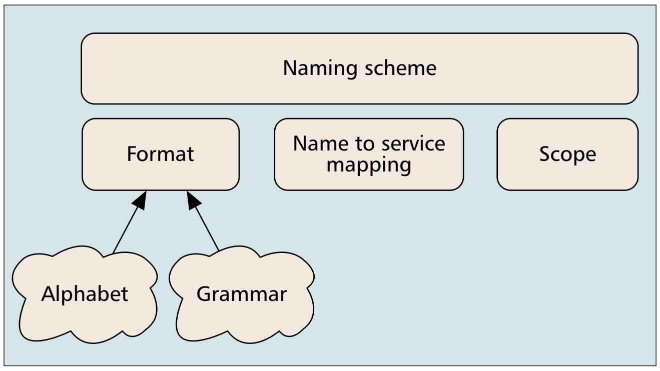


Figure 1. *Anatomy of a naming scheme.*

requirements of a large-scale multi-domain setting. We then conclude the article. In our main discussion we have selected a set of representative naming schemes. Appendix 1 supplements this discussion by introducing a number of specialized naming approaches that were omitted in our analysis, and we provide reasons for their exclusion.

TERMINOLOGY

Service: *A set of functionalities associated with a process or system that performs a task. We say that the process or system implements the service.*

Resource: *An entity that is used or acted upon by a process or system. A service functionality takes resources as input.*

To further clarify the difference between the two above definitions, a functional representation can be used to show the relationship between a service and a resource. We can denote a service by a function f, and a resource by an input variable x. The outcome of the service is then returned as $f(x)$. Notice that a service can be invoked by another service. In this case, the outcome of the service f becomes a resource and is used by another service, say g. This is called service composition and the outcome of the service g becomes $g(f(x))$. Hence, we do not classify the service itself as a resource; rather, the outcome produced by the service can be, again, treated as a resource.

Since the act of providing a resource is in itself a service, for the sake of simplicity we will occasionally say *service naming* where we mean *naming of services and resources*.

We now define several key concepts related to naming and name resolution. These concepts will be elaborated in detail in following sections.

Name: *A linguistic object that singles out a particular entity from among a collection of entities [3].*

Namespace: *The collection of all valid names.*

In RFC 2611 [4], namespace is defined as the collection of unique identifiers that have already been assigned. However, in the literature it is also defined as a collection of all valid names [5]. The latter definition is more consistent and hence adopted in this work.

Address: *An intermediary identifier between a name and a route [3] that allows a resource or a service to be reached.*

Address space: *The collection of all valid addresses.*

Naming authority: *An entity that has the authority to assign names to resources or services.*

Name resolution: *The mapping between namespace and address space.*

The following definitions concern a few concepts related to services.

Service description: *The set of descriptive attributes of a particular service.*

A service description exposes the ability to perform certain functions, may stipulate the manner in which available resources will be used to perform the functions, and may specify its method of access.

Context: *The circumstance in which an application runs. Context may include physical state, computational state, and user state [6].*

Cross-domain service: *A service that can be discovered and invoked by a user outside the provider's administrative domain.*

In this article we envision a multi-domain service infrastructure that supports cross-domain services.

ANATOMY OF A NAMING SCHEME

Before we focus on the requirements of a naming architecture in the context of a multi-domain service infrastructure, we will first formalize the concepts related to naming. By identifying the types and components of a name, we establish a basis for analyzing naming schemes and their suitability for naming various types of services. Clearly defining the components of name resolution gives us tools for understanding their impact on flexibility and performance of the application in which the naming architecture is used.

Note that the anatomy/structure of a name is not an evaluation criterion that we use for comparison of naming schemes, but rather a means to gain a better understanding of naming, and to build a basis for developing comparison criteria. Figure 1 summarizes the components of a naming scheme. The three main issues in naming are the prescribed format or structure of a name, the characteristics of assigning names to entities, and the scope of a naming scheme.

FORMAT

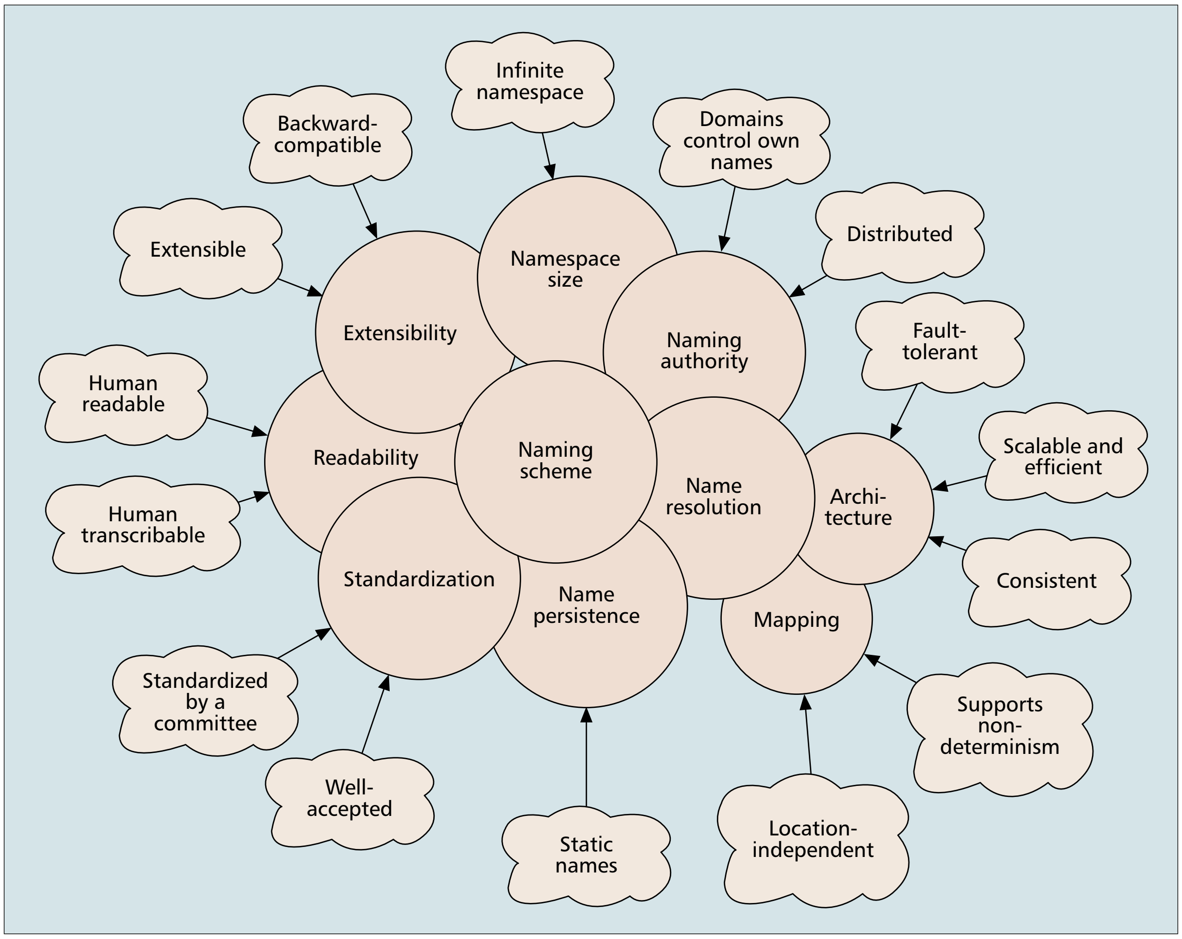
We can formally describe a namespace as a language. Given a finite alphabet Σ of symbols, a namespace $\mathcal{N} \subseteq \Sigma^*$ is a set of finite-length strings (or names) produced by a formal grammar $\mathcal{G}$. The format of the names in a namespace depends on the following characteristics of Σ and $\mathcal{G}$.

- Alphabet: Names can be composed of ASCII or Unicode characters, numbers, or a combination of the above. They could also be defined as non-human-readable strings of ones and zeroes.
- Grammar: The grammar of a naming scheme concerns the organization of name components. Names can be characterized as primitive (or flat), partitioned, or descriptive [3].

A primitive or flat name has no internal structure (e.g. JXTA 128-bit random UUIDs [7]).

A partitioned name is a succession of primitive names identifying, respectively, a domain, subdomain, sub-subdomain (where domain can be considered in a network, administrative, or more abstract sense), followed by a primitive name identifying the entity inside that sub-sub-…-domain. Domains are arranged in a strictly nested structure and must not overlap (e.g., URL [8]).

A *descriptive* name is a list of attribute-value pairs that are true for exactly one entity (in this case, a service). A partitioned name is a particular form of descriptive name that has a rigid structure (a rigid set of attributes). For example, the Solar approach [6] uses descriptive names. (However, as we will see, the Solar naming scheme leaves enforcing uniqueness up to the application.) Hybrid types of naming schemes are also possible, for example, a XORP Resource Locator [9] is a hybrid partitioned-descriptive name type.



■ **Figure 2.** *Summary of criteria and desirable traits.*

Mapping Names to Services

An issue related to name uniqueness is the mapping of names to the services (entities) that the names represent. Because a name is defined as inherently unique to an entity, two entities may not have the same name in the same scope. However, an entity may or may not be allowed to have multiple names. For example, a network card can have only one MAC address at a time, but a Web server can have more than one Domain Name.

Scope

The scope of a naming scheme refers to the set of services that will be named using the given namespace. This set may or may not involve entities in different domains. In the design of a naming scheme, it may be useful to consider how the naming scope will cope with intra-domain or inter-domain naming.

Description of Criteria

To provide a comprehensive and in-depth analysis of the various approaches to service naming, we hereby define a set of evaluation criteria. These criteria are generated from issues considered in a variety of surveyed works, and from the discussion presented earlier. This section provides a short description of each criterion and the rationale for its inclusion. To ground these criteria in our domain of interest, we also discuss how these criteria affect the suitability of naming systems to a multi-domain service infrastructure. Figure 2 summarizes the evaluation criteria and suggests their desirable values.

Readability

Names may or may not be considered *human-readable*, depending on whether they are memorable or comprehensible, and whether they provide useful information to the humans who use them. Readability is determined by the set of allowable characters (alphabet) in a naming scheme, as well as the format and structure of a name.

A naming scheme may define names that are strictly human readable, strictly non-human-readable, or it may allow both kinds of names. In some cases, non-human-readable portions of a name may have a human-transcribable representation. The readability/human-friendliness of an individual name is a subjective measure, influenced by the authority assigning the name. However, some naming schemes may lend themselves better than others to direct human use (e.g. a Domain Name is more human-friendly than an IP address). In a service infrastructure, human-readable names are important when service management is performed by humans, or when services are handled by human users during discovery or invocation.

Extensibility

The naming system may be required to be extensible for future updates. Updates can occur in the format and structure (e.g., more bits are allocated to the name), in the scope (i.e., introduction of new services to be named), and in the grammar generating the namespace.

In a quickly-growing and quickly-changing environment such as the Internet, it is desirable for naming schemes to be extensible, while remaining compatible with existing, older names from the same naming schemes. The naming scheme should accommodate changes in namespace and scope.

Namespace Size

The size of the namespace determines how many unique entities can be named. A limit on the size of a name implies that the namespace is finite. A finite namespace size allows devices to allocate a fixed amount of space for storing and transmitting names, which can improve computational performance and reduce implementation complexity. However, a finite namespace runs the risk of being exhausted, as evidenced by the impending depletion of the IPv4 address space [10, 11].

In a multi-domain environment, a naming scheme should have a namespace that is large enough to uniquely denote all existing and future entities in its desired scope. Since we can expect a global service infrastructure to grow unexpectedly, as the Internet did, an infinite namespace is ideal.

Naming Authority

A naming authority is the entity that assigns and manages names in a namespace. In some cases a central authority is responsible for the entire namespace (e.g., the Canadian government is a central naming authority that assigns social insurance numbers to every eligible resident of Canada). However, in a multi-domain naming scheme, it may be preferable to implement distributed authority, whereby each domain is responsible for naming the services under its own management. In such cases, namespace conflicts are avoided by assigning a defined section of the namespace to each domain.

In some cases a service may become its own naming authority. For example, in AutoIP [12] a node joining a network chooses its own IP at random (from a specified range), and contacts nodes on that network to make sure that no other node has chosen the same address. Regardless of its management scope, a naming authority is responsible for assigning and changing names, as well as preventing namespace conflicts.

For reasons of scalability, we prefer a distributed naming authority. Furthermore, in a multi-domain environment, domains might not want to rely on a centralized naming authority but rather have full control over the services within their own domains.

Name Resolution Architecture

While the choice of naming authority determines who assigns names to entities, the name resolution process determines how to translate names to the addresses that allow a user to access the corresponding entities.

Name resolution encompasses two issues: the characteristics of mapping the names to addresses, and the implementation of the name resolution architecture. In the following sections we formalize these two issues.

The Characteristics of Mapping

While the related issue in naming discussed the mapping of names to entities, the mapping discussed here concerns assigning names to the addresses of entities. This is a slightly different issue, as entities may have more than one address.

Given a namespace N and an address space A, we define a name resolution mapping (or a resolver) Res : $N \rightarrow \{A, \emptyset\}$ such that given a name $n \in N$, a single execution of *Res* returns at most one $a \in A$. An execution of $Res(n)$ returns $\emptyset$ when $Res(n)$ is not defined, that is, if n is not mapped to any address in A.

Name resolution can be *deterministic* or *nondeterministic*. A deterministic resolver maps each name in N to at most one address in A. Still, it is possible for *Res* to be defined in such a way that it maps more than one name to a single address (e.g., a name may have several aliases). Hence, a deterministic mapping can be one-to-one or many-to-one.

A nondeterministic resolver may map a name in N to two or more addresses. That is, two separate executions of *Res* may return different values (however, only one at a time). In this case, the name resolution mapping can be one-to-one, one-to-many, many-to-one, or many-to-many.

While most name resolution systems are deterministic, there are many cases where nondeterministic resolvers can be useful. For example, busy Web sites may assign a single DNS name to a cluster of machines with different IP addresses, or to machines in diverse geographic locations. When the resolver is invoked with that name, it may return any of the IP addresses that correspond to the machines, according to its own selection algorithm. If the machines form a load-balancing cluster, *Res* may select addresses at random, in a round-robin fashion, or select the address of the machine with the lightest load. If the goal is to improve performance by increasing locality, the resolver will select the address of the machine that is physically or logically closest to the user. It should be noted that if mapping data is replicated, and the mapping changes, the resolver may become unintentionally nondeterministic because some replicas may contain a stale version of the mapping.

Another aspect of the name-to-address mapping is how closely the name is coupled to the address at which the referred service resides. This aspect determines whether a name is *location-dependent* or *location-independent*.

Implementation

System Architecture — In addition to implementing a name-to-address mapping, an important issue in the design of a name resolution system is its architecture. A name resolution system may be centralized (a single name resolution service), hierarchical (as in DNS [13]), entirely distributed (peer-to-peer), or a combination (e.g. layered peer-to-peer, peer-to-peer with supernodes [14]).

There is an inter-dependence between the structure of the naming scheme and both the name resolution process and the architecture of the name resolution system. For example, in DNS (Domain Name System) the hierarchical naming scheme is tightly coupled to the name resolution process: each hierarchical layer of DNS servers is responsible for a specific section of the domain name. Other distributed name resolution architectures (such as those based on CAN [15]) decouple the naming scheme from the name resolution process, allowing a location-independent naming scheme (for both physical and logical meanings of "location"). This location-independence is achieved by making the name of the entity independent of the logical position of the name in the hierarchical naming scheme.

Each of the design choices described above presents its own set of challenges and tradeoffs in relation to scalability, efficiency, robustness, and consistency. These four criteria are described below:

Scalability — If a name resolution system is to handle a large number of name-address mappings or a large number of service users, it must be able to handle the load. Also, since the size of a cross-domain system is likely to grow with the number and size of domains, the architecture should be able to gracefully accommodate an increasing load.

The load on a name resolution architecture includes:

- The amount of storage required to keep track of the name-to-address mapping.
- Computation load on name resolution servers.
- The communication overhead of queries and responses.
- In the case of a distributed architecture, the communication overhead of sending update messages.

A fully centralized architecture does not need to handle update messages (unless it uses replication), but creates a potential performance bottleneck. A hierarchical system distributes the load among layers of name servers; however, the top-layer servers can still experience overload. To alleviate these conditions, implementations of centralized and hierarchical name resolution employ replication and caching [16].

Fully distributed systems scale well with respect to computation, storage, and network locality, but may require a high communication overhead in update messages.

Efficiency — In addition to being scalable, a name resolution system must also be as efficient as possible in its use of network/computational resources and must provide an acceptable level of performance. The exact definition of "acceptable level of performance" may vary, but since name resolution is used frequently, it is essential that it should not impede the performance of other parts of the system.

To illustrate the importance of improving name resolution performance, consider that in 1997 a study of wide-area Internet traffic reported that DNS messages comprised 18 percent of overall flows (where a flow is defined as a uni-directional traffic stream with unique source and destination IP addresses, port numbers, and IP protocol fields), second only to World Wide Web traffic [17].

In a wide-area network, the response time of name resolution servers can be improved by moving the name resolution server closer to the source of requests. This can be achieved by caching previous requests and/or replication of name resolution services. It has been found that in a network as diverse as the Internet, the effectiveness of a client-side cache can be very limited, with a cache miss ratio close to 100 percent [18]. However, in a system where users routinely access the same resources, caching can be used to improve performance significantly.

Fault Tolerance and Robustness — Name resolution is a vital part of any network or system, and its failure usually means a severe limitation of the system's functionality. Therefore, it is important that name resolution remain a robust, high-availability service that retains a reasonable level of performance even after the failure of other components of the system.

An important goal toward a robust name resolution architecture is that it should not contain a single point of failure. A centralized architecture is, in itself, a single point of failure, but can be made more robust by employing replication or keeping a hot-backup failover system in case of disaster. A properly designed distributed architecture has no single point of failure, but it still has to consider issues of fault tolerance and fault recovery in the event of failure of individual components or of path failure between components.

Consistency — While data replication and data caching can improve the performance, efficiency, and robustness of a name resolution system, it introduces the problem of maintaining consistent data across all replicas. This is especially important in the case of a system providing name resolution for a variety of highly dynamic services. Stale name resolution data may make some services temporarily unreachable, or may result in unnecessary network load as users repeatedly attempt to access services that have moved or have become unavailable. Information must therefore be exchanged between replicas in the form of push or pull update messages. Depending on how often name resolution information is updated, the system seeks to achieve a balance where data is kept consistent in a timely manner, but where the update messages do not cause undue communication load.

We examine the name resolution architecture with respect to the characteristics of name resolution mapping and system architecture, observing especially their effect on the four comparison sub-criteria: scalability, efficiency, robustness, and consistency.

Name Persistence

Names can be divided into two categories: static or dynamic [3]. A *static* name is a name that permanently denotes the same entity. This is a very usual property of names and is often implied. A *dynamic* name is a name that is assigned to an entity for only a limited period of time, which is short with respect to the lifetime of the entity.

A system that uses dynamic names may prevent the meaning of a name (i.e., the entity to which the name refers) from changing without the knowledge of consumers. This allows the users to use a name without testing if its meaning has changed since the last time it was used. However, this name persistence is very hard to support in a large scale distributed system, since the system must keep track of names and find out which users hold a given name, which is similar to, for example, the problem of distributed garbage collection. An alternative solution could involve broadcasting periodic updates of name validity; however, this is also unlikely to be supportable in a large-scale system [19].

If a name can change its meaning without notifying the users, then some identification mechanism must be provided to allow users to confirm that the name refers to the entity that they expect. In a large scale system, it is difficult or impossible to inform all current and potential users of changes in the meaning of names. Therefore, careful consideration of name persistence has to be taken when designing a large-scale naming scheme.

Standardization and Implementation

The final criterion in comparing naming schemes is their present level of acceptance in the Internet community. A naming scheme that has a well-defined standard and is easy to incorporate into the existing infrastructure is far easier to deploy than an experimental, constantly-changing scheme, even if the latter is superior in other aspects. Ideally, the naming scheme used in a multi-domain service management system would be accepted by an official standards body, and would have most components of its name resolution already in place.

Overview of Naming Approaches

In this section we give a brief overview of several existing approaches to service naming. Examined approaches include UUID; the different flavors of URIs, including URLs and URNs; Web Service and Grid Service naming; XRLs; and the INS and Solar naming approaches. In our comparison we have attempted to cover those approaches most likely to be suitable for large-scale applications, those that were already well-accepted in the Internet community, and those that gave a representative sample of various types of naming approaches.

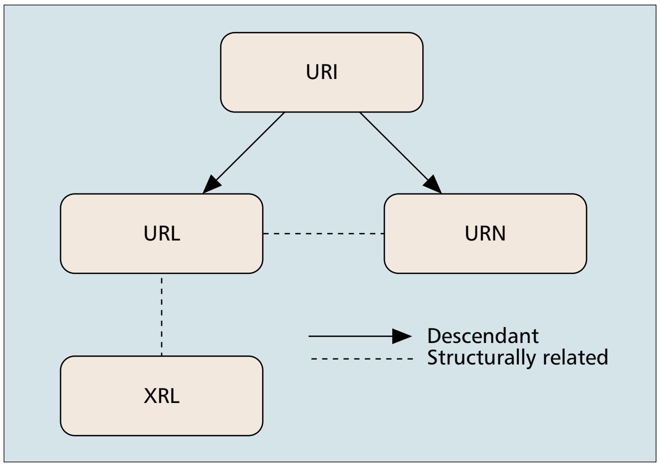


Figure 3. *URI hierarchy.*

UUID

Universal Unique Identifiers (UUIDs), also known as Globally Unique Identifiers (GUIDs) [20], are flat 128-bit identifiers where the uniqueness of these identifiers is claimed to be guaranteed across space and time without requiring a central registration process.

The UUID relies upon a combination of components to ensure the uniqueness of a UUID. A UUID contains a reference to the network address of the host that generated the UUID, the timestamp (the precise time when the UUID is generated), and a randomly generated component as a protection against situations when the first two components fail to guarantee the uniqueness of a UUID.

Depending on the generation algorithm used, UUIDs are either guaranteed to be unique until 3400 A.D. or extremely likely to be different. Below is an example of a string representation of a UUID:

```
f81d4fae-7dec-11d0-a76-00a0c91e6bf6
```

URI

The two main members of the Uniform Resource Identifier (URI) family are the Uniform Resource Locators (URL) and Uniform Resource Names (URN).

URL — Uniform Resource Locators provide formalized information for location and access of resources via the Internet [8]. They are used to 'locate' resources by providing an abstract identification of the resource location. A URL may also specify operations to be performed on an already located resource such as access, update, replace, or find attributes. In general, only the access method needs to be specified for a URL scheme.

A URL typically contains the Internet application protocol (http, ftp, etc.) required to access the resource to which it refers, the domain name of the host that provides the resource, and a path name. Below is an example of a URL [21].

```
http://directory.google.com/Top/Reference/
```

URN — Uniform Resource Names (URNs) [22] are Internet resource identifiers with the specific requirements for enabling location independent identification, as well as longevity of reference. URNs are part of the larger Uniform Resource Identifier (URI) family [23] with the specific goal of providing persistent naming of resources.

URNs can be distinguished from other URIs by the initial **urn:**, followed by a Namespace Identifier (NID), a colon, and a namespace-specific string. The official list of registered NIDs is maintained by IANA. In April 2005 there were twenty formal registered NIDs [24]:

```
ietf, pin, issn, oid, newsml, oasis, xmlorg,
publicid, isbn, nbn, web3d, mpeg, mace, fipa,
swift, liberty, swift, uuid, uci and clei
```

IANA also keeps a list of informal NIDs of the form "*urn* — ⟨*number*⟩" where ⟨*number*⟩ is assigned by IANA. Currently, urn-1 through `urn-5` are the registered informal NIDs [25]. Below are two examples of URNs [26, 27].

```
urn:isbn:0-395-36341-1
urn:xmlorg:objects:dtd:xml:docbook:v4.1.2
```

There has been a certain amount of discussion about the relationship among the concepts of URIs, URLs, and URNs. More precisely, there are currently two incompatible views on URI partitioning [25]: the "classical" view and the "contemporary" view.

Classical view. URI is partitioned into two classes: URL and URN, where a URL specifies the location of a resource and a URN specifies its name. Hence, http: would be a URL scheme, while isbn: would be a URN scheme.

Contemporary view. URL does not refer to a formal partition of URI. Instead, it is an informal concept that describes a subclass of URI schemes. URNs are defined by the URI scheme "urn:."

Figure 3 summarizes the URI hierarchy. Here, URLs and URNs are seen as distinct but related subsets of URIs, and XRLs as having a similar structure as URLs.

Web Services and Grid Services

Web Services are a set of standards and protocols that use Web technologies (such as XML and HTTP) to provide a means of operating between different software applications, running on a variety of platforms and/or frameworks. Grid Services build on Web Services to create a Grid system architecture, which includes support for stateful service instances, supporting reliable and secure invocation (when required), lifetime management, notification, policy management, credential management, and virtualization [2].

Instead of defining their own naming schemes, Web Services and Grid Services take advantage of existing naming schemes such as UUID and URI. We have decided to include them in this article because of their importance as service-providing mechanisms, and because they illustrate how existing naming schemes can be combined and adapted to the needs of a particular application.

The most popular approach for Web Service name resolution is the Universal Description, Discovery and Integration (UDDI) registry, in which service providers publish UDDI documents. Each UDDI document contains the name and description of a particular Web service. There are several basic components of the UDDI data model, each of which is associated with its own UUID identifier [28]:

- `BusinessEntity` (white pages)
- `BusinessService` (yellow pages)
- `BindingTemplate` (green pages), and
- `tModel` (allows the use of an external Web Service reference such as a Web Service Description Language (WSDL) document).

UDDI combines the UUID with other information such as *discoveryURL*, description protocol binding and access points to integrate Web Service naming and name resolution, facilitating the use of Web services.

Grid Services use a two-layer naming scheme, whose components — GSH and GSR — are described below.

GSH: The Grid Service Handle (GSH) is a globally-unique name that distinguishes a specific Grid service instance from

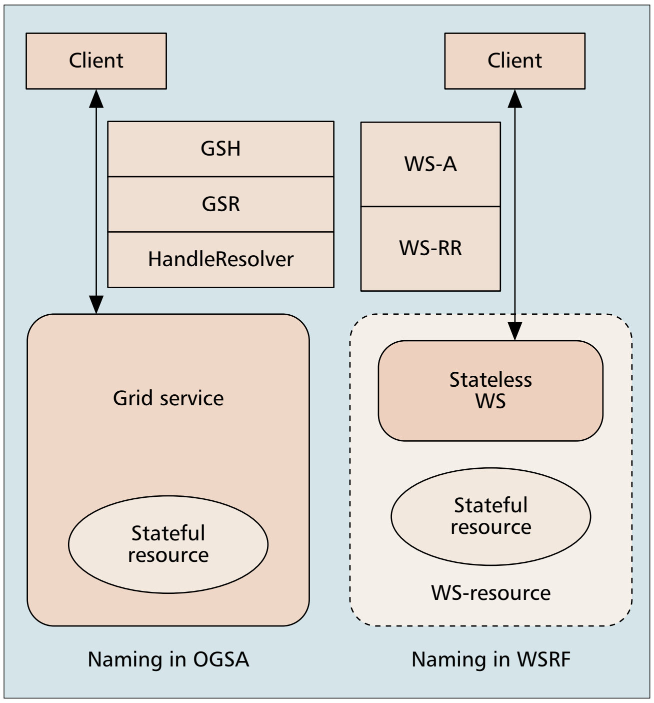


Figure 4. *OGSA vs. WSRF.*

all other Grid service instances that have existed, exist now, or will exist in the future. A GSH is often viewed as a permanent network pointer to a particular Grid service instance. A GSH is a minimal name in the form of a URI and may not carry enough information to allow a client to communicate directly with the service instance.

GSR: The Grid Service Reference (GSR) contains all information that a client requires to communicate with the service instance via one or more network protocol bindings. The format of the GSR is specific to the binding mechanism used by the client. For example, if SOAP binding is used, a GSR would contain a URL pointing to a WSDL document.

Before using a Grid service, a GSH must be resolved into a GSR. In contrast to GSH, GSR has an expiry-time and temporarily binds a grid service to a network protocol. OGSI specifies a standard interface, named "HandleResolver," for GSH-to-GSR mapping. A client performs this mapping before accessing the service for the first time and at the end of each expiry interval. The main purpose of using a two-tier naming scheme is to separate service discovery from service access, which offers great flexibility for large scale applications. However, the gain of this separation has a cost, as it does introduce some complexity to the name resolution process.

The more recent WS-addressing [29] proposal has presented a transport-independent mechanism that encodes the source and destination and other important address information directly within the Web Service messages. In addition, an endpoint reference is used in the WS-addressing proposal, making it very similar to the concept of a GSR. WS-Addressing extends the WSDL model (i.e. the endpoint references take the form of an XML-based document) to allow:

- Dynamic generation and customization of service endpoint descriptions.
- Creation of stateful Web services.
- Flexible and dynamic exchange of endpoint information.

The recent proposal for Web-Service Resource Framework (WSRF) [29] combines WS-Addressing (WS-A) and WS- RenewableReference (WS-RR) [30] for addressing and accessing statefull-resource(s) through stateless Web service. Analogous to the HandleResolver interface in OGSI, WS-RR extends WS-A endpoint reference to incorporate service handle renewal and resolution information. This WS-A/WS-RR combination is analogous to the GSH/GSR/HandleResolver construct (Fig. 4), and preserves the principle of separating service identifier from service access.

XRL

XORP (eXtensible Open Routing Platform) [31] is an experimental open-source router platform intended to test new router software and routing protocols. XRLs (XORP Resource Locators) [9] are used to mediate IPC (Interprocess Communication) within XORP processes. Structurally, XRLs are very similar to URLs. Instead of representing a hierarchical resource location, XRL uses a specific syntax to describe an inter-process procedure call and its parameters. An XRL contains the *protocol family* to be used for transport, the *arguments for the protocol family*, the *interface of the target* being called, the *target's version*, the method, and the *argument list*. An example XRL is given in Fig. 5.

To clarify this example:

finder: refers to a special process that coordinates the resolution of XRL.

fea: (forwarding engine abstraction) is a special module that is used to abstract the physical organization of the underlying router hardware on which the XORP system is running. XORP also supports

- *rib*: (Router Information Base)
- *bgp*: (Broader Gateway Protocol) and *ospf* (Open Shortest Path First)

fti: refers to an interface within the process fea, and 1.0 is the version number of the interface *fti*.

add_address: the method within the interface *fti* that is being called.

The rest of the XRL represents the parameter to the method *add_address*. The parameter section starts with "?" and parameters are separated by "&." Each parameter is expressed as *name:type=value*. For the first parameter of the above example *vit* is the parameter name, *txt* is the parameter type, and *fxp0* is the parameter value.

INS

Intentional Naming System (INS) [32] is a research project introduced by the MIT Laboratory for Computer Science in 1999. INS provides protocols for service naming and discovery in dynamically changing and mobile networks.

INS integrates resource/service name and description into hierarchical descriptive names, called Intentional Names or

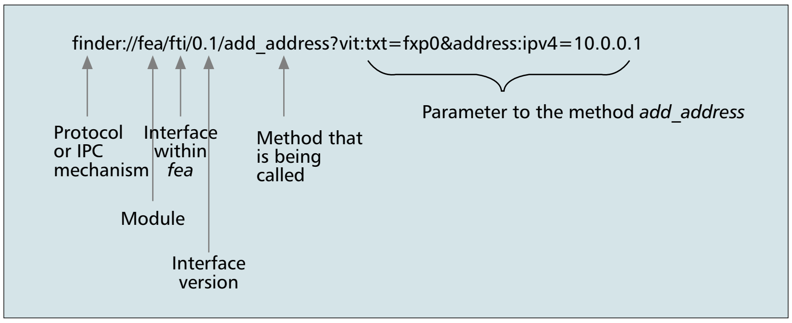

finder://fea/fti/0.1/add_address?vit:txt=fxp0&address:ipv4=10.0.0.1
Parameter to the method add_address
Protocol or IPC mechanism
Interface within fea
Method that is being called
Module
Interface version

Figure 5. *An example of an XRL.*

name-specifiers. More precisely a *name-specifier* is a tree-like structure of attributes and values, and expresses the relationship between attributes. If attribute *Y* depends on attribute *X* then Y is placed as a descendant of *X*, otherwise *X* and *Y* are placed in different sub-trees. This naming scheme is flexible and can be used to name a wide variety of resources and services.

For example, suppose we want to name a printer located in room 3335 of the Davis Centre in the University of Waterloo. Suppose also that the printer has public access and supports print resolution up to 1200 dpi × 800 dpi. Clearly the resolution, location, and access rights of the printer are not related; however, "room number" is meaningless without the building name. A name-specifier for the printer is shown in Fig. 6 (assuming an organizational scope).

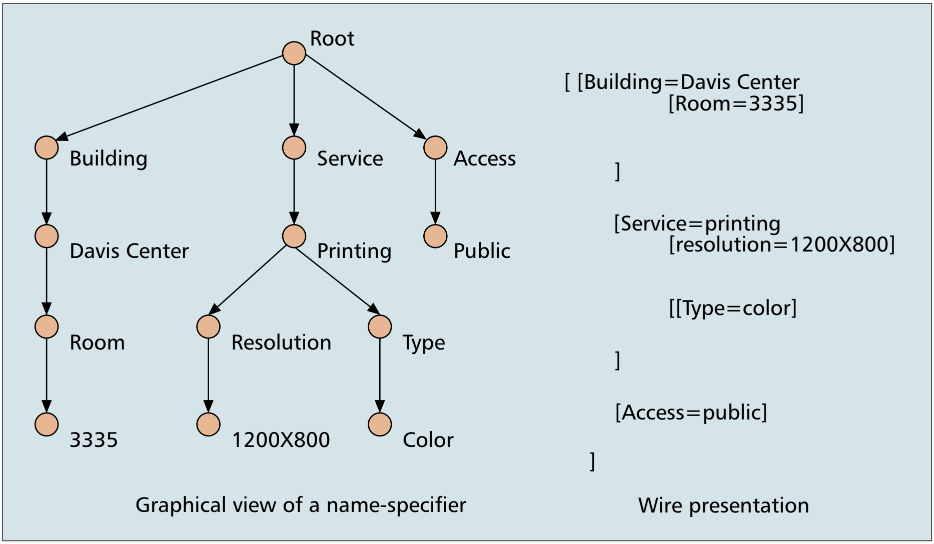


Figure 6. *Example of an INS name-specifier.*

Each name-specifier is accompanied by a name-record. Two important entries of a name-record are network address (i.e., IP address:port number pair) and *AnnouncerID* (unique identifier for a service). These are used to distinguish between multiple instances of a service running in different machines, or in the same machine, respectively. Hence, while the descriptive INS names are not intrinsically guaranteed to be unique, the name-record provides this uniqueness of service instances.

SOLAR

Chen *et al.* [6] present a naming scheme that is similar to INS, but additionally supports names that are dynamic (change over time). The proposed naming scheme is descriptive (i.e. contains a list of attribute-value pairs) and some values in a name can change to provide context information, typically the location of a mobile service. The variable parts of a name are evaluated each time the name is referenced.

Suppose we want to name laptops (with wireless cards) at the University of Waterloo according to the Solar naming scheme and want the names to incorporate location information. Assume that inexpensive location tags are placed in different rooms and passively transmit the building and room information in close vicinity. Naming software on the laptops interprets a nearby transmission and updates two variables named *$building* and *$room*. Then a laptop belonging to Bob can be named:

```
[Device=Laptop, Owner=Bob,
location=$building:$room].
```

When Bob moves to room 3335 of the Davis Centre (DC), his laptop will be named

```
[Device=Laptop, Owner=Bob, location=DC:3335].
```

As in INS, each entity in Solar is also identified by a unique name-record.

COMPARISON OF NAMING APPROACHES

Based on the criteria defined earlier we now compare and contrast naming approaches introduced in the previous section.

READABILITY

It is worth noting that except for UUID, which we consider as non-human readable, the readability of a naming scheme does not depend on the scheme itself. More precisely, what we are discussing is whether the naming scheme is able to support or handle human-readable names. In fact, most of the schemes we discuss here support using human-readable names, but it is up to the naming authority to decide whether to name the service in a human-readable manner.

UUID — A UUID is a non-human readable 128-bit sequence. It can be represented in other string formats, such as dash-separated hexadecimal strings.

A few solutions have been proposed to deal with the opacity of UUID as a service identifier. For example, Balakrishnan *et al.* [33] propose mapping human-readable canonical names to flat, not human-readable (UUID-like) identifiers. They note also that the flat identifiers would be returned to a human as a result of a human-directed search, which would already provide a reasonable amount of supplementary information. In addition, UUID identifiers returned as a search result could be accompanied by descriptive meta-information that would be independent of the naming scheme.

URI — While URIs are not required to be human-readable, they are required to be *human-transcribable* [34, 35], i.e., it must be possible for a human to copy out a URI on an article or with a computer keyboard.

a) URL. A URL normally consists of human-readable ASCII strings. However, non-ASCII octets (as well as reserved characters) may be included in a URL by encoding them in a character triplet consisting of a " percent," followed by two hexadecimal digits. [8]

b) URN. The URN namespace ID consists of ASCII characters. Most URN namespaces specify names in a convenient, human-readable form. However, as in URLs, non-ASCII characters can be encoded by character triplets consisting of a "percent" and followed by two hexadecimal digits.

Web Services and Grid Services — In a UDDI entry, non-human readable UUIDs are used to identify the businesses and services. In OGSA, a GSH takes the URI format.

XRL — In XORP, two forms of XRLs are used: the unresolved form and the resolved form. An example of these two forms is shown below.

```
finder://fea/fti/0.1/add route?
   net:ipv4net=10.0.0.1&gateway:ipv4=192.150.187.1
xudp://192.150.1.5:1992/fti/0.1/add route?
   net:ipv4net=10.0.0.1&gateway:ipv4=192.150.187.1
```

Unresolved XRLs are expected to be issued by a human user from a command-line interface (CLI) or be embedded in configuration and management scripts. For this reason unresolved XRLs are designed to be human-friendly. On the other hand, a resolved XRL can include a non-human-friendly network-specific address (like an IP-address:port-number pair), and are used within the application, transparent to the user.

INS — Names in INS are human readable and have fewer structural restrictions than other naming schemes such as URLs and XRLs. INS name specifiers can express hierarchically-related or orthogonal name components. Logically, the components are interpreted to have a tree-like arrangement, and can be written using nested parentheses or XML notation. Figure 6 shows an example of an INS name specifier in a tree representation and its corresponding nested-parentheses notation.

Solar — Solar names are human-readable and are constructed using a list of descriptive attribute-value pairs, where the variable portions of a name are marked as $variable-name. Meaningful variable names can be selected to improve readability.

EXTENSIBILITY

UUID — UUID uses 128-bit binary strings to identify services. To accommodate a larger number (i.e. larger than 2128) of services, more bits will be needed. However, the UUID scheme does not consider an extension to its namespace. The UUID generation algorithm suggested in the current UUID scheme would also need to be modified in order to generate UUIDs with longer length. Also, it is unclear how to ensure the compatibility between the new (longer) UUIDs and the existing UUIDs.

URI — The URI namespace can be extended by registering new schemes that can be used to denote new types of resources or services. Some of the existing URI schemes are `ftp`, `http`, `file`, and `urn`. The `urn` scheme can be further subdivided into namespaces such as `urn:ietf` and `urn:isbn`. In this example the first namespace is used to specify IETF documents, the second to uniquely specify book identification records [36]. Proposed URI extensions include:

- Use of URIs as identifiers for non-network resources (for example, to identify an abstract object such as an XML namespace, or a physical object such as a book or a person) [25].
- IRIs (Internationalized Resource Identifiers): the extension of URI syntax to non-ASCII characters, in order to allow the use of languages which do not use a Latin alphabet [37].

Web Services and Grid Services — In the case of Web Services, information about businesses and services is modeled in the UDDI data model. Due to the use of UUIDs as unique identifiers, the extensibility of this scheme largely depends on how UUIDs can be extended to accommodate more businesses and services (as discussed earlier). In the case of Grid Services, since a GSH takes the format of a URI, it is extensible for future updates.

XRL — XRLs are designed specifically for inter-process communication in the XORP platform and are flexible enough to handle new XORP APIs. However, for our purposes (i.e. naming a wide variety of services), XRLs are not very extensible in scope due to their rigid structure. The components of an XRL are fixed and have pre-determined semantics. Unlike URLs, XRLs have a fixed number of name components in the section preceding parameters. Hence, it may be difficult or impossible to extend the structure of the XRL naming scheme. For example, XRL does not give us the liberty to name entities that cannot be addressed in terms of a procedure call.

INS — INS allows names to have service-defined attributes and values. There is no restriction on the number of levels in a name-specifier, or on the number of children of a node in the name-specifier tree. These features ensure the extensibility of the INS scheme. INS aims to name a wide variety of services and is expected to adapt to the changes in scope. It is possible to adapt INS to a multi-domain environment by arranging individual name-trees that represent different administrative domains under a dummy root node.

Solar — Like INS, names in Solar are composed of arbitrary sets of attributes and values. A service has the liberty to specify attributes and values according to its own requirements. This mechanism makes possible the naming of a wide variety of services with great extensibility.

NAMESPACE SIZE

All of the naming schemes discussed here, except for UUID, have an infinite namespace size. The size of the UUID namespace is fixed at 2^{128}.

NAME RESOLUTION ARCHITECTURE

UUID — The UUID specification does not specify a name resolution architecture. Applications that use UUID for naming must provide their own name resolution mechanism. The UUID resolution process, i.e., the process of matching the UUID to the data structure or document with which it is associated, depends on the architecture and the data structures used by the application. Hence, the properties of name resolution remain an implementation issue.

Recently, Balakrishnan *et al.*, [33] have proposed an Internet-scale three-layered service naming scheme based on flat, non-human-readable names such as UUIDs. In this scheme, given a user-level descriptor (ULD) or search string, a search engine query would return a flat (e.g., UUID) service identifier (SID). The SID would, in turn, resolve to one or more triples containing a flat endpoint identifier (EID) accompanied by additional access information about the transport protocol and port. For example, if a SID refers to a Web server, it might be resolved to (`EID of the web server`, `TCP`, `port 80`). The EID would then be resolved into the current IP address of the endpoint. Both the SID-to-EID and the EID-to-IP resolution would be executed using a distributed hash table (DHT), giving $O(logN)$ resolution performance, where N is the number of nodes in the DHT.

URI

URL: A hierarchical architecture is used to resolve URLs. The Domain Name System [13] is used to resolve the domain name component of the URL into an IP address, while the rest of the URL is usually resolved by application-specific components. For example, when a URL is used to locate a Web page, the Web server hosting the page resolves the URL path into a physical file location.

The hierarchical DNS architecture used to resolve URLs has proven to be efficient and scalable, and suitable for Internet-size applications. The top-layer resolvers introduce potential bottlenecks and single points of failure. However, this problem can be resolved both by caching resolutions at lower-layer DNS servers, and by replicating the top-layer DNS servers. Although caching higher-layer DNS requests provides a performance increase and a measure of fault-tolerance, it also introduces a potential delay in propagating name mapping changes throughout the system, which could introduce inconsistency of resolution. Therefore, the URL resolution scheme (and, in particular, Internet-scale DNS) is less suitable for resolution of highly dynamic names.

A distributed hash table (DHT)-based Cooperative Domain Name System (CoDoNS) [38] has recently been proposed as an alternative to hierarchical DNS resolution. This system uses the Pastry DHT [39] along with the Beehive [40] replication framework to provide failure resistance and a measure of protection against denial-of-service attacks, at the price of increased storage and communication overhead. The proactive caching technique adjusts the amount of replication for each object based on its popularity, and dynamically adjusts as the popularity of the object changes over time, resulting in an average constant-time lookup. This dynamic adjustment of replication allows for handling unexpected loads on the resolution architecture such as the flash crowd effect. CoDoNS can be used as a stand-alone resolution architecture, or as an extension of the existing DNS. While a DHT-based name resolution architecture is more fault-tolerant than a hierarchical architecture [38], it introduces a problem of ownership of resolution nodes. The canonical DNS infrastructure depends on a "pay-for-your-own" model: domains provide their own service, while the central facilities required (the root servers) are minimal and relatively inexpensive [33].

A cooperative DNS architecture would require a new economic model, likely consisting of cooperating resolution service providers with mutual peering relationships.

URN: The URN specification separates the notions of name assignment and name resolution [41]. A provider of a resource or service can choose resolver services independently of other providers.

A resolver translates URNs into URLs, URCs (Uniform Resource Characteristics) or other URNs; it may also provide direct access to the referred service or resource. Resolution is not required (or guaranteed) to be deterministic; the resolution of a URN into an instance of a resource may reach different instances under different conditions [41].

The URI community uses the concept of a heuristic of following meta-information "hints" to achieve URN name resolution. To better formalize URN resolution concepts, Architectural Principles of URN Resolution [41] define a set of guidelines/requirements with respect to the evolvability of URN resolution, delegation of naming authority, efficiency of resolution, and privacy/security concerns. An additional specification [42] proposes a formal description of interaction in the URN resolution process.

To facilitate URN resolution, the Naming Authority Pointer (NAPTR) specification [43] defines a new DNS resource record that may be used to discover resolvers for URNs. The "services" field in the record specifies the "resolution protocol" as well as the "resolution services" it offers. Resolution protocols for URN include Z3950 [44], THTTP [45], RCDS [46], HDL [47], and RWHOIS [48]. The NAPTR specification also lists a variety of resolution services such as:

- N2L (given a URN, return a URL).
- N2R (given a URN, return the named resource).
- N2Ns (given a URN, return URNs that refer to the same resource, also known as equivalent URNs).

For example, the THTTP protocol uses HTTP GET commands to resolve a URN into a URL or a list of URLs that specify the location of the desired resource. It can also be used to retrieve the resources themselves. For example, a URN that describes a picture can be resolved into JPEG, GIF, and PNG versions of that picture.

Web Services and Grid Services — In Web Services, the name resolution process involves translating a given UUID into an access point of a service, contained in a UDDI document. An access point is usually a URL but could also be an external reference to a WSDL document that contains protocol-binding information in XML format. Both in Web Services and Grid Services, WSDL can be used to specify how location-independent names can be resolved into access points so that services can be invoked through these access points. However, storing name and access point mapping information for all the businesses and services in a centralized registry does not scale well.

Web Services uses a centralized registry model for name resolution. A "cloud" of UDDI registry services can be maintained to distribute replicas of the directory information among many UDDI nodes so that the UDDI registry is logically centralized but physically distributed. The use of external WSDL reference allows service providers to update the service descriptions (including attributes such as protocol binding and access points) without changing the registry record. This reduces not only the amount of information stored in the registry but also the amount of service update requests to the UDDI registry, helping scalability. However, a centralized approach always retains a potential scalability problem, balancing the replication overhead with the need for consistency. Efficiency of name resolution depends on the load and distance of the replica, and on how efficiently the external WSDL document can be retrieved.

Name resolution in Grid Services (specifically, in the Open Grid Service Architecture (OGSA)) is achieved through the use of `HandleMap` or `Handle Resolver PortType`, which is a specialized Grid service that resolves GSHs to GSRs. A GSH can be resolved into different GSRs at different times and by different users according to policies. However, a single resolution of a GSH will return only one GSR. A centralized registry is used for name resolution, which improves consistency of resolution, but implies the scalability and single-point-of-failure problems associated with any centralized approach.

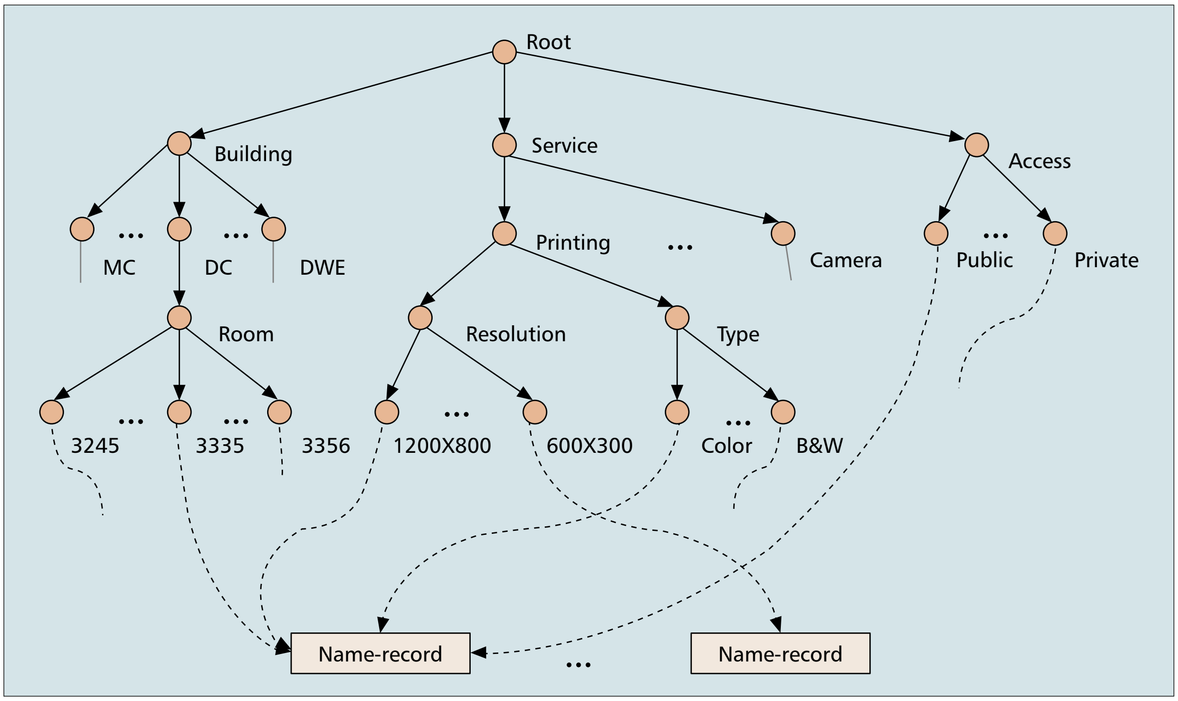
Although Grid Services are based on Web Services, the current implementation of the Globus Toolkit (the widely-used open-source set of tools used for building computational grids) does not yet support UDDI.

XRL — XRL name resolution is performed using an inter-process communication (IPC) call to a central process called a *finder*, which keeps track of all the registered *target names* and the communication protocols they support. In an unresolved XRL, the protocol family is set to "finder" and the protocol parameters are set to the *target name* for which the XRL call is intended. When a process wishes to dispatch an XRL (activate the IPC call) for the first time, it passes the unresolved XRL to the finder, who then replaces the target name with a network address and "finder" with appropriate the protocol, sending the result back to the client. The client process then uses the resolved XRL to access the desired resource. The client process also caches resolved XRLs to avoid consulting the finder process for future references.

All of the resolution messages are sent through IPC mechanisms (between processes). Hence, the XRL name resolution scheme is only as scalable as the underlying IPC. The centralized finder is efficient and ensures consistency, but introduces a single point of failure.

INS — In INS, a name-specifier can be resolved into one or more name records. The name record contains two components related to the resolution process:

- *Network address* of the service, i.e. IP address and port number.
- *AnnouncerID*, a unique identifier for a service, constructed from the host IP address and the service creation time. *AnnouncerID* is used to distinguish between services advertising the same name-specifier and residing in



■ **Figure 7.** *A sample INS name-tree.*

the same host.

All the name-specifiers in the system are stored in a data-structure called a `name-tree`, an attribute-wise merge of all the name-specifiers. Suppose two name-specifiers have the same attribute X in the same level with values V_1 and V_2. Then the name-tree will have a node X in that level with two children, V_1 and V_2. The leaf nodes of the name-tree have links to the associated name-records. An example name-tree is shown in Fig. 7. The name resolution process uses the name-tree to look up a name-specifier and returns the associated name-records.

In INS, the directory information (i.e. the name-tree) is replicated in each resolver node, also known as an INR (Intentional Name Resolver). Hence, name resolution is performed locally to an INR, based on cached information. Keeping the replicated name-trees consistent consumes a considerable amount of network bandwidth, especially in dynamically changing networks such as the Internet. Because of this overhead, this system does not scale well in wide-area networks. The system can tolerate failure of services but cannot handle INR failures efficiently.

To address the scalability issues of INS name resolution, in 2002 the MIT Laboratory developed INS/Twine [49] as an extension to INS. INS/Twine splits name-specifiers into strands (partial or complete paths from root to leaves in the tree), generates a numeric key for each strand using MD5, and uses Chord [50], a distributed hashing mechanism, to distribute these keys (and the accompanying name-records) in appropriate INRs. Some replication among INRs is additionally used to provide better tolerance of INR failures.

Solar — Solar adopts the INS name resolution mechanism. In Solar, mobile services (e.g., roaming laptops) connect to a proxy, which resolves a dynamic name into a static name. The static name is then transmitted to a nearby INR, which performs further name resolution (Fig. 8).

To populate an INR with name resolution information, an application registers with a proxy, which then advertises the appropriate INS name to the INR network. Whenever the dynamic name changes in another proxy, the changed name is disseminated using Intentional Multicast to all proxies where the name was previously registered.

As in INS, the INR network in Solar fully replicates all the name resolution information. The proxies, while retaining more independence, may also require a great deal of replication if services often move between proxies. The amount of required update messages hinders Solar's scalability, while the dynamic character of names can harm consistency of resolution if the network of proxies is too slow in propagating name changes.

Naming Authority

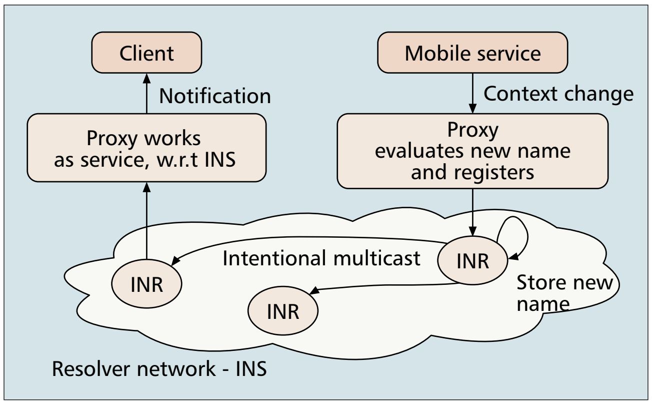
UUID — Ensuring the uniqueness of UUIDs does not require a centralized naming authority. The uniqueness of UUIDs is ensured through the use of a timestamp, a random component and a unique value over space for each UUID generator (usually a MAC address). Each UUID generator in this case becomes a naming authority, so that names can be assigned locally (i.e., on each device). The generation algorithm suggested in [51] supports very high allocation rates of up to 10 million requests per second per machine if necessary.

URI

URL: For use in Internet-wide applications, URL scheme identifiers such as http or ftp are registered with IANA [52]. Furthermore, the Domain Name portion of a URL must be registered with the Internet Corporation for Assigned Names and Numbers (ICANN) [53]. Finally, the remaining portion of a URL is assigned hierarchically within a domain.

URN: Each URN Namespace Identifier (NID) must be registered with the central authority IANA using the process outlined in *URN Namespace Definition Mechanisms* [4]. The entity that registers the NID then becomes the naming authority for names inside that namespace. URN name authority can be assigned hierarchically, where each domain is free to choose the structure of names within its own namespace, subject to the restrictions or constraints of its parent namespace.

Web Services and Grid Services — The use of UUID in Web Services implies that there is no centralized naming authority, since each UDDI node can assign UUIDs to the services that want to register in the UDDI registry. A UDDI registry is logically centralized but can be physically distribut-



■ **Figure 8.** *The use of INS in Solar.*

ed. There can be many types of UDDI registries and many UDDI operators that host UDDI registry nodes. (For example, Microsoft and IBM both host a UDDI Business Registry node for e-commerce applications.) These registry nodes interoperate with each other to form a "cloud" so that a business can register its service on any UDDI node.

In OGSA, since a GSH is a valid URI, it also follows the hierarchical approach used in URI name assignment. Also, since a factory approach is used to create a transient service instance for each client, we have to consider how to uniquely name every service instance. OGSA uses globally unique names (URLs) combined with locally-generated identifiers (usually a large hash value) to form a valid URI that can uniquely identify Grid service instances.

XRL — XORP is an open source research project maintained by the XORP group [31]. The *protocol family* part of an XRL is defined by this group, and the implementations for new protocols (on XORP platform) can be realized through this group. In the runtime environment, the finder process works as the naming authority for handling *module names*, *interface names*, *interface version*, and *method names*.

INS — In INS, a service has the freedom to choose the attributes and values according to its requirements. However, to make the lookup process more efficient, a predefined root-level attribute named "name-space" is used to narrow down the scope of the search by fragmenting the entire namespace into "virtual namespaces."

Solar — Solar allows services to select their own names, so the existence of a central naming authority is not required. Both of the descriptive naming schemes (INS and Solar) do not assume a centralized naming authority, and applications can specify any attribute and value pairs. Although this offers a great deal of freedom in naming services, it raises an important question of how to discover services and how to ensure the completeness of a service discovery request. The most important issue, however, is one of name uniqueness, which is not ensured by the INS and Solar naming schemes as is, and is left to implementation. One possible solution is to have a set of predefined well known attribute-value pairs to facilitate the service discovery process. However, this would be equivalent to having a naming authority similar to other naming schemes. This issue has not been addressed clearly in these two schemes.

Name Persistence

UUID — Theoretically (and for all practical purposes), UUIDs are unique across both time and space, and hence can be used as static (persistent) names.

URI — URIs can be persistent or not (i.e. static or not), depending on the amount of location information they contain. For example, if a document's URI contains the IP of the machine on which the document is located, the URI will become invalid if the document is moved to a different machine. In order to keep the URIs of documents valid for as long as possible, it is recommended that they do not contain the author's name, the subject, status, access-control status, filename extension, or software mechanisms (e.g. cgi, exec). [54] Clearly, since URLs are location-dependent, it is common for URLs to become invalid when, for example, the name of their parent domain changes. URNs attempt to overcome this problem by providing location-independent static names that can be resolved into location-specific names such as URLs.

Web Services and Grid Services — In Web Services, UDDI entries are persistent since persistent UUIDs are used to identify the business and services. When a service moves to a different location or becomes unavailable, the access point can be updated or the UDDI entry can be removed from the registry. This will not change the UUIDs that are used to identify the services. A GSH is persistent since it is defined as a globally unique name that distinguishes a specific Grid Service instance from all other Grid Service instances that have existed, exist now, or will exist in the future [2, 55]. On the other hand, a GSR is not persistent since it is created with an expiry time, or may become invalid due to various reasons (e.g. the service access point has been moved to a different machine).

XRL — Unresolved XRLs are static. Whenever the interface definition for a module is changed, the *version number* changes as well. However, different versions of a *module name* can coexist on the same XORP platform, enhancing the persistence of XRLs. On the other hand, in resolved XRLs the module name is replaced by the network location (e.g., IP-address:portnumber pair) of the module. A resolved XRL, therefore, is not static because of the changes that can occur in this "network location" module.

INS — In INS, names associated with a service remain valid throughout the lifetime of the service. State information of services, such as current location (i.e. network address) and load, are kept separately in the associated name-records, hence a change in state does not change the name. In essence, INS offers location independence by storing the descriptive name of a service in the name-specifier and the actual network address information in the name-record.

Solar — Solar uses a dynamic naming scheme where some part of a name can be variable. A name containing variable component(s) has to be resolved each time it is used. The resolved name can change over time, depending on the state of the associated service.

Standardization and Implementation

Of the naming approaches examined here, UUID and URI stand out as the most standardized and accepted by the Internet community. Both are standardized by IETF, with URL having been embraced as the de-facto naming standard. In particular, the advantages of a URI-based approach is that it can be easily incorporated into the existing Internet naming-authority and name-resolution infrastructure. The more involved Web Service and Grid Service naming schemes are both based on URIs. The remainder of the approaches, while well-known, are experimental. See Table 1 for an illustration of this comparison.

Scheme	Status	Institution	References
UUID	Defined by a standards body or forum	IETF	[20]
URI	Defined by a standards body or forum	IETF; de facto standard	[8, 22, 23, 25]
Web services	Defined by a standards body or forum	W3C [56]	[28, 57, 58]
Grid services	Defined by a standards body or forum	OGSA [59] OASIS [60]	[55, 61]
XRL	Experimental Project	XORP Project [31]	[62]
INS	Experimental Project	MIT Laboratory for Computer Science	[32, 63]
Solar	Experimental Project	Dartmouth College	[6]

■ Table 1. *Standardization and implementation of naming approaches: comparison.*

Criterion	UUID	URI	Grid naming	WS naming	XRL	INS	Solar
Human-readable	No	Yes	Yes	Yes	Yes	Yes	Yes
Extensibility	No	Yes	Yes	Yes	Up to API	Yes	Yes
Namespace size	Finite (2^{128})	Infinite	Infinite	Infinite	Infinite	Infinite	Infinite
Naming authority	Fully distributed	Hierarchical	Hierarchical	Fully distributed or centralized	API-determined	Fully distributed	Fully distributed
Name resolution	DHT or app-specific	Hierarchical or DHT	Centralized registry	Centralized registry	Application-dependent	Replicated or DHT	Centralized and replicated
Name persistence	Static	URL: location-dependent; URN: static	GSR: location-dependent; GSH: static	Location-dependent	Static	Static	Dynamic
Standardization	Defined by IETF	Defined by IETF	Defined by W3C	Defined by OGSA	Experimental project	Experimental project	Experimental project

■ Table 2. *Comparison of naming approaches: summary.*

Guidelines for Selection

In this section we intend to evaluate the existing technologies using our selected criteria, and eventually select a set of candidates that fit well in our context and meet most, if not all, of our requirements.

The requirements of a multi-domain naming scheme are, most of all, scalability, extensibility, and flexibility. These criteria have been the driving force in our selection. Having studied a variety of naming schemes, we feel that in general, a naming scheme that would fare well in our vision:

- Has a human-readable name.
- Is fairly flexible and extensible.
- Has persistent names.
- Has location-independent names.
- Has a partitioned name structure with a limited set of descriptive attributes.
- Has a distributed naming authority.
- Has a distributed name resolution architecture.
- Is standardized and widely accepted in the Internet community.

Table 2 summarizes the naming approaches we chose to compare in this work. By comparing this summary to the illustration from Fig. 2, and to the list of requirements outlined above, we see that no single naming scheme meets all our desired criteria for a multi-domain environment. However, based on our background research into various naming approaches, as highlighted earlier, we feel that the naming schemes that are based on URIs and have a loose coupling between naming and name resolution, are the best-suited candidates for a multi-domain naming scheme. These schemes include URN, Web Service naming, and Grid Service naming schemes. It is worth noting that a combination of a candidate scheme with other naming schemes such as the UUID might offer richer features (through the possibility of a fully-distributed assignment of unique names) and is worth exploring.

While the choice of a naming mechanism for service management would ultimately be determined by the choice of the service discovery, management, and invocation mechanisms, the insights and innovations of other mechanisms we have studied in this article can serve to influence the finer points of the design of a naming and name resolution scheme.

Conclusions

This article has investigated the characteristics and challenges in designing a multi-domain, large-scale naming service. As a guideline for analysis, and to ease the readers through various aspects of a naming system, we first presented our view on the anatomy of a naming scheme along with naming design concerns and challenges. Following this discussion, we reasoned a set of design criteria that not only represents the fundamental issues of a naming scheme design, but also the unique challenges intrinsic to a multi-domain and large-scale context. More specifically, these requirements stress heterogeneity, flexibility, and a high degree of scalability.

Based on our design criteria, we performed a critical study of the well-known naming schemes in academia and industry, focusing on how each scheme fits our design criteria, and drew conclusions about their usefulness in a multi-domain, large-scale context. The results of our analysis lead us to believe that URI naming and the related Web Service and Grid Service naming schemes are the most suitable current candidate schemes. This suitability can be attributed to their scalability of naming resolution and name authority architectures, extensibility of the naming scheme, and the level of standardization of the selected approaches.

This article can serve as an aid for architects and system engineers in designing a large-scale naming system designed for a multi-service and multi-domain environment. The results of our investigation, as distilled in this article, indicate the current lack of such a naming system. Since this kind of system would be a crucial function block for any large-scale service framework, it remains an important research challenge.

ACKNOWLEDGMENT

This research is partially supported by Alcatel Canada and the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] H. Haas and A. Brown, “Web Services Glossary,” August 2003. Available at http://www.w3.org/TR/2003/WD-ws-gloss-20030808/

[2] I. Foster *et al.*, “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,” 2002. Available at citeseer.nj.nec.com/foster02physiology.html

[3] B. M. Hauzeur, “A Model for Naming, Addressing, and Routing,” *ACM Trans. Office Info. Sys.*, vol. 4, no. 4, Oct. 1986, pp. 293–311.

[4] L. Daigle *et al.*, “RFC 2611: URN namespace definition mechanisms,” June 1999. Available at http://www.ietf.org/rfc/rfc2611.txt?number=2611

[5] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design, 3rd Ed.*, Addison-Wesley, 2001.

[6] G. Chen and D. Kotz, “Context-Sensitive Resource Discovery,” *Proc. 1st IEEE Int’l. Conf. Pervasive Computing and Communications (PerCom’03)*, Mar. 2003, pp. 243–52.

[7] B. Traversat *et al.*, “Project JXTA 2.0 super-peer virtual network,” May 2003. Available at http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf

[8] T. Berners-Lee, L. Masinter, and M. McCahill, “RFC 1738 Uniform Resource Locators (URL),” Dec. 1994. Available at http://www.w3.org/Addressing/rfc1738.txt

[9] X. Project, “XORP Inter-process Communication Library Overview,” Nov. 2003. Available at http://www.xorp.org/releases/current/docs/libxipc/libxipc overview.pdf

[10] U. o. S. C. Information Sciences Institute, “RFC791: Internet Protocol,” Sept. 1981. Available at http://www.faqs.org/rfcs/rfc791.html

[11] K. Hubbard *et al.*, “RFC2050: Internet Registry IP Allocation Guidelines,” Nov. 1996. Available at http://www.ietf.org/rfc/rfc2050.txt

[12] R. Troll, “Automatically Choosing an IP Address in an Ad-Hoc IPv4 Network, IETF draft,” Jan. 1999.

[13] P. V. Mockapetris, “RFC 1035: Domain Names — Implementation and Specification,” Nov. 1987. Available at http://www.ietf.org/rfc/rfc1035.txt

[14] M. Ripeanu, I. Foster, and A. Iamnitchi, “Mapping the Gnutella Network: Properties of Large-scale Peer-to-peer Systems and Implications for System Design,” *IEEE Internet Comp. J.*, vol. 6, no. 1, 2002. Available at http://uchicago.edu/»matei/PAPERS/ic.pdf

[15] S. Ratnasamy *et al.*, “A Scalable Content Addressable Network,” *Proc. ACM SIGCOMM 2001*, 2001. Available at http://citeseer.nj.nec.com/ratnasamy01scalable.html

[16] G. Ballintijn, M. van Steen, and A. S. Tanenbaum, “Scalable Human-Friendly Resource Names,” *IEEE Internet Computing*, vol. 5, no. 5, pp. 20–27, Sept. 2001.

[17] K. Thomson, G. Miller, and R. Wilder, “Wide Area Internet Traffic Patterns and Characteristics,” *IEEE Network*, vol. 11, no. 6, Nov. 1997. Available at www.vbns.net/presentations/papers/MCItraffic.pdf, pp. 10–23.

[18] H. B. Jaeyeon Jung, E. Sit, and R. Morris, “DNS Performance and the Effectiveness of Caching,” *Proc. ACM SIGCOMM Internet Measurement Wksp. ’01*, San Francisco, California, Nov. 2001. Available at citeseer.nj.nec.com/article/jung01dns.html

[19] P. Sousa and A. Silva, “Naming and Identification in Distributed Systems: A Pattern for Naming Policies,” 1996. Available at http://citeseer.nj.nec.com/sousa96naming.html

[20] P. Leach, M. Mealling, and R. Salz, “Internet-draft: A UUID URN namespace,” Dec. 2004. Available at http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-05.txt

[21] “Google Search Engine.” Available at www.google.ca

[22] R. Moats, “RFC 2141: URN syntax,” May 1997. Available at http://www.ietf.org/rfc/

[23] T. Berners-Lee, R. Fielding, and L. Masinter, “RFC 2396: Uniform Resource Identifiers (URI): Generic syntax,” Aug. 1998, status: DRAFT STANDARD. Available at ftp://ftp.internic.net/rfc/rfc2396.txt

[24] “URN Namespaces — [RFC2141, RFC3406],” Feb. 2004. Available at http://www.iana.org/assignments/urn-namespaces

[25] T. joint W3C/IETF URI Planning Interest Group, “URIs, URLs, and URNs: Clarifications and Recommendations 1.0,” Sept. 2001. Available at http://www.w3.org/TR/uri-clarification/

[26] K. Best and N. Walsh, “RFC 3120 — a URN Namespace for XML.org,” June 2001. Available at http://www.faqs.org/rfcs/rfc3120.html

[27] C. Lynch, C. Preston, and R. Daniel, “Rfc 2288: Using Existing Bibliographic Identifiers as Uniform Resource Names,” Feb. 1998. Available at http://www.ietf.org/rfc/rfc2288.txt

[28] S. Luc Clement *et al.*, “UDDI version 3.0.2,” Oct. 2004, UDDI Spec Technical Committee Draft. Available at http://uddi.org/pubs/uddi_v3.htm

[29] A. Bosworth and *et al.*, “Web Services Addressing,” Mar. 2003, MSDN Library.

[30] K. Czajkowski *et al.*, “From Open Grid Services Infrastructure to wsresource Framework: Refactoring & evolution,” Dec. 2004. Available at http://www.ibm.com/developerworks/library/ws-resource/ogsi to wsrf 1.0.pdf

[31] X. Project, “XORP: Extensible Open Router Platform — Home Page.” Available at http://www.xorp.org

[32] W. Adjie-Winoto *et al.*, “The Design and Implementation of an Intentional Naming System,” *Symp. Operating Systems Principles*, 1999. Available at citeseer.nj.nec.com/adjie-winoto99design.html, pp. 186–201

[33] H. Balakrishnan *et al.*, “A Layered Naming Architecture for the Internet,” *Proc. ACM SIGCOMM ’04 Conf.*, Portland, OR, Aug. 2004. Available at http://www.acm.org/sigs/sigcomm/sigcomm2004/papers/p497-balakrishnan111.pdf.

[34] J. Kunze, “RFC 1736 — Functional Recommendations for Internet Resource Locators,” Feb. 1995. Available at http://www.ietf.org/rfc/rfc1737.txt

[35] L. M. K. Sollins, “RFC 1737: Functional Requirements for Uniform Resource Names,” Dec. 1994. Available at http://www.ietf.org/rfc/rfc1737.txt

[36] “International Standard Book Number.” Available at http://www.isbn.org

[37] W. M. Duerst and M. C. M. Suignard, “RFC 3987: Internationalized Resource Identifiers (IRIs),” Jan. 2005. Available at http://www.ietf.org/rfc/rfc3987.txt

[38] V. Ramasubramanian and E. G. Sirer, "The Design and Implementation of a Next Generation Name Service for the Internet," *Proc. 2004 Conf. Applications, Technologies, Architectures, and Protocols for Comp. Commun.*, ACM Press, 2004, pp. 331–42.
[39] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-scale Peer-to-peer Systems," Lecture Notes in Computer Science, vol. 2218, 2001. Available at http://research.microsoft.com/»antr/PAST/pastry.pdf, pp. 329–50.
[40] V. Ramasubramanian and E. G. Sirer, "Beehive: Exploiting Power Law Query Distributions for O(1) Lookup Performance in Peer to Peer Overlays," *Proc. Symp. Networked Systems Designs and Implementation*, Mar. 2004.
[41] K. Sollins, "RFC 2276: Architectural Principles of Uniform Resource Name resolution," Jan. 1998, status: INFORMATIONAL. Available at ftp://ftp.internic.net/rfc/rfc2276.txt
[42] M. Mealling and J. R. Daniel, "RFC 2483 — URI Resolution Services Necessary for URN Resolution," Jan. 1999. Available at ftp://ftp.internic.net/rfc/rfc2483.txt
[43] M. Mealling and R. Daniel, "RFC 2915: The Naming Authority Pointer (NAPTR) DNS Resource Record," Sept. 2000. Available at http://www.ietf.org/rfc/rfc2915.txt
[44] "Z39.50 International Standard Maintenance Agency." Available at http://www.loc.gov/z3950/agency/
[45] R. Daniel, "RFC2169: A Trivial Convention for using HTTP in URN Resolution," June 1997. Available at http://www.ietf.org/rfc/rfc2169.txt
[46] K. Moore *et al.*, "Resource Cataloging and Distribution System (RCDS)," University of Tennessee, Tech. Rep., Jan. 1997.
[47] J. Bosak, "Proposal for a Language Optimized for WWW Delivery." Available at http://www.w3.org/Style/History/www.ora.com/davenport/HDL/hdl.proposal.html
[48] "Referral whois (RWHOIS)." Available at http://www.rwhois.net/
[49] M. Balazinska, H. Balakrishnan, and D. Karger, "Ins/twine: A Scalable Peer-to-peer Architecture for Intentional Resource Discovery," *Proc. 1st Int'l. Conf. Pervasive Comp.*, Springer-Verlag, 2002, pp. 195–210.
[50] I. Stoica *et al.*, "Chord: A Scalable Peer-to-peer Lookup Service for Internet applicatçions," *Proc. ACM SIGCOMM '01 Conf.*, San Diego, California, Aug. 2001. Available at http://www.pdos.lcs.mit.edu/chord/papers/paper-ton.pdf
[51] P. J. Leach and R. Salz, "INTERNET-DRAFT: UUIDs and GUIDs," Feb. 1998. Available at http://hegel.ittc.ukans.edu/topics/internet/ internet-drafts/draft-l/draftleach-uuids-guids-01.txt
[52] www.iana.org, "Uniform Resource Identifier (URI) Schemes." Available at http://www.iana.org/assignments/uri-schemes
[53] "Internet Corporation for Assigned Names and Numbers." Available at http://www.icann.org/
[54] T. Berners-Lee, "Cool URIs Don't Change," 1998. Available at http://www.w3.org/Provider/Style/URI
[55] S. Tuecke *et al.*, "Open Grid Services Infrastructure (OGSI) Version 1.0," June 2003.
[56] "World Wide Web Consortium." Available at http://www.w3.org/
[57] "Soap version 1.2 part 0: Primer," June 2003. Available at http://www.w3.org/TR/2003/REC-soap12-part0-20030624/
[58] E. Christensen *et al.*, "Web Service Definition Language." Available at http://www.w3.org/TR/wsdl
[59] I. Foster *et al.*, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure wg, Global Grid Forum," 2002. Available at http://www.globus.org/research/papers/ogsa.pdf
[60] "Organization for the Advancement of Structured Information Standards (OASIS)." Available at http://www.oasis-open.org/home/index.php
[61] K. Czajkowski *et al.*, "The WS-Resource Framework Version 1.0," Mar. 2004. Available at http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf
[62] M. Handley, O. Hodson, and E. Kohler, "XORP: An Open Platform for Network Research," 2002. Available at http://cir.org/mjh/xorphotnets.pdf
[63] "INS/Twine Software Distribution." Available at http://nms.lcs.mit.edu/software/instwine
[64] E. Kidd, "XML-RPC How to," 2001. Available at http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html
[65] "DCE Distributed Computing Environment." Available at http://www.opengroup.org/dce/
[66] O. M. Group, "Naming service, v1.2," Sept. 2002. Available at http://www.omg.org/technology/documents/formal/naming service.htm
[67] "Distributed Component Object Model DCOM." Available at http://www.microsoft.com/com/tech/DCOM.asp
[68] M. Wahl, T. Howes, and S. Kille, "Lightweight Directory Access Protocol (v3)," Dec. 1997. Available at http://www.ietf.org/rfc/rfc2251.txt
[69] G. Good, "RFC2849: The LDAP Data Interchange Format (LDIF) — Technical Specification," June 2000. Available at http://www.ietf.org/rfc/rfc2849.txt
[70] T. Howes and M. Smith, "RFC 2255:the LDAP URL format," Dec. 1997. Available at http://www.ietf.org/rfc/rfc2255.txt
[71] U. Warrier and L. Besaw, "RFC 1095: The Common Management Information Services and Protocol over TCP/IP (CMOT)," Apr. 1989. Available at http://www.ietf.org/rfc/rfc1095.txt
[72] DMTF, "Desktop Management Interface Specification," Distributed Management Task Force, Tech. Rep., Jan. 2003.
[73] "Distributed Management Task Force DMTF." Available at http://www.dmtf.org
[74] K. McCloghrie, D. Perkins, and J. Schoenwaelder, "RFC2578: Structure of Management Information Version 2 (SMIv2)," Apr. 1999. Available at http://www.ietf.org/rfc/rfc2578.txt
[75] D. M. T. Force, "Common Information Model (CIM) Standards." Available at http://www.dmtf.org/standards/cim/
[76] S. Shepler *et al.*, "RFC 3530: Network File System (NFS) Version 4 Protocol," Apr. 2003. Available at http://www.ietf.org/rfc/rfc3530.txt
[77] R. Sharpe, "Just What is SMB?" Oct. 2002. Available at http://samba.org/cifs/docs/what-is-smb.html
[78] A. S. Tanenbaum and M. van Steen, Distributed Systems: Principles and Paradigm, Pearson Education, Sept. 2001.
[79] "Coda File System." Available at http://coda.cs.cmu.edu/
[80] J. Bester *et al.*, "GASS: A Data Movement and Access Service for Wide Area Computing Systems," *Proc. 6th Wksp. Input/Output in Parallel and Distributed Sys.*, Atlanta, GA: ACM Press, 1999. Available at citeseer.ist.psu.edu/bester99gass.html, pp. 78–88.

BIOGRAPHIES

REAZ AHMED (r5ahmed@bbcr.uwaterloo.ca) is a Ph.D. student at the School of Computer Science at the University of Waterloo, Canada. His interests include service discovery architectures, distributed indexing schemes, and peer-to-peer networks.

RAOUF BOUTABA (rboutaba@bbcr.uwaterloo.ca) received the M.S. and Ph.D. degrees in computer science from the University Pierre Marie Curie, Paris, France, in 1990 and 1994, respectively. He is currently an associate professor in the School of Computer Science at the University of Waterloo. Before that he was with the Department of Electrical and Computer Engineering at the University of Toronto. Before joining academia, he founded and was the director of the telecommunications and distributed systems division of the Computer Science Research Institute of Montreal (CRIM). He conducts research in the areas of network and distributed systems management and resource management in multimedia wired and wireless networks. In these areas, he has published more than 150 papers in refereed journals and conference proceedings. He is the recipient of the Premier's Research Excellence Award, the NORTEL Networks research excellence Award, and several Best Paper awards. He is a fellow of the faculty of mathematics at the University of Waterloo, has served as a distinguished lecturer of the IEEE Computer Society, and is currently a distinguished lecturer of the IEEE Communications Society.

FERNANDO CUERVO (fernando.cuervo@alcatel.com) has a bachelor degree in electrical engineering from the Universidad de Los Andes, Colombia, and a master of science in computer science from the University of Western Ontario. He is currently a product manager in Alcatel's IP Network Management division. He has

transitioned from Research & Innovation, where he was for the past two years a senior researcher for the Next Generation Network and Service Management strategic project. Within this project he was the chief architect for the Alcatel Policy Architecture and the Cross Domain Integration Project. These projects address the needs of operators to be more efficient providing flexible network services and service management. Fernando's experience covers a wide range of management topics, including, information modeling, performance of management systems, management process re-engineering, workflow integration, and integration of management and control. He has also contributed to standards and industry fora such as IETF and MSForum.

YOUSSEF IRAQI (iraqi@bbcr.uwaterloo.ca) received the B.Sc. in computer engineering with high honors from Mohamed V University, Morocco, in 1995. He received M.S. and Ph.D. degrees in computer science from the University of Montreal in 2000 and 2003, respectively. He is currently a research assistant professor at the School of Computer Science at the University of Waterloo. From 1996 to 1998 he was a research assistant at the Computer Science Research Institute of Montreal, Canada. His research interests include network and distributed systems management, resource management in multimedia wired and wireless networks, and peer-to-peer networking.

TIANSHU LI (dtianshu@bbcr.uwaterloo.ca) received the B.Sc. degree in computer science from University of Victoria, Canada in 2001. He received his M.Math degree in computer science from the University of Waterloo, Canada in 2003, where he also worked as a research assistant on several projects. He now works at Kitware Inc. in Clifton Park, NY, USA. His current research interests include grid computing, Web-based computing, parallel computing, and applying distributed computing techniques in large-scale scientific computing and visualization.

NOURA LIMAM (nlimam@bbcr.uwaterloo.ca) received the B.S. degree from the National School of Computer Science, Tunisia, in 2001, and the M.S. degree in networking from the University Pierre Marie Curie, Paris, France, in 2002. She is working toward the Ph.D. degree and is currently a research assistant at the University of Waterloo, Canada. Her research interests include network and service management, service discovery, and service-oriented architectures.

JIN XIAO (j2xiao@bbcr.uwaterloo.ca) is a Ph.D. student at the School of Computer Science at the University of Waterloo, Canada. His primary research interests are in autonomous network management intelligent service provisioning. Recently, he has been working on QoS-aware service composition for multi-domain networks. He is a student member of the IEEE.

JOANNA ZIEMBICKI (jiziembi@bbcr.uwaterloo.ca) received a BMath degree in pure mathematics and computer science (co-op) from the University of Waterloo in 2003. She is now working toward her M.Math degree at the School of Computer Science at the University of Waterloo. Her research interests include Web semantics, service discovery, peer-to-peer networking, and large-scale network management.

APPENDIX 1: OTHER NAMING SCHEMES

While this article attempts to provide a good survey of naming approaches, it omits a number of technologies deemed irrelevant or unsuitable to a multi-domain service-infrastructure setting. In this Appendix we describe some of the omitted approaches and the reasons for their unsuitability for this article.

HUMAN-FRIENDLY NAMES

Currently, most resources on the Internet are identified with Universal Resource Locators (URLs), which identify both the resource and its location/access method. The drawback associated with this dual function is that it makes it difficult to assign the same identifier to widely-distributed multiple replicas of a resource, and that it ties the identifier to a specific location. URNs (which resolve to URLs) have remedied these drawbacks by creating a name that identifies only the resource, without specifying the replica identity or the location. However, URNs are not required to be human-readable (only human-transcribable), therefore they may still be difficult for humans to use.

Ballintijn, van Steen, and Tanenbaum [16] have proposed a naming scheme that builds on the present URN/URL system, and uses descriptive human-friendly names (HFNs) which resolve to URNs that identify resources. This scheme allows for each HFN to resolve to one or more URNs, each of which can resolve to one or more URLs. Because DNS is used for name resolution in this scheme, HFNs have a similar format to URNs.

The HFN scheme is designed specifically for highly-popular and replicated Web resources, and is not suitable for a large number of lesser-used resources or highly-mobile resources. It also provides an extra level of name resolution onto the URN-to-URL infrastructure, thus adding a significant overhead. These restrictions make this approach inappropriate for a system that identifies a large variety of services and resources which may or may not require to be directly usable by humans.

IPC MECHANISMS

Several mechanisms exist for enabling (procedure-oriented and object-oriented) inter-process communication in distributed applications. In addition to naming objects, these mechanisms include facilities for remote procedure calls, and often for advanced access-control and security. These mechanisms are often quite complex and require significant effort to implement. They are better suited to tightly coupled enterprise and desktop applications rather than to distributed Web applications [64]. Also, the naming schemes included with these systems are designed specifically for naming software components, and may not have the flexibility required for naming a large variety of services and resources.

In this article we have examined XRL, a language for naming entities for the eXtensible Open Routing Platform, which is designed specifically for routing systems. We have chosen to include XRL in the survey due to its applicability to routing, and hence to cross-domain applications. Other IPC mechanisms include the Distributed Computing Environment (DCE) [65], Common Object Request Broker Architecture (CORBA) [66] (which uses the Interface Definition Language), and Microsoft's Distributed Common Object Model (DCOM) [67].

DIRECTORY SYSTEMS

Another approach to naming objects is through directory systems such as X.500, or its more lightweight grandchild, Lightweight Directory Access Protocol (LDAP) [68]. LDAP uses the LDAP Data Interchange Format [69] to store directory entries, which are sets of attribute-value pairs, including a *distinguished name* that combines several of the a-v pairs to uniquely identify each entry. LDAP supports a variety of operations, including searching and updating; a format exists to express these operations in the form of a URL [70]. While LDAP can be used to store information about a variety of entities, its most natural function is to store phonebook-type information about people and institutions. Moreover, its purpose is better geared toward description rather than naming itself.

NETWORK MANAGEMENT NAMING SYSTEMS

No discussion related to management of network services would be complete without mentioning network management naming systems such as SNMP, CMIP, DMI, and CIM, which incorporate mechanisms for uniquely identifying and accessing entities.

The Simple Network Management Protocol (SNMP) is the de-facto standard for managing network and computer system devices. A more powerful system, Common Management Information Protocol (CMIP) [71], has been devised for telecom devices; however, few implementations exist because of its complexity and resource overhead. Desktop Management Interface (DMI) [72] was designed by the Distributed Management Task Force (DMTF) [73] as a method to standardize obtaining and managing information about the internals of a desktop system. All of the above approaches use the ASN.1 registration tree for assigning globally unique Object Identifiers, which can take the form of a dot-separated number or translated to more human-readable dot-separated strings. While the sole purpose of this registration hierarchy is to assign unique identifiers, the systems that use this hierarchy assign their own set of inheritance and containment relationships on the tree. The Management Information Base (MIB) specifies the collection of related objects in specific devices, following the rules set out by the Structure of Management Information (SMI) model [74].

CIM, The Common Information Model (CIM) [75], is an object-oriented naming scheme designed for management information for systems, networks, and applications. It unifies existing standards (SNMP, DMI, CMIP) and uses a standard language — *Managed Object Format*. The model replaces the Management Information Bases with Management Object Files (MOF) claimed to make it easier to track the relationships between managed objects, and, like SNMP, supports model operation methods. CIM strives to be more comprehensive and extensible than the previous standards.

While these network management systems provide a rich set of functionalities for naming and managing network and software resources, the included naming systems identify the *types* of entities rather than the entities themselves. Moreover, these types of systems are notoriously complex, rigid, and difficult to implement.

FILE SYSTEM NAMING SCHEMES

Distributed file systems provide a method of naming resources (files) across local-area or wide-area networks. The classic example is the Network File System [76] developed by Sun Microsystems, which allows a computer to access files over a network as if they were on its local disks. Other approaches include the Common Internet File System (CIFS) [77], the Andrew File System (AFS) [78], and Coda [79].

Most distributed file systems provide a location-independent hierarchical naming scheme, which closely follows the familiar conventions of a single-machine file system, and may or may not include facilities for context-sensitive variable file name components. Distributed file systems are generally designed for local-area networks, or at most single-domain wide-area networks. A larger-scale attempt, GASS (Global Access to Secondary Storage) [80], which forms a part of the Globus toolkit for Grid applications, defines a global name space via Uniform Resource Locators. However, even those file systems designed for wide-area use provide facilities for naming only a narrow set of resources: files and those resources that can be identified with files.