Naming in Distributed Systems

OGSA Naming Design Team working note OGSA-Naming-02.

December 10, 2004

Andrew Grimshaw

(Edited by Dave Snelling)

Introduction

The objective of this working note is to begin the discussion on a "WS-Naming" standard. To stimulate discussion, and provide background, we begin with a short section on terms and desirable properties in a classic two-level name scheme – abstract names to addresses. This document serves a second purpose in providing input to the OASIS/WSRF TC with respect to requirements for the proposed Renewable References specification. It is understood Renewable References address only a subset of the full requirements set for Naming, but the requirements addressed by Renewable References may come very close, hence this full set of requirements for naming is being presented to the WSRF TC.

Naming in distributed systems has a rich history and literature – and the basics are well understood. At the end of the document are references to a set of extant naming schemes. It is important to understand those schemes because it is likely that one of these can be adopted.

David Snelling 10/12/04 18:06
Deleted: very

David Snelling 10/12/04 18:07
Deleted: other

David Snelling 10/12/04 18:08
Deleted: simply

Traditional distributed systems often have a three layer naming scheme. Human names[1] such as paths or attributes are mapped to abstract names, which are then mapped to some form of address. In this document we are not concerned with human names. Rather we are interested in the abstract name to address mapping.

Terms

Resource – A resource is a namable entity that accepts a set of method calls specified in an IDL such as WSDL. A resource may be an instance of some class or template (itself a resource) – but this is not required. How resources come into existence is not an issue for the purposes of naming. Resources may have metadata or attributes associated with them. It is useful if resources have a set of common basic methods such as get_name, get/set

David Snelling 10/12/04 18:37
Deleted: that

[1] In general "Human Names" will be human readable, but human names are not synonymous with "printable names." By human names we typically mean names assigned and understood by humans. This implies printable, but also assumes some semantic content within the name, which is meaningful to humans within some context.

attribute, and get_interface. The distinction between stateful and stateless services is irrelevant in naming schemes – though it may impact the implementation of services.[2]

Abstract resource name: an abstract name should not rely on any location, type, implementation, or number information. This ensures that the abstract name can persist across resource migration, container restart, etc.

David Snelling 10/12/04 18:39
Deleted: include

Resource Address: A resource address can be used directly to communicate with the resource using some protocol. A resource address refers to a particular communication endpoint and may include or imply the protocol to use. In this document we will assume that the "address" is an EPR as defined in the WS-Addressing standard.

David Snelling 10/12/04 18:45
Deleted: in this case

Resource identity. An identity not only uniquely identifies a resource – as does a resource address, an identity also has authentication properties that permit verification (usually via cryptographic means) of the entity. Note that the notion of identity is orthogonal to that of naming, but is defined here for clarity.

Human names. The two most frequently used examples of human names are paths such as /home/grimshaw/datafile, and properties (a.k.a. attributes and metadata), e.g., "invoice where modification date=9-15-03 and value<$45.00".

David Snelling 10/12/04 18:49
Deleted: cost

Binding scheme: The mechanism that "binds" abstract names to resource addresses, i.e., given an abstract name the binding scheme returns a resource address that can be used to communicate with the resource.

Bind time: The point at which an abstract name is bound to a resource address. Note that this can happen at many different times, e.g., at compile time, at program load time, at first use in a program, on failure of an old binding, and/or on every use. For all but the last case, on every use, we assume that the binding may be cached somehow in the callers context. In some schemes, binding can be an expensive operation, thus there is a trade-off in bind time decisions between performance and dynamics.

David Snelling 10/12/04 20:16
Deleted: is

David Snelling 10/12/04 20:16
Deleted: address space

David Snelling 10/12/04 20:17
Deleted: Binding

Name granularity: This defines what the named entities are, e.g. large (whole files), small (fields in a data base or member variables in a class), or somewhere in between. The granularity should be as large as possible (for effeciency) but small enough to support desired application semantics.

David Snelling 10/12/04 20:17
Deleted: What are the named entities?

David Snelling 10/12/04 20:18
Deleted: L

Motivation

Distributed systems research has a rich literature. In the literature virtualization of resources and objects is a common solution to many problems. Naming is essential to the creation of virtualization. This virtualization results in a "transparency". Nine of these show up again and again, and have been called the "nine golden transparencies". They are used with respect to accessing a remote service or object. In all cases the intent is that

---

[2] The WSRF WS-Resource specification defines Resource normatively in the context of the Resource Access Pattern, see http://docs.oasis-open.org/wsrf/2004/09/wsrf-WS-Resources-1.2-draft-01.pdf. The above definition is consistent with this definition.

David Snelling 10/12/04 18:34
Formatted: Title page info description

the programmer/user need not know about or deal with an object or entity, but can do so if they want to. These transparencies are:

Access. The mechanism used for a local invovation is the same as for a remote invocation. Many have argued this is a special case on location transparency.

Location. The caller need not know where the object is located, California or Virginia it makes no difference.

Failure. If the object or service fails the caller is unaware. Somehow the requested service or function is performed, the object is restarted, or whatever is needed.

Heterogeneity. Architecture and OS boundaries are invisible. Objects can migrate without limitation due to system architecture or operating environment. At a bare minimum communication with objects on other architectures requires no explicit data coercion.

Migration. The caller need not know whether an object has moved since they last communicated with it.

Replication. An object or service's function can be provided by one or many instances without the caller being aware - the same name applies to any and all instances simultaneously. The caller need not know about nor deal with coherence issues.

Concurrency. The caller need not be aware of other callers invocating operations on a given object or service.

Scaling. An increase or decrease in the number of servers requires no change in the software. Naturally performance may vary.

Behavioral. Whether an actual object or a simulation of the object is used is irrelevant to the caller. Similarly – the existane an object or service is immaterial as one may be created as needed.

Use Cases

Several of the transparencies are non-controversial and are captured by existing Web Services best practices, e.g., behavioral, heterogeneity, and concurrency. Persistent or abstract naming are required to provide other transparencies. The following use cases highlight these issues.

Migrate closer to a heavy user. Suppose that a client application is making intense use of a service or resource, and that the client and resource are widely separated, inducing a large latency. For example, an application in California reading and modifying a file in New York that other clients may need to continue to access – though less intensely. One would like to be able to migrate the file resource from New York to California without any interruption in service.

Migrate away from failing or overloaded resource. Consider a service or resource executing on a host that is heavily loaded (either the host, or perhaps the network into the site where the host is located). We would like to be able re-schedule the execution to another host without interrupting the service and without disrupting on-going interactions with this and other services and resources. Similarly, it may be known that a host is going to “go down” soon, either because of maintenance, or perhaps problems with the physical

David Snelling 10/12/04 22:23
Deleted: something

David Snelling 10/12/04 22:24
Deleted: The

David Snelling 10/12/04 22:24
Deleted: call

David Snelling 10/12/04 22:24
Deleted: call

David Snelling 10/12/04 22:28
Deleted: cannot

David Snelling 10/12/04 22:32
Deleted: Is there one object or many objects behind the name?

David Snelling 10/12/04 22:33
Deleted: or

David Snelling 10/12/04 22:34
Deleted: Are there other concurrent users of an object?

David Snelling 10/12/04 22:34
Deleted: of them.

David Snelling 10/12/04 22:35
Deleted: Is it live or is it Memorex?

David Snelling 10/12/04 22:36
Deleted: is there

David Snelling 10/12/04 22:36
Deleted: there at all

David Snelling 10/12/04 22:37
Deleted: – or is

David Snelling 10/12/04 22:40
Deleted: CA

environment (power shortage, air conditioning failure, etc.). Once again we want to be able to migrate the service to another location without interrupting on-going interactions.

Recovery from a failed resource. Consider a stateful resource that has failed (either due to underlying hardware failure, or perhaps a software failure) and needs to be restarted – perhaps on a different physical resource. We want to be able to "migrate" the active instance to a different location or machine.[3]

Replica management and usage. Some resources may have multiple "back-ends", endpoints that can each perform the service as well as the other. We would like to be able to dynamically select which replica to use. For example, one replica may be closer (in network terms) than another. Or, one may offer better QOS in some dimension (e.g., performance).

Desirable properties of abstract naming and binding scheme.

Required:

The following are required by many Grid like use cases similar to the above. There is some expectation that all of these might be met by current proposals for Renewable References.

Unique – If two names are the "same", they refer to the "same" resource. "Sameness" is defined by the semantics of the service that the resource provides. Further, names must be unique in space and time, and therefore names cannot be reused.

Persistent – The name for a resource is valid as long as the resource exists.

Location portable – An abstract name can be used anywhere and will refer to the same resource from any context.

Support the usual transparencies – Location, migration, failure, replication, and implementation.

Extensible – To accommodate future requirements a naming scheme should be extensible.

High-performance – Low performance naming schemes prevent scalability and usability. If the scheme is too slow, it will not be used, defeating the purpose.

Dynamic binding – Binding an abstract name to an address is not static, i.e., it can change. The speed at which re-binding takes place is critical – if it takes hours to rebind, then resource mobility is highly constrained.

Scalable binding – We expect the scale of future distributed systems to be very large. The binding scheme must scale with the system. Often this can be achieved with caching. However, caching must be balanced against the need to have dynamic bindings.

---

[3] How the state management between the failed instance is kept synchronized with the replica is not the issue here. There are many well-known techniques, periodic checkpoints, message logging, etc. The important fact is that naming facilitates this process.

David Snelling 10/12/04 22:43
Deleted: (How the state management between the failed version is kept synchronized with the replicant is not the issue here. There are many well-known techniques, periodic checkpoints, message logging, etc.)

David Snelling 10/12/04 22:46
Deleted:

David Snelling 10/12/04 22:46
Deleted: (Once again we ignore here the issue of state consistency between replicants. There are well-known techniques.)

David Snelling 10/12/04 22:46
Deleted: nt

David Snelling 10/12/04 22:46
Deleted: nt

David Snelling 10/12/04 22:48
Deleted: they

David Snelling 10/12/04 22:48
Deleted: should

David Snelling 10/12/04 22:49
Deleted: . Thus,

David Snelling 10/12/04 23:04
Deleted: implementation

David Snelling 10/12/04 23:06
Deleted: name

David Snelling 10/12/04 23:06
Deleted: Nobody wants a low

David Snelling 10/12/04 23:09
Deleted: (

David Snelling 10/12/04 23:08
Deleted: ,

David Snelling 10/12/04 23:09
Deleted:)

David Snelling 10/12/04 23:11
Deleted: 's performance (e.g., bindings per second or time to bind)

Language neutral. Some naming schemes were developed specifically to support particular programming languages.

Concurrent name generation. This is an aspect of scalability. It must be possible to partition the name space, and generate names concurrently.

Large name-space. Running out of names is not acceptable.

Desirable:

Identity – The name can also act as an identity.

Authentication – Mutual authentication between two named resources without requiring a trusted third party has advantages, particularly with respect to scalability.

Comparable – Given two different names – it is possible to determine equality, ideally without having to contact either the resource or some other third party.

Widely adopted – The usefulness of a naming scheme increases when it is widely used.

Not require trust – Schemes that do not require a trusted third party are, all else being equal, preferred over those that do require a trusted third party.

Free. The Grid community expects open software license with no license fee.

Human readable and printable. Often it will be necessary to print a name.

David Snelling 10/12/04 23:12
Deleted: *Efficient/concurrent*

David Snelling 10/12/04 23:13
Deleted: You don't want to run

David Snelling 10/12/04 23:03
Deleted: have

David Snelling 10/12/04 23:15
Deleted: Whom do you trust?

David Snelling 10/12/04 23:16
Deleted: Open software license with no license fee

David Snelling 10/12/04 23:16
Deleted:

Naming schemes in the recent past

Note that it may be possible to use an existing scheme rather than invent a whole new one. Some existing schemes, which address some or all of these requirements, include:

OGSI (GSH-GSR).

WS-Addressing – http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/

DNS – http://www.ietf.org/rfc/rfc1034.txt?number=1034

Handle.net – http://www.handle.net/

SGNP – http://www.globalgridforum.org/Meetings/ggf4/bofs/SGNP%20-%20GWD-R%202002.02.05.pdf

LSID

WSRF and RR

URI/URN/URL

JXTA