2005

Towards persistent resource identification with the uniform resource name

Luke Brown
Edith Cowan University

Towards Persistent Resource Identification with the Uniform Resource Name

Luke Brown

A thesis submitted in partial fulfillment of the degree

of

Bachelor of Computer Science (Honours) at

School of Computer and Information Science

Edith Cowan University

November 2005

I certify that this thesis does not, to the best of my knowledge and belief:

(i) incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education;
(ii) contain any material previously published or written by another person except where due reference is made in the text of this thesis; or
(iii) contain any defamatory material;

Luke Brown

28 November 2005

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Acknowledgements

This work would not have been possible without the assistance, encouragement and occasional derogatory levelling of my supervisors, friends and family.

Thanks to the ECU School of Computer Science, most notably the Mike's Johnstone and Collins. Thanks also to the team at the ANU Supercomputing Facility, especially to Markus Buckhorn and Bob Gingold for suggesting and supporting this project.

Thanks to my friends of both coasts for ensuring my procrastination was never just wasting time. Thanks especially to Easties Teresa, Jules, Tom, Al, Cole and Tim, and Westies Tom, Andy, Mike and Burgie.

Thanks also to my family for supporting and encouraging my entire education, from 123 through to URN.

Special thanks to Tessa - for always being there, no matter how far apart.

Abstract

The exponential growth of the Internet, and the subsequent reliance on the resources it connects, has exposed a clear need for an Internet identifier which remains accessible over time. Such identifiers have been dubbed persistent identifiers owing to the promise of reliability they imply.

Persistent naming systems exist at present, however it is the resolution of these systems into what Kunze, (2003) calls "persistent actionable identifiers" which is the focus of this work. Actionable identifiers can be thought of as identifiers which are accessible in a simple fashion such as through a web browser or through a specific application.

This thesis identifies the Uniform Resource Name (URN) as an appropriate identification scheme for persistent resource naming. Evaluation of current URN systems finds that no practical means of global URN resolution is currently available.

Two new approaches to URN resolution, unique in their use of the Domain Name System (DNS) are introduced.

The proposed designs are assessed according to their Usability, Security and Evolution and an implementation described for an example URN namespace of language identifiers.

Contents

Chapter 1

Introduction

1.1 Motivation

The exponential growth of the Internet, and the subsequent reliance on the resources it connects, has exposed a clear need for an Internet identifier which remains accessible over time. Such identifiers have been dubbed persistent identifiers owing to the promise of reliability they imply. Persistent identifiers provide means to "track a specific object regardless of its physical location or current ownership" (CENDI, 2004).

The brittle nature of the Uniform Resource Locator (URL) - responsible for most Internet identification at present - makes the need for persistent Identification quite immediate in several application areas. The URLs primary shortcoming is the location-centric approach taken in its design which leaves URLs subject to failure when resource locations change - resulting in the all too common error 404, "Not Found" (Fielding et al, 1999).

Fortunately, the URL is not the only option for resource identification on the Internet. The Uniform Resource Name (URN), developed concurrently with the URL, was devised as a means of location independent resource naming. The URN facilitates persistent naming whilst remaining human readable, unique and manageable.

Despite the URNs presence, widespread use is being delayed by the lack of

means to resolve URNs into the URLs they identify. The primary aim of this thesis is to investigate a solution to this problem toward the goal of facilitating the use of URNs for persistent identification.

1.2 Approach

Despite the weaknesses of the URL as an identifier, its successful use of Internet domain names as locators has led to the development of mature resolution through the Domain Name System (DNS). The DNS is a hierarchically distributed database of mappings between domain names and Internet Protocol (IP) addresses.

Given the maturity and widespread adoption of the DNS it is the intention of this work to describe and implement a resolution system for the URN which leverages the DNSs functionality. The DNS has already faced and solved several major obstacles such as that of security, load balancing and redundancy, issues that any new distributed database system would have to address. The DNS is also suitable as an open standard which is implemented and available on most modern operating systems as a standard feature. DNS servers would not require software changes for a URN resolver, simply the addition of new records.

The notion of URN resolution via the DNS is not entirely unique. Previously, a proposal before the Internet Engineering Task Force (IETF) known as the Dynamic Delegation Discovery System (DDDS) (Mealling, 2002) suggested using the DNS to take URNs as input and return a server or list of servers able to resolve that URN into a URL.

This work intends to establish through prototyping the viability of the DDDS proposal and, through development of the DDDS prototype, the viability of

complete URN resolution using the DNS.

1.3 Contribution

Persistent naming systems exist at present. However, it is the resolution of these systems into what Kunze, (2003) calls "persistent actionable identifiers" that is the focus of this work. Actionable identifiers can be thought of as identifiers which are accessible in a simple fashion, such as through a web browser or through a specific application. There are currently no persistent identifier systems which are actionable without the use of a proxy resolver service.

An actionable persistent identifier will have widespread applications in areas of industry and research which value the ability to reliably access and share data collections. Such areas include the Digital Library movement, data intensive sciences such as experimental particle physics and various e-commerce applications.

The proceeds of this research will arm software developers with sufficient specifications and working implementations to deploy a URN resolver client. Further, it will enable network administrators to populate their existing DNS zones with data for resolving URNs.

1.4 Organisation

Chapter 2 introduces the research questions this thesis seeks to answer. The research methods which will be employed to answer these questions are discussed.

Chapter 3 presents an overview of identification on the Internet, introducing

the Uniform Resource Identifier (URI) and its subclasses the Uniform Resource Locator (URL) and Uniform Resource Name (URN).

Chapter 4 explores the requirements for URN resolvers and the present means of persistent identification available on the Internet.

Chapter 5 introduces the Dynamic Delegation Discovery System (DDDS), a proposed system for the discovery of authoritative URN resolvers. This design is assessed according to the guidelines outlined in Chapter 3.

Chapter 6 outlines a series of extensions to the DDDS which form the Extended Dynamic Delegation Discovery System (EDDDS). These extensions are also assessed according to the guidelines outlined in Chapter 3.

Chapter 7 presents a proof of concept experiment involving implementation of the DDDS and EDDDS designs. Results are assessed in terms of the outcome for users of these systems.

This paper concludes with an overview of the contributions made by this work and the future work needed in this field.

Appendix A lists an implementation of the DDDS and EDDDS designs in Java.

Chapter 2

Research Design

This thesis seeks to develop a usable means of persistent Internet identification. Previous attempts at persistent identification have shared a common lack of effective means for name resolution. Of the various identification schemes available, the Uniform Resource Name (URN) is considered to be the best suited to this task.

This thesis aims to achieve this goal of enabling resolution by assessing the viability of resolving URNs using the Domain Name System. The success of this assessment relies upon the effective selection and adoption of research methods to answer specific research questions.

Providing a resolution system for the URN can be achieved through design and implementation of a prototype or "proof of concept" resolver. Development of such a prototype will prove that such resolution is possible. A second, more detailed consideration is required to determine the practicality of the resolver system developed. This consideration will be provided through implementation of a URN resolver for a specific namespace.

Several research methods were considered to answer the two research questions posed by this thesis. These questions, and the methods proposed to answer them, are described below.

2.1 Question One

- **Does the proposed extension to the DDDS provide an adequate resource resolution system for the URN?**

The first question posed by this research intends to determine the suitability of the DNS to complete URN resolution. Should an extension of the DDDS proposal provide for such URN resolution this question can be answered positively.

2.1.1 Research Method

It is proposed that this question can be addressed through adoption of the experimental research method. Experimental research design seeks to prove or disprove a hypothesis by completing a series of controlled tests. As the hypothesis is concerned with the feasibility of a technical goal the quantitative nature of experimental research design is ideal.

Such tests in this context involve both the adherence of the system devised to a series of design goals and the demonstrated technical feasibility of URN resolution. Due to the nature of this work as a "proof of concept", the experiments will prove or disprove the potential for DNS based URN resolution.

2.1.2 Anticipated Outcomes

While the URN resolver must work for this question to be answered positively, the extent to which resolution is possible is of note. By exploring the requirements of URN resolution, this thesis will be able to develop clear design goals for a resolver to reach. The ability to achieve these goals will determine if this hypothesis is proved or disproved.

2.2 Question Two

- **Can the URN resolver developed be used to resolve resources for a URN namespace?**

While question one should prove the technical feasibility of this work, question two seeks to examine its practical application. This question is concerned with whether or not the resolver can be used for a particular namespace and if so, how it would be used.

2.2.1 Research Method

In answering the first research question, this thesis seeks to develop a URN resolver which adheres to a set of design goals. Whether the first research question is proven is governed by whether the resolver achieves these goals. This approach, conducted through use of the experimental research method, can also be adopted to answer the second research question. The goals however will be concerned with how immediately usable the resolver system is for the various users involved, as distinct from how technically functional it is.

2.2.2 Anticipated Outcomes

Should the development of a URN resolver in question one succeed, it is intended that a URN namespace be registered and a resolution network for URNs within this namespace be developed. The process by which names registered in this namespace can readily be resolved will be assessed in accordance with predetermined design goals.

Chapter 3

Internet Identification

3.1 Overview

The purpose of this thesis is to enable persistent identification through the implementation of a resolver system for the Uniform Resource Name. This Chapter offers an examination of the identification systems presently available on the Internet and justifies the choice of this class of identifier.

3.2 Identifier Characteristics

It is widely understood that the technical feasibility of persistent identification is but one of the challenges faced in what is largely a managerial issue. Thus, whilst it is important to have an available means of resolving an appropriate identification scheme, it must first be established that the scheme in question actively promotes the concept of persistent naming. It is then relevant to consider the properties of identifiers which promote long term availability, or persistence.

A general enumeration of desirable characteristics is proposed by (Falstrom and Huston, 2004) comprising uniqueness, consistency, persistence, trust, robustness, withholding, referential consistency and structure. These characteristics are detailed below:

- Uniqueness: identifiers which are not re-used and which are only used to refer to one object.

- Consistency: ensuring the same interpretation of the identifier within a particular address space.

- Persistence: in this context, an identifier which remains constant for a period of time as well as remaining accessible.

- Trust: a form of assurance to users that the identity requested is issued by a valid entity.

- Robustness: the ability for an identifier scheme to resist various security threats posed.

- Withholding: an identifier should only reveal those parts of its structure relevant to the operation being performed.

- Referential Consistency: the goal of consistent interpretation of identifiers when either the resource being represented or the resolution service employed changes.

- Structure: the provision within the identifier for a hierarchy of resolution, thereby ensuring efficient interpretation and resolution.

3.3 The Uniform Resource Identifier

At a user level, all resource identification on the Internet is achieved through use of a Uniform Resource Identifier (URI). The term URI refers to a class of Internet identifiers specified in (Berners-Lee, 1998) and developed primarily through the work of the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

Oddly, there are several views as to what constitutes a URI. The confusion is such that a paper was devised from the URI interest group to clarify the situation. The paper states: "Web-identifier schemes are, in general, URI schemes, as a given URI scheme may define subspaces" (Mealling, 2002). This definition asserts that all web identifier schemes are URI schemes with the distinction based on the characteristics of the individual identifier.

For example,the identifiers "`http://www.example.com/`", "`mailto://jim@jim.net`" and "`ftp://test:123@testing.com`" are all instances of the URI scheme known as the Uniform Resource Locator (URL). They do, however, use different name spaces as defined by their name space identifiers ("http:" the Hypertext Transfer Protocol (HTTP), "ftp:" the File Transfer Protocol (FTP) and "mailto:" for email addresses). This concept is further clarified in Figure 3.1.

The URI container describes two important subspaces - the Uniform Resource Locator (URL) designed to facilitate resource retrieval, and the Uniform Resource Name (URN) designed for naming of resources. The URI standard also provides for meta data storage in the form of the Uniform Resource Catalog (URC). To date, the URL scheme has achieved almost universal use throughout the Internet with very limited implementation of the URN or URC standards. The URC locations referred to in this document imply a URL which references metadata information.

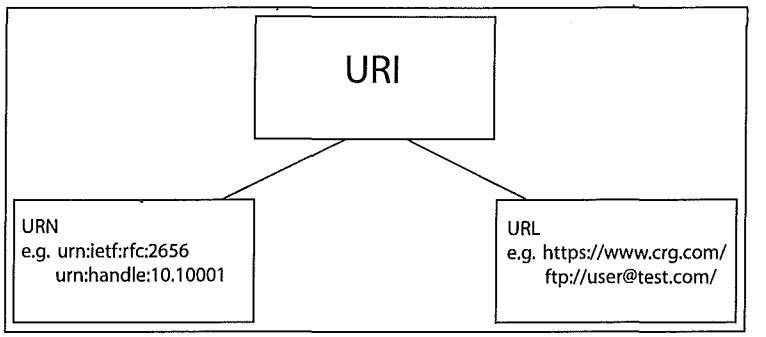

Figure 3.1: The Uniform Resource Identifier

3.3.1 The Uniform Resource Locator

The URL is a URI scheme which describes resources in terms of their location. The W3C (Connolly and Berners-Lee, 1993) states that their purpose is to "reduce the tedium of 'log in to this server, then issue this magic command ...' down to a single click". The URL provides a means of encapsulating all the instructions required to retrieve a resource into a human readable string. A URL can therefore be thought of as an algorithm for resource access. This is clearly evident when inspecting the syntax of the URL as shown in Figure 3.2.

```
⟨NID⟩://⟨domainname⟩:⟨port⟩/⟨directorypath⟩/⟨filename⟩⟨arguments⟩
```

Figure 3.2: Syntax of the Uniform Resource Locator

Where:

- NID: a name space identifier (e.g.: http, ftp)
- Domain Name: an Internet domain name (e.g.: www.foo.com)
- Port: an optional destination port number.
- Directory/Filename: a directory path to the resource.

The "namespace identifier" (NID) describes the type of service which, when queried, will return a resource result. Common NID examples include "http://" for hypertext documents, "mailto://" for email addresses and "ftp://" for FTP addresses. Applications use namespace identifiers to ascertain which protocol and port number to use in communicating with destination servers.

The NID "http", for example, informs applications to attempt to communicate using HTTP on destination port 80 - the default for this namespace. The mapping of ports to the services identified in the NID is maintained by the Internet Assigned Names Authority (IANA). Operating systems which provide Internet access have their own mechanisms for translating service abbreviations to port numbers. In UNIX this is accomplished by querying the text file "/etc/services".

The "domain name field" represents an Internet domain name from which a resource can be retrieved. Domain names are used to provide human readable representations of IP addresses. Translation of domain names into the IP addresses they represent occurs through querying the DNS. The DNS is a hierarchical resolution system available as a function of most modern operating systems.

Port numbers can be optionally appended to domain names to specify the destination communication port to query. This option is frequently exercised to override the default port for a particular namespace, or to differentiate between several instances of a particular service on the same host.

The "directory/filename" field of a URL represents the actual location of a resource upon the server identified in the URL. In some cases, arguments can be passed to the filename to return part of a resource or to specify parameters to a program identified by a URL.

Locating a resource on the Internet requires the interaction of all fields in a

URL. Each of the URL namespaces in use define specific means of interpreting and processing arguments.

An example of the process of resource retrieval using URLs can be illustrated with the URL "`ftp://luke:test1@example.com/Data_Folder/test.dmg`". When this URL is entered into a web browser, several steps are completed before the resource is returned to the user.

1. The domain name - "example.com" is extracted from the URL and, through querying the DNS, translated into the IP address 130.59.23.221.
2. Given the specification of the "ftp" namespace, the host 130.59.23.221 is queried using the FTP protocol on port 21.
3. Assuming a connection to the host is established, the login name "luke" and the password "test1" is sent to the host.
4. Assuming a successful login, the host is then queried for the resource "test.dmg" stored in the directory "Data_Folder".
5. The resource identified "test.dmg" is returned to the user from the FTP service on the host "example.com".

The persistence of an identifier can be expressed as its resistance to change. The URL - an extremely extensible and compact means of expressing the location of a resource, is highly susceptible to change in several respects.

Our previous example URL, "`ftp://luke:test1@example.com/Data_Folder/test.dmg`", identifies an object named "test" located on a sever "example.com". Every component in this URL could be reasonably expected to change.

The use of the NID "ftp" states that presently this resource is stored on a server accessible using the FTP. FTP is currently the most popular means of large-scale data transfer on the Internet. However, it is quite simplistic in its security

and transfer mechanisms. New approaches to file transfer, with corresponding new URL namespaces, are highly likely to emerge. Should the administrator of this resource choose to adopt such new technology, the identifier for this resource would change.

Likewise, the user and password field - essential to resource access in this instance - are highly susceptible to change given any variation in security policy or any addition or removal of users on the server.

Although domain names are themselves abstractions to IP addresses which can change freely, they are by nature of their hierarchy exposed to change in structure outside of their control. Users of the "example.com" repository may, for example, have administrative control over the "example" domain. However, the administration of the "com" domain is outside of their control and, should the policies of this registrar change, so in turn will the domain name and therefore the identifier.

Changes to the directory and filename structure are extremely common. Considered design of directory structure can control such change, however it cannot prevent it altogether. Filenames change for any number of reasons, most commonly due to changes in the resource type or changes in the directory structure on the server hosting the resource. The filename "test.dmg" for example identifies a disk image file. Several storage options are available for such resources. File extensions such as ".iso" for mountable disk images and ".tgz" for compressed UNIX archives could be appropriate. Should the curators of the "test" resource choose to adopt a new storage technology, the filename, and therefore the identifier, would change.

Should any of the above changes occur the URL used to identify the "test" object would change. Given such change, the URL cannot be readily considered as an option for persistent identification on the Internet.

3.3.2 The Uniform Resource Name

Given the brittle nature of the URL and the increasing need for reliable resource access the Uniform Resource Name (URN) was developed, "intended to serve as a persistent, location-independent, resource identifier" (Moats, 1997). URNs provide unique names for resources which can be resolved into the location of information about a resource. Such information can include resource metadata and the location of the resource itself. A URN assumes the form shown in Figure 3.3.

```
urn:⟨NID⟩:⟨NSS⟩
```

Figure 3.3: Uniform Resource Name Syntax

Where:

- NID: is a namespace identifier (e.g.: "ietf")
- NSS: is a namespace specific string (e.g.: "rfc:2404")

The URN namespace identifier (NID) refers to a URN name space which has been assigned by the IANA. Several restrictions are imposed on the structure of the NID in the URN syntax standard (Moats, 1997). This document states that the NID must be a case insensitive string comprised of alphanumeric characters. Furthermore the NID "urn" is reserved and excluded from use.

The Namespace Specific String (NSS) serves as the actual resource name and can include any hierarchy deemed suitable by the NID authority. It is limited to alphanumeric characters with some select additional characters. Overall, it is the suggestion of Sollins and Masinter, (1994) that URNs be kept "short, use a minimum of special characters and be case insensitive" to aid in human transcription.

Several requirements are outlined for NIDs before they can be incorporated as a URN scheme. These requirements, outlined in (Sollins and Masinter, 1994), define both the functional and presentation requirements for NIDs and are listed below:

- Global scope: a URN is a name with global scope which does not imply a location. It has the same meaning everywhere.
- Global uniqueness: the same URN will never be assigned to two different resources.
- Persistence: it is intended that the lifetime of a URN be permanent. That is, the URN will be globally unique forever, and may well be used as a reference to a resource well beyond the lifetime of the resource it identifies or of any naming authority involved in the assignment of its name.
- Scalability: URNs can be assigned to any resource that might conceivably be available on the network, for hundreds of years.
- Legacy support: the scheme must permit the support of existing legacy naming systems, insofar as they satisfy the other requirements described here. For example, ISBN numbers, ISO public identifiers, and UPC product codes seem to satisfy the functional requirements, and allow an embedding that satisfies the syntactic requirements described here.
- Extensibility: any scheme for URNs must permit future extensions to the scheme.
- Independence: it is solely the responsibility of a name issuing authority to determine the conditions under which it will issue a name.
- Resolution: a URN will not impede resolution (translation into a URL, q.v.). To be more specific, for URNs that have corresponding URLs, there must be some feasible mechanism to translate a URN to a URL.

- Single encoding: the encoding for presentation for people in clear text, electronic mail and the like is the same as the encoding in other transmissions.
- Simple comparison: a comparison algorithm for URNs is simple, local, and deterministic. That is, there is a single algorithm for comparing two URNs that does not require contacting any external server, is well specified and simple.
- Human transcribability: for URNs to be easily transcribable by humans without error, they should be short, use a minimum of special characters, and be case insensitive. (There is no strong requirement that it be easy for a human to generate or interpret a URN; explicit human-accessible semantics of the names is not a requirement.) For this reason, URN comparison is insensitive to case, and probably white space and some punctuation marks.
- Transport friendliness: a URN can be transported unmodified in the common Internet protocols, such as TCP, SMTP, FTP, Telnet, etc.
- Machine consumption: a URN can be parsed by a computer.
- Text recognition: the encoding of a URN should enhance the ability to find and parse URNs in free text.

Whereas URL NIDs differentiate the various technical mechanisms for locating resources, URN NIDs provide a means of classifying resource types. Such classification can be derived from various means including the type of resource being named or the organisation responsible for its curation.

Once a group registers a URN namespace they are able to dictate various technical specifications, such as resolution, according to their needs. They are also

free to dictate the functionality provided by the NSS and the syntax used to describe this functionality. The NSS adheres to the needs of the resources being named as dictated by those best informed of what those needs are.

An example of a URN NID is the "ISBN" URN namespace for International Standard Book Numbers (ISBNs) (Hakala and Walravens, 2001), a numbering system intended to provide a unique means of identifying books. The NSS for this purpose takes an ISBN such as "188098508X" and separates the various components which comprise an ISBN. This forms the URN "`URN:ISBN:1-880-98508-X`".

ISBN URNs could be used for various purposes. Take for example the URL "`http://www.amazon.com/exec/obidos/tg/detail/-/188098508X`", which represents the location of information required to purchase a book. The various means in which a URL can change have been explored previously. By replacing this URL with a URN such as "`URN:ISBN:1-880-98508-X`" we provide flexibility for such change through abstraction of locating a resource from naming a resource.

It would be naive to seek a means of controlling the various manners in which resource locations, and therefore the URLs which describe these locations, can change. The URN, through abstraction of the location of a resource, provides means for a URL to change freely leaving the URN persistent.

While the URN provides for persistent naming, its usability relies on an ability to resolve URNs into the URLs they represent. The widespread adoption of URNs is being hindered by the lack of such resolution technology.

3.4 Summary

This Chapter outlined requirements for persistent identifiers and introduced the URI. The two URI subspaces used for resource location and naming, URL and URN, were also introduced.

Through examination of the structure and purpose of the URL and the URN, it has been concluded that the the URN identifier is suitable for persistent identification. A means of resolution is required, however, before URNs can be considered for widespread implementation.

Given this conclusion, Chapter 4 examines the requirements for URN resolvers and details the current URN systems available.

Chapter 4

Uniform Resource Name Implementations

4.1 Overview

Given the choice of the URN as a suitable means of persistent identification in Chapter 3, Chapter 4 outlines the requirements of an effective URN resolution system. A list of desirable URN resolver requirements is presented and these requirements form the basis upon which the presently available means of URN implementation are assessed.

4.2 URN Resolution Requirements

The maturity of the URN standard, and the imminent need for its implementation, would be reasonably expected to result in an available means of resolution. Unfortunately, despite several proposals, this is not the case.

Several challenges are present in URN resolution most notably concerning the flexibility by which the names can be resolved by different users with vastly different requirements. Whereas URL namespaces propose a technical goal, achievable when the community agrees upon a standard command syntax,

URN namespaces are designed to provide identification that meets the various needs of a particular group.

URN resolvers exist to provide a means of resolving the name of a resource (URN) to its location (URL) or its description (URC). This resolution functionality has largely been discussed in terms of a set of "hints" or "rules" which are interpreted by the resolver. Rules provide a means for interpreting URNs without consideration of location by matching all or part of a URN and returning a location which can provide further information on the URN.

URN resolvers can only function if the URNs they resolve abide by the syntax prescribed by both the URN specification (Sollins and Masinter, 1994) and their own name space definition. Whilst there exists potential for a resolver to check for adherence this is largely within the responsibility of publisher and administrators. Of paramount importance is the requirement of URN uniqueness - a feature completely necessary for reliable resolution.

Fortunately, for those seeking to implement URN systems, resolution is a topic much discussed. As several documents exist outlining community consensus on requirements for this procedure. The overall requirements for URN resolution are summarized in the three headings offered by Sollins, (1998) "Usability, Security and Evolution". The requirements outlined in this document are described below from the perspective of the different groups who will be using the system: the clients resolving URNs, the publishers distributing URNs and the administrators responsible for the name server infrastructure.

4.2.1 Usability

The first requirement for URN resolution comprises those considerations which affect the ease with which users can resolve URNs, publishers can submit URNs and administrators can manage URNs. It is the most important requirement in the immediate sense as without usability sufficient to encourage adoption none of the other requirements bear consideration. The primary goal of ensuring a usable URN system is to promote persistence through simplicity. The requirements Sollins, (1998) state "it is not sufficient for a URN resolution system merely to make it possible for URNs to have long lifespans", insisting that URN resolution systems should actively encourage persistence through their design.

Client

From a user or client perspective usability is judged by the simplicity and speed by which URN queries can be resolved into a corresponding resource locator. Furthermore, requirements state that users should be armed with enough functionality to "specify preferences and priorities" Sollins, (1998). This functionality should be mutable in the instance that users wish to leave such selection up to the resolver. Overall it is the performance that users will notice first. As such, the process of URN resolution - only the first in potentially several steps in resource delivery - should be as fast as possible.

The exact requirements are reproduced from (Sollins, 1998) below:

- The interface to the resolver must be simple, effective, and efficient
- The client and client applications must be able to understand the information stored in and provided by the resolver easily, in order to be able to make informed choices.

Publisher

Publishers are, in most cases, unlikely to be computer scientists, more often curators. More often, they will be curators, research scientists and individuals. As such URN resolution systems should be simple enough to provide these groups with means to assign and distribute resource names. The names allocated should be verifiable, easily published and once installed, "correctly and efficiently resolvable by potential clients" Sollins, (1998). Given the vastly different requirements of the groups using URNs, and the various URN solutions available, it is essential that publishers are able to choose between resolvers and - should the need arise - change resolvers in a relatively simple fashion.

The exact requirements are reproduced from (Sollins, 1998) below:

- URN to Hint Resolution must be correct and efficient.
- Publishers must be able to select and move among resolver services to locate their resources.
- Publishers must be able to arrange for multiple access points for their location information.
- Publishers should be able to provide hints with varying lifetimes.
- It must be relatively easy for publishers to specify to the management and observe their hint information as well as any security constraints they need for their hints.

Administration

URN administrators are likely to be those people responsible for the current DNS infrastructure that enables the use of the URL. As such, the use of URNs

should pose as few new constraints upon their resources as possible. These constraints extend to the simple insertion and management of hint information, realistic network overheads and the flexibility to manage URN use.

The exact requirements are reproduced from (Sollins, 1998) below:

- The management of hints must be as unobtrusive as possible, avoiding using too many network resources.
- The management of hints must allow for administrative controls that encourage certain sorts of behavior deemed necessary to meet other requirements.
- The configuration and verification of configuration of individual resolver servers must be simple enough not to discourage configuration and verification.

4.2.2 Security

The issue of security in the scope of URN resolution is almost entirely within the responsibility of the administrators. Several threats are posed to naming systems. The three most notable cited in (Sollins, 1998) are unauthorised insertion of records, unauthorised replication of databases to servers masquerading as slaves and the potential for denial of service (DoS) style attacks. These threats are addressed further in (Sollins, 1998) in the form of three security goals: Access Control, Server Authenticity and Server Availability.

The three security goals are essentially quite simple and common to most computer systems, though the application to this system is still important. Access control, enacted upon the database of URN mappings, implies both a single "authoritative" version of the rule set and a reliable means of privilege control to this authoritative server. Authenticity demands a means of ensuring that the slave servers which request updates are in fact the servers to which authorisation has been given to act as slaves. Through replication of authoritative servers the potential for denial of service style attacks, can be isolated by providing several redundant instances of the database.

The privacy of those using URN systems is an important consideration. Usage scenarios exist where the requests sent to URN servers should not be world viewable. Furthermore, publishers and administrators of URN information may wish to prevent access to all of the resolution information they provide. In the case of the URL, such privacy is largely handled as a function of resource delivery, not resolution. Developments upon such privacy will pose a large problem for URN implementors.

The exact requirements are reproduced from (Sollins, 1998) below:

- It must be possible to create authoritative versions of a hint with access-

to-modification privileges controlled.

- It must be possible to determine the identity of servers or avoid contact with unauthenticated servers.
- It must be possible to reduce the threat of denial of service by broad distribution of information across servers.
- It must be possible within the bounds of organizational policy criteria to provide at least some degree of privacy for traffic.
- It must be possible for publishers to keep private certain information such as an overall picture of the resources they are publishing and the identity of their clients.
- It must be possible for publishers to be able to restrict access to the resolution of the URNs for the resources they publish, if they wish.

4.2.3 Evolution

Any URN resolution system derived from current requirements needs to be flexible enough to change when new technologies present themselves or when other requirements change. Though it is not possible to predict the future, we can build some reasoned assumptions into our software to enable change. Such reasoned assumptions, in the context of URNs, are primarily concerned with changes in the syntax, new resource endpoints and threats to security. Adapting to changes in the syntax and interpretation of URNs is an evolutionary goal which will be immediately relevant. Each name space adopting URNs has their own syntactical requirements and as such, the resolver needs to be generic enough to cope with varied interpretations of syntax and further changes to this interpretation.

Changes in the resources named by URNs are highly predictable. Whilst at present the popular means of resource location is the URL there is no guarantee that in the future such locator's will be relevant. Similarly, there needs to be extensibility within the resolver to handle queries for metadata and other resource pointers.

Evolutionary considerations with regard to users of URN resolvers are important, but it is also relevant to consider the trend of continuing sophisticated security threats to computer systems. It is unwise to assume that such threats against URN resolvers, will never transpire, and as such, patches and updates to fix security problems need to be guaranteed.

The exact requirements are reproduced from (Sollins, 1998) below:

- A resolver must be able to support scaling in at least three dimensions: the number of resources for which URNs will be required, the number of publishers and users of those resources, and the complexity of the dele-

gation, as authority for resolution grows and possibly reflects delegation in naming authority.

- A hint resolution environment must support evolution of mechanisms - specifically for a growing set of URN schemes, new kinds of local URN resolver services, new authentication schemes and alternative resolver schemes acting simultaneously.
- A resolver must allow the development and deployment of administrative control mechanisms to manage human behavior with respect to limited resources.

4.3 URN Schemes

The following section presents four of the approaches to persistent naming. Each of these approaches specifies syntax and resolution semantics which enables users to deploy identifiers with some or all of the URN characteristics. Specification of syntax and resolution processes can, however, restrict users and does not allow for the diversity of requirements that is likely.

As most of the requirements outlined in (Sollins, 1998) are not addressed by these systems, they are presented in terms of their history, syntax and resolution. Some consideration is also given to their adherence to the evaluation criteria for URN resolvers.

4.3.1 Handle

Developed by the US based Corporation for National Research Initiatives (CNRI), the handle system provides "an efficient, extensible and secured global name service for use on networks such as the Internet". The Handle system has been implemented by several groups such as Hewlett Packard (HP) and the massachusetts Institute of Technology (MIT), with their DSpace repository project (Hewlett Packard, 2003), the International Digital Object Identifier group (IDF), with their Digital Object Identifier (DOI) system (Paskin, 2004) and various American defense agencies working on the Defense Virtual Library (DVL) (Markheim, 2004).

Major outcomes of the Handle project have been the development of a URN name space for Handle documents and an open protocol for the resolution of these names. By defining their own name space, as all other implementations discussed here have, handle implementers are able to control resolution architecture much more effectively. The Handle system is comprised of multi-

ple local resolvers which, when registered with an appropriate Global Handle Registry (GHR), become part of a global set of unique identifiers.

Syntax

Each "handle" takes the syntactical form shown in Figure 4.1:

```
urn:⟨handle⟩:⟨HandleNamingAuthority⟩/⟨HandleLocalName⟩
```

Figure 4.1: Handle Identifier Syntax

Where:

- Handle Naming Authority: is the unique name of a delegated resolution authority.
- Handle Local Name: is the locally unique resource name being referenced.

There are two important fields in the handle URN - the naming authority and the local name. Naming authority strings are delegated by a GHR to a Local Handle Service (LHS). The LHS assumes responsibility for mappings between handle local names and resource locations within its name space. Delegating resolution authority in such a fashion renders the GHR solely responsible for uniqueness of its delegated name spaces and the LHS solely responsible for uniqueness of the resources in its database.

The assigned naming authority string provides a point of resolver delegation and, as such, is tightly controlled. However, it is left to the local authorities to determine the syntax of the local name. This leaves publishers free to devise their own structures to ensure uniqueness and usability.

Resolution

Resolution of handles into uniform resource locations (URLs) occurs in a four stage process as shown in Figure 4.2.

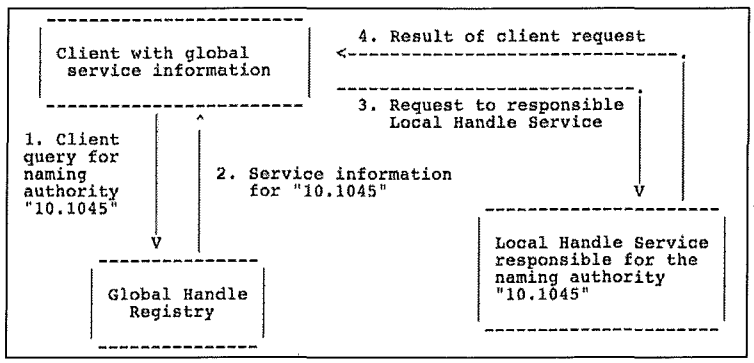

Figure 4.2: Handle resolution procedure (Sun et al, 2003)

The resolution process starts with a client querying the GHS for the LHS responsible for the naming authority the handle cites. Once determined, this information is returned by the GHS to the client with specific service information necessary to contact the LHS. This service information is used to build a query against the LHS which in turn responds with a URL for the resource.

Handle resolvers are presently implemented as a Java service which can run on either a single server per name space or can be distributed across several servers for performance and reliability. Performance can be further enhanced through use of the caching functionality within the handle framework. Handle caching servers can also be used to reduce bandwidth requirements and speed up requests.

As mentioned, the handle service relies on its own resolution system, which means conventional web clients wishing to access handle services will presently need to either have a browser plug in or make use of a proxy server in order to resolve handles to URLs. A further and more pressing problem is the lack of

interoperability between handle servers. Rather than providing a global resolution space, handle resolvers provide resolution only for the resources known to its database.

Evaluation

Handle's resolution framework offers strong security features. However, its evolutionary and usability features are open to criticism.

The Handle System requires new client and server software to be installed on every node seeking to participate in resolution. Although this software appears quite simple to install and configure, many organisations will require strict security and performance validation before installing new software on production servers, thereby limiting the adoption and usability of the system. Furthermore, the choice of the Java language for servers (for portability reasons) leads to serious questions about server performance should the handle server be faced with numerous queries.

The DOI project has shown that the Handle can be implemented to incorporate economic incentives. This project requires payment of registration fees in return for allocation of identifiers and has, to date, been quite successful.

Security concerns in Handle are well addressed with various technologies. The issue of Access Control has been met with per-resource controls on data with a Challenge-Response style authentication system. The issue of privacy has been addressed with optional cryptography of all client and server data interchange.

In terms of evolutionary considerations, Handle does not specifically address issues of change in its design. However, the availability of source code for the platform means that implementers will be able to implement any changes they consider appropriate so long as they have the technical expertise. However,

the issue of URN growth is of some concern. Despite provisions for caching and distribution of resolution it remains to be seen how the Handle server will perform under a heavy load.

4.3.2 Persistent URLs

The Persistent Uniform Resource Locator (PURL) system is a means of embedding persistence into the current URL standard through the maintenance of a database of redirection addresses. This framework relies on the HTTP redirection functionality which has been available since the release of HTTP 1.0 (Fielding et al, 1999). Whilst not technically a URN system, its rate of adoption amongst Digital Library groups warrants consideration of its value.

Syntax

The PURL syntax is the same as that of a URL as shown in Figure 4.3.

```
http://⟨ResolverAddress⟩/⟨directoryfield⟩/⟨resourcename⟩
```

Figure 4.3: Persistent URL Syntax

Where:

- Resolver Address: is an Internet domain name that represents a resolver for this resource.
- Directory Field: represents further hierarchical classification of the resource.
- Resource Name: represents the name of the resource sought.

Whilst the PURL does appear as a standard URL, the domain name field has an important difference as it is used to identify the address of a resolver from which the name specified as a resource location can be redirected into its actual URL. This URL is not at all necessarily linked to the PURL.

Resolution

Lacking a global structure, the PURL system essentially creates identification islands whereby one resolver cannot be used to gain information further up the resolution chain. Within the domains themselves there is some provision for hierarchy as the name field can be used as a delegation point. This provides for two varieties of domains - "top level domains and sub domains" (Shafer et al, NA) - which are differentiated based upon whether they appear in the resolver address field (top level domain) or the directory field (sub domain).

This hierarchy does not provide the benefits which it might should it extend over several resolvers. However, it does provide an administrative hierarchy which aids PURL database administrators greatly when considering user rights management for database updates. The clear delegation of resolvers has resulted in simple web based registration and rights assignment to the resolver itself.

Users can point a browser at a PURL server and apply on line to be a registered administrator for a top level or sub level domain. PURL resolution can also be managed by means of access group or on a per user basis which restricts unregistered users from being able to view pages at all should the access permissions on the pages not be specified public.

Though the PURL system enjoys widespread use, its applications are somewhat limited to institutions that do not require any form of interaction between their resources and the resources resolvable through other PURL servers. Furthermore the isolation of resolvers means the benefits of efficiency and interoperability that distributed resolution systems such as the Handle system and the DNS enjoy are not available.

Evaluation

The PURL system is not technically a URN resolution system and does not warrant evaluation as such. However, the approaches taken to usability by this project are valuable.

PURLs are manageable through a series of web interfaces which are accessible when a user directs their browser at a PURL resolver. If the user is an authorised administrator, all mappings of PURL to resource can be managed on line. These mappings are stored on the PURL server and form the basis of HTTP redirects which are executed when a user requests the given PURL. This simplicity of administration and deployment is the PURLs strongest feature.

4.3.3 Archival Resource Key

Another proposed solution to the problem of persistent identification is the Archival Resource Key (ARK) devised by John Kunze from the University of California for first release in 1992. The ARK system emphasizes several neglected points with regard to persistent identifiers - most importantly, "persistent actionable identifiers, where an actionable identifier is one that widely available tools such as web browsers can use" (Kunze, 2003). Its specification further stresses the "importance of the association between a string and an information object" (Kunze, 2003).

Syntax

The ARK syntax is designed to be encapsulated within a standard URL, however its structure expands well beyond and, similarly to Handle, is purposely resemblant of a URN identifier as shown in Figure 4.4. The purpose of this design is to enable users to extract that ARK component of a URL string to ensure ongoing usefulness "when the web no longer exists" (Kunze, 2003).

```
http://⟨NMAH⟩/ark:/⟨NAAN⟩/⟨ResourceName⟩
```

Figure 4.4: Archival Resource Key Syntax

Where:

- NMAH: is the Name Mapping Authority Host port.
- ARK: is the ARK label.
- NAAN: is the Name Assigning Authority Number.
- Name: is a identifying string issued by the NAAN.

In a URN context this identifier would be changed such that the NMAH, essentially the location of a proxy server for URN resolution, would be "discovered" and therefore not tied into the identifier. The NAAN number is a globally unique assigned number that directs the resolver to the organisations that originally assigned the Name to the object in question. NAANs are numbers in the form of 5 or 9 digits. This scheme will allow for up to a 100,000 NAANs in 5 digit form and up to a billion in 9 digit form.The name component is a NAAN-assigned alphanumeric string which can also include six characters ("=", "@", "$", "_", "*", "+", "#"). The "%" character is used to present encoded representations of characters not in the allowed list. Object hierarchy in naming is permissible through the inclusion of further /'s in the naming of the ARK.

Resolution

ARKs can be resolved into either a resource location, a location for meta data information or a statement outlining the guarantees of persistence the identifier implies.

In its present form ARK requires the use of proxy resolution services and the NMAH points to the location of such a service. It is anticipated that this will not be the case for long as means by which to resolve URNs become available. ARKs identified by the NMAH are global - that is "ARKs that differ only in the optional NMAH part identify the same object" (Kunze, 2003). This is similar in functionality to the Handle system and an important difference when compared to the PURL system as there is a implicit guarantee of uniqueness.

Once resolvers are located however the local resolution of ARKs is conducted through a four step process very similar to the Handle process outlined above. The technical implementation of this process is unclear from the initial pro-

posal documents. However, suggestions exist for protocol development specifically for this purpose.

Evaluation

ARKs have potential for success given the strong importance placed upon their persistence throughout their design. However, they have yet to address security in their specification and some questions remain about their usability.

Using the ARK system involves editing flat-file databases of resource to identifier mappings. Although these can be laid out in a simple format, the issues of access and editing procedures need to be addressed. Furthermore, the issue of economic incentives has not been introduced or suggested in the ARK context.

According to the ARK specification, the system is designed with evolution in mind - "ARK mechanisms are first defined in high level, protocol independent terms so that mechanisms may evolve and be replaced over time without compromising fundamental service objectives" (Kunze, 2003). This philosophy is present throughout the documentation as the technologies involved in the implementation of the ARK system are described in general terms with their purpose clearly defined.

4.3.4 Life Sciences Identifier

The Life Sciences Identifier (LSID) is a namespace of URN identifiers which "are persistent, location-independent resource identifiers for uniquely naming biologically significant resources including but not limited to individual genes or proteins, or data objects that encode information about them". Developed in cooperation with International Business Machines (IBM) the LSID scheme has a well developed means of resolution, making use of emerging Web Services technology for describing and resolving resources.

Syntax

The LSID has a well defined identifier structure which aids in its efficient resolution. This structure is displayed in Figure 4.5.

```
urn:lsid:⟨AuthorityID⟩:⟨NamespaceID⟩:⟨ObjectID⟩(:⟨RevisionID⟩)
```

Figure 4.5: Life Science Identifier Syntax

Where:

- AuthorityID: is an Internet domain name representing the URN owner and resolver.
- NamespaceID: represents the collection the identifier belongs to.
- ObjectID: is an uniquely assigned number for a resource inside this collection.
- Revision ID: is an optional version iterator.

Being a domain specific namespace there is little to be gained from an examination of this syntax. Of notable importance, however, is the authority ID

field, an Internet domain name representing an authoritative server for this resource. While dynamic resolution is discussed in the LSID proposal, this means of retrieval presents a "shortcut" which effectively rears the head of location dependence.

Resolution

Resolution in the LFID system is achieved through use of Web Services technology. Simply put, Web Services provide an application interface for the World Wide Web (WWW). Through use of the WWW, communication between applications and the platform neutral standards used throughout is greatly simplified, making Web Services a very useful distributed systems technology.

The first step in the resolution of an LFID is the discovery of a resolver for the identifier cited. In most cases, the resolver is expressed as an Internet domain name in the AuthorityID field. Work has been completed on providing a location independent means of resolver discovery using the Dynamic Delegation Discovery System (DDDS). However, the use of this system is optional for LFID at present. The DDDS is discussed at length in Chapter 5.

Once a suitable resolver has been found, the client sends a Simple Object Access Protocol (SOAP) message to the authoritative resolver asking for the available services for this resource. This process is completed using the "getAvailableServices()" method and, if successful, will return a list of services available along with the protocols required to access them.

Finally the user is able to query any of the services available using the "getData()" method.

Evaluation

LSID systems are targeted toward a very specific subset of the scientific community - consequently, assessment of their usability has to take into consideration the types of users that can be expected. In light of this the simple method calls required to retrieve LSID resources are quite reasonable. Furthermore, the various client utilities developed for the purpose of LSID retrieval provide an alternative and highly user friendly means of resource access.

In terms of evolutionary considerations the extensive use of Web Services technologies can be seen as both a strength and a potential weakness. On one hand, the method implementation for resource retrieval is abstracted from the user and therefore free to change almost completely. However, the use of Web Service protocols such as SOAP leaves the resolver open to failure as standards evolve.

Though largely ignored throughout the specifications security issues in the LSID system could be comprehensively addressed through use of the various standards present in the field. Such standards could be extended to provide privacy to users and publishers. However, it remains to see how this will be achieved.

4.4 Summary

This Chapter introduced four systems which aim to solve the problem of persistent identification through URN deployment. Each system was found to be deficient in its means of resolution due to the inability for actionable identification without proxy resolution.

Chapter 5 introduces the DDDS - a globally actionable means of URN resolver discovery which could potentially remove the need for proxy resolution.

Chapter 5

The Dynamic Delegation Discovery System

5.1 Overview

Chapter 4 outlined the various URN implementations available presently. This Chapter explores the Dynamic Delegation Discovery System (DDDS), a proposed mechanism for the location of URN resolvers. This system is described and assessed according to the requirements of usability, security and evolution introduced in Chapter 3.

5.2 Design

Perhaps the most significant move toward wide scale URN resolution has been the DDDS, developed as a generic application to "implement lazy bindings of strings to data, in order to support dynamically configured delegation systems" (Mealling, (2002). The DDDS is outlined in a set of five "request for comment" papers and although it remains generic enough for a variety of potential applications it provides a potential means for URN resolver discovery.

Unlike the URN systems discussed previously, the DDDS does not specify syn-

tax for identifiers, nor does it actually provide functionality for resource resolution. Instead the DDDS aims to partially solve the URN resolution problem by accepting a URN string input and traversing a database of "rules" toward the goal of locating a resolver server which is able to further dissect the URN into a resource location. This process is known as resolver discovery. By resolving a URN into the location of a responsible server the DDDS achieves the valuable goal of connecting the disparate URN resolution schemes available into an interoperable network of identifiers. This provides implementors with the flexibility to develop their own name semantics and their own algorithms for URN resolution.

5.2.1 Rules

The DDDS concept is based upon the notion of a "rule" which is the product of querying a database given a certain "key". The key required is itself the product of processing an "Application Unique String", a string input by the user, against a "First Well Known Rule", a default rule which is specific to the particular DDDS application at hand. It is common for one query to return several rules for the given key. There are six fields which comprise a rule; order, preference, service, flag, regular expression and replacement.

The order and preference components dictate the schedule of processing for the numerous rules returned. The order field is usually sufficient for this task. In the case that several records have been assigned the same order the preference field is consulted to determine which rule is to be processed first. This flexibility can be used in several situations - for example, when distinguishing between the different resolution services offered on the one destination server.

The service field is used to ascertain which of the prescribed application services available are sought. The services available will differ depending on the

DDDS application. Likely examples in the URN context include "return URLs for given URN" and "return metadata for given URN".

The flag component of a rule is responsible for "steering" the application according to the types of rules it is receiving. Flags are responsible for declaring whether the rule processed is to be deemed "terminal" - in which case the application returns the processed key back to the user. Flags also tell the application what type of information has been stored in the key.

The regular expression field stores a POSIX regular expression rule, used to describe or match a string. When regular expression rules find a match they can replace the matching string component with a replacement - as present in the replacement field of the DDDS rule. These fields are used in conjunction to test if the application unique string is of the format described by the rule and, to produce a new string in the case of a match. The string produced forms the new key either used in the next lookup procedure or returned to the user.

5.2.2 Resolution Process

The exact manner in which rules are processed is documented in the DDDS specification (Mealling, 2002) and can be summarised in the following steps:

1. The first well known key is applied to the application specific string to produce a key.
2. The database identified by the key is queried for an ordered set of rules.
3. The regular expression in each rule is processed in order until a non-empty string is produced.
4. The service field is checked against user requirements, return to step 3 if incompatible.

5. If terminal flags are present return key to the user. Otherwise, return to step 2 with new key.

6. When a terminal lookup is found, return the key, services and flags to the user.

This process is shown in Figure 5.1

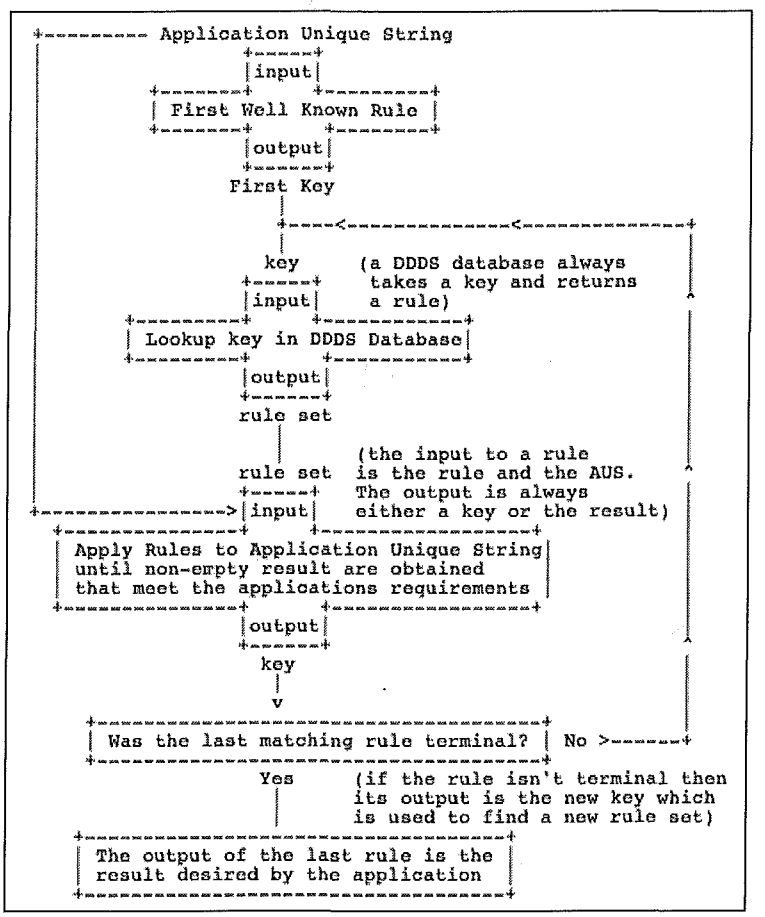

Figure 5.1: DDDS resolution process. (Mealling, 2002)

In applying the DDDS algorithm to the resolution of the URN we are required to determine the values relevant to the fields comprising a rule. Consideration, must also be given toward the exact semantics of processing rules.

The Application Unique String, the URN specified by the user, will be governed by the syntax conventions outlined in the URN specification (Moats, 1997). These requirements state that a URN should assume the basic structure of a "urn:" prefix, a Namespace Identifier (NID) and a Namespace Specific String (NSS). The further restrictions based on case and allowable characters stated in this document provide a well defined URN specification and, therefore, a well defined Application Unique String.

The first well known rule, in the URN case, will be responsible for ensuring the input provided is a valid URN and returning a key which represents the root server responsible for URN resolution. The regular expression required to validate a URN as conforming to the specifications outlined in (Moats, 1997) is represented in Figure 5.2. This rule matches the URN prefix and the namespace identifier syntax, as well as the explicitly disallowed characters. It does not further restrict the namespace specific string, as this is variable on a namespace basis.

```
[uU][rR][nN]:[.[^\"&<>[]\^`{\|}~]]+:[.[^\"&<>[]\^`{\|}~]]+
```

Figure 5.2: First well known rule for URN validation

The issue of services in the URN context is open to interpretation at present. A document exists outlining service specifications (Mealling, 1999), however the services outlined are for resolution of URIs (i.e. URLs or URNs of any form) and make no consideration of the functionality offered by the DDDS. A subset of these services can be considered relevant - those being the services which provide resolution of URNs to URIs and URNs to metadata information. In the DDDS implementation presented these services are represented as "N2L" and "N2C". As the DDDS is not actually a complete resolution system, no immediate consideration of the mechanics of these services is required.

Finally, the flags component of the rules will need to inform the application when to leave the query process and, furthermore, what action to take upon completion. The four flags specified by (Mealling, 1999), "S", "A", "U", and "P" are case insensitive and mutually exclusive. "S" informs the application that the key returned provides a location of service records, "A" denotes a DNS A record has been returned, "U" denotes a URI for a resolver service and "P" states that the rest of the algorithm is application specific and should be handled outside the DDDS algorithm.

Though not dictated by the specifications, an obvious and practical choice for the database which provides rule storage in the DDDS is the DNS. The DNS is practically universally available on modern operating systems and is therefore well placed to facilitate widespread URN adoption. In order to represent the DDDS rule structure in the DNS, a new Resource Record (RR) was devised, known as the Name Authority Pointer (NAPTR). This record provides for storage of all six rule fields in the standard manner by which other Internet resources are represented. An example NAPTR record is shown in Figure 5.3.

```
www.foo.com.
;;        order pref flags   service   regexp      replacement
 IN NAPTR 100   100  "s"    "http+I2R"   ""      _http._tcp.foo.com.
 IN NAPTR 100   100  "s"    "ftp+I2R"    ""      _ftp._tcp.foo.com.
```

Figure 5.3: NAPTR format

The NAPTR record represents the rule structure previously discussed. Its inclusion in the DNS specification allows theoretical access to the caching and security functionality that is critical within the context of a distributed resolution network.

The DNS currently does not provide a simple method for inserting rules into the rule database. Several graphical user interfaces and web portals exist for this purpose, however the common means of inserting rules is editing config-

uration files directly.

5.3 Evaluation

5.3.1 Usability

The DDDS offers several extensions to the usability of current URN systems and closely adheres to the requirements suggested in (Sollins, 1998).

Users

From the perspective of a user or client, the DDDS provides a simple and efficient method of resolving URNs which requires minimal user interaction. As the DDDS algorithm and rule storage mimics that of the URL quite closely, the interfaces with which clients interact should not pose a significant learning curve. The DDDS resolution procedure should pose quite a small time overhead to the user within the larger goal of resource access. It is important to note however that the interactions between the DDDS and the resolver which is discovered may in fact result in increased time to resource access and increased complexity.

Publishers

The DDDS offers publishers several advantages not present in other resolvers - most notably the option to change between resolver frameworks without issuing new identifiers and freedom regarding the identifiers they issue. The DDDS does not however solve the problem of presenting varying resource "lifetimes" and introduces complexity with the use of regular expression rules. The DDDS also does not suggest an obvious means of publisher managed rules.

Administrators

The largest unsolved problem regarding usability exists within the scope of administering and managing rules. This is simply because of the complexities

posed by the use of regular expressions. Although issue should not require frequent expert assistance, deriving rules to efficiently setup a namespace is a non-trivial task. Where publishers are left with minimal regular expression interaction by the DDDS, managers and administrators will need to be well versed in the creation of rules.

The DDDS offers managers and administrators several concessions. The network traffic overhead posed by the DDDS is quite minimal and the use of the DNS as a rule database should alleviate any new security concerns that may be posed by other URN systems such as Handle, which developed their own network protocols. Finally, the issue of configuring new resolvers should be quite simple given the DNSs ability to create slave servers securely and simply.

5.3.2 Evolution

A primary concern with evolution is managing the ability to scale within a system which is likely to experience growth. Fortunately, the adoption of the URL has reached such colossal figures that many of the problems likely to be faced can be solved by incorporating the same solutions that worked for the URL. Employing the DNS as a database for rule storage effectively solves these problems immediately.

The DNS has several features that ensure its scalability - most notably replication functionality, caching and a well structured name hierarchy. These functions are available without any consideration required on the part of URN administrators. The use of NAPTR preference and order values further ensures that URN managers can control the traffic that reaches both the resolvers it operates and the resources to which they refer.

Along with the rule database, DDDS resolvers themselves need to expect changes in the mechanics of URN resolution. These changes will not necessarily affect the persistence of the resources identified, but may affect their accessibility if DDDS resolvers cannot cope with such changes. The present DDDS algorithm is generic enough to incorporate new services and flags. Major changes will, however, require updates to resolver software. It is quite probable that such changes would extend functionality of resolvers, and that the functionality offered presently will be persistent throughout change.

Finally, the DDDS needs to be able to delegate resolution to new resolver systems as they emerge. This should not pose an immediate problem for the DNS based approach so long as the resolvers devised can be accessed through one of the three access methods previously discussed. The DNS "SRV" record approach to resource access provides perhaps the most evolvable means of delegating resource resolution.

The more general issue of evolving to a need for economic frameworks for URN issuing and maintenance should be quite trivial given the hierarchy of resolution the DNS employs. This hierarchy is already home to an economic model through the current management of Internet domain names by Internet service providers. Extension of this system, or the development of new URN based providers using similar business models, should be relatively simple and will be necessary as demand for resources identified by URNs continues.

5.3.3 Security

The DDDS inherits the security flaws present in the DNS along with the measures made to address these flaws. Privacy in the DDDS, however, is unfortunately non existent - no functionality exists to protect the privacy of users or publishers. Several steps can be considered to improve this situation, although the solutions would require quite substantial improvements to the DNS as a whole.

Access control, as it relates to the hint databases, is reliant on the credentials of the operating system hosting the DNS server. Whilst this does expose the database to potential corruption through the use of an insecure operating system, the DNS database is somewhat protected as long as the administrative accounts on the systems are secure.

Authenticity is a major problem with DNS servers, consequently users have to implicitly trust remote servers. Resolution requests can be intercepted and, if returned in time, answered by malicious hosts to redirect users to false hosts. Similarly, DNS "Cache Poisoning" can occur, involving malicious information being entered into valid DNS servers via cache. These security problems are partially addressed in the DNS Security Extensions (DNSSEC) proposed by the DNSSEC IETF working group (Arends et al, 2005), however no widespread implementation has been achieved as yet.

The threat of denial of service attacks can be addressed through replication of the database both by the means provided in the DNS software and by providing multiple records for each resource. Such multiple records and servers can be further protected by sheer number, geographic location and Internet connectivity as appropriate.

5.4 Summary

The DDDS presents a way to join the currently disparate URN resolution systems into a URN system capable of global resolution. Although the interactions between the DDDS and the resolvers it joins are not as yet defined, the ability to return various resource access methods should ensure these interactions are possible.

While the DDDS does not provide a means to resolve resource locations, it does manage to achieve several of the goals outlined in the URN resolver specifications. The DDDS provides efficient resource access to users with sufficient prospects for evolution in its structure even though it faces several challenges in relation to security and privacy.

Given the adherence to resolver requirements demonstrated in this Chapter, Chapter 6 seeks to outline extensions to the DDDS which would provide for complete resource resolution.

Chapter 6

The Extended Dynamic Delegation Discovery System

6.1 Overview

Chapter 5 introduced the DDDS, a means of discovering resolvers for URNs. This Chapter introduces the Extended DDDS (EDDDS) - a system for the discovery of URN resolvers and the subsequent resolution of URNs into resource locations. This system is described, prototyped and assessed according to the requirements of usability, security and evolution used throughout this thesis.

6.2 Design

The EDDDS presents a three phase approach to resolving URN identifiers: resolver discovery, resource resolution and service execution. While the discovery process is essential to globally actionable URNs, it is possible - depending upon the other URN systems in place - for the resolver to delegate resource resolution and service execution to another system, such as Handle.

Resolver discovery involves locating an authoritative resolver - capable of translating a URN supplied into the location of the resource it identifies. This

phase of the EDDDS is completed in a very similar fashion to that of the DDDS and, should the implementer wish, the EDDDS resolver can be used to point to alternative URN systems, as the DDDS does. The primary difference is the ability for the EDDDS to continue resolution of the URN once an authoritative server has been located.

Resource resolution in the EDDDS involves the translation of a URN into the location of information about the resource it identifies. This information can be its location (URL), metadata about the resource (URC), the location of information to ensure data integrity, or the location of information regarding resource persistence.

The first two phases: discovery and resolution - provide the functionality required of a resolver. These two phases, likely to be implemented as an operating system library, provide application programmers with the ability to build applications which use URNs. The final phase - service execution - dictates how the data the URN refers to is processed and presented to the user and would therefore be part of a resolution application for a specific namespace.

While resolution of the URN is completed before the service execution phase, a resolver application will need to implement the services offered in order to be useful. The service execution phase involves the retrieval and processing of the information stored by a URN. In several cases, this procedure can be quite simple. User requirements can vary greatly depending on namespace, however, and these requirements are best met with variations in service execution. A detailed explanation of the plethora of ways URNs could be processed is clearly outside the scope of this thesis. However, an example of how a namespace could implement services is discussed in Chapter 7.

```
IN NAPTR "<ORDER>" "<PREFERENCE>" "<FLAGS>" "<SERVICES>""<REGEX>""<REPLACE>"
IN NAPTR "100" "10" "RES:FTP" "N2L:audio/mpg""urn:paradisec:AB1:001:A2""ftp://ftp.paradisec.org.au/AB1/001/A2"
```

Figure 6.1: EDDDS NAPTR standard format and example

6.2.1 Rules

EDDDS uses a DNS based database of NAPTR records as rules to guide the resolution process. The rules used by the EDDDS are required to be flexible to provide both partial resolution (resolver discovery) and complete resource resolution (location information).

The overall NAPTR structure as a six field resource record (order, preference, flag, service, rewrite and replacement) has been retained. The NAPTR record appears as shown in Figure 6.1.

The interpretation of the order and preference values will remain the same in the EDDDS. These fields will assume new capabilities in two respects: guiding resolution of replicated resources to cater for demand, and distinguishing between various access methods for identical resources. These capabilities are inherent in the nature of a resource resolver and do not require further understanding of the operation of these fields.

The representation of regular expressions in the EDDDS will remain the same as before - however, their interpretation and capabilities have been extended greatly. Where before simple matching expressions proved adequate resource curators with thousands of identifiers to manage may choose to use more elaborate expressions to reduce the amount of records required for a resource. It will be up to publishers and administrators to provide guidelines for expression use, simple expressions must be encouraged for non-technical users, the power of more complex expressions may be attractive to administrators.

The contents of the flag and service fields have been changed significantly to

provide the extra functionality demanded by a resource resolver.

Flags

The flags field, only returned to the user in the final step of resolution, is responsible for notifying resolvers as to the type of server which has been reached. This extension is essential to inform applications on whether a resolver server has been discovered, or if a resource has already been resolved. The presence of any flags in a rule represents a terminal lookup if the rule matches. This means the resolution process stops querying the database and either returns a discovered server address or a resolved location to the user.

The flag syntax consists of two strings separated by a ":" character. The first string represents server type and is either "DIS" - to represent the resolver discovery phase of resource resolution or "RES" - to inform the client application the entire resolution process has been completed.

While the process of resolver discovery is a matter of locating a server, resolution involves returning a locator to a service. The important distinction is the difference in the communications protocols required between servers and services. Another important distinction exists between the notion of Internet services discussed in the context of the flag NAPTR field, and services as discussed in the context of the URN resolution service represented in the NAPTR.

In the discovery case valid entries for the flag field are "A" to represent a DNS A record, "AAA" to represent the IPv6 variant of the A record, "SRV" to represent a DNS service record and "URL" to represent a Uniform Resource Locator. In the case that the resolver located is actually an EDDDS server capable of resource resolution, the application is returned a "NAPTR" - representing the location of further NAPTR rules. These strings have been changed subtly from the format they took in the DDDS in order to follow, where applicable,

the naming conventions for DNS resource records. The use of these IETF managed conventions will ensure both a consistent interpretation by implementors and the simple addition of new standards as they become available.

In the resolver case, the services being used are represented through their Internet service abbreviations, as maintained by the IANA. These abbreviations define the access methods for the service in question. The abbreviation "HTTP", for example, is maintained by the IANA to represent the Hypertext Transfer Protocol, a service which commonly operates on destination port 80.

Services

With the addition of resolution capabilities the importance of services in the EDDDS will increase dramatically. The service field syntax accepts any of the following URN resolution services: "N2L" for URN to URL resolution, "N2C" for URN to URC resolution, "N2S" for the resolution of data integrity information and "N2P" to resolve a URN into an assertion of the persistence offered by its authoritative resolver. Other service requirements can be reasonably expected to arise and should follow a similar three character structure. Multiple service capability can be asserted by separating service identifiers with "+" characters. A server capable of fulfilling N2S and N2L services, for example, would have the service string "N2L+N2S".

In addition to the new services offered in the EDDDS, users are now able to specify a preferred means of resource delivery where available. These service identifiers vary according to the service specified and, should the user wish, can be muted altogether. Where possible, the descriptions used adhere to relevant standards.

The N2L and N2C services both identify a type of content which is described by the IANA in its Multipart Internet Mail Extensions (MIME) standard. By

using MIME identifiers to describe content being requested, the user and publisher have a clear idea of the resource being published and requested. A service field such as "N2L:audio/mpeg" would be an appropriate means of requesting audio in a mpeg format. A more general service request of "N2L:audio" would simply request the audio resources identified by a given URN.

The N2S service describes a means of asserting digital integrity of the resource identified. There are several means by which implementors may choose to offer this service. The RSA and DSA algorithms provide similar functionality to assert digital identity. Therefore, a service field such as "N2S:RSA" requests an RSA encoded digital signature assertion of the integrity of this resource.

6.2.2 Resolution Process

Resolution in the EDDDS occurs in a two phase process incorporating resolver discovery and resource resolution. The resolver discovery function, similar to that provided by the DDDS, is essential to providing globally resolvable URNs - whether they be eventually resolved by EDDDS servers or other URN systems. Users are able to specify the contents of the flag and service fields completely or partially. The service type is the only field which must match for the a rule to be considered as appropriate.

The discovery process occurs in a 6 step process:

1. The First Well Known Key is applied to the Application Specific String to Produce a Key.
2. The Database represented by the Key is queried for an ordered set of rules.
3. The regular expression in each rule is processed in order until a non-empty string is produced.

4. Check service and flags fields against user requirement, return to step 3 if incompatible.

5. If terminal flags are present return key to the user, otherwise return to step 2 with new key.

6. When terminal lookup found return the key, services and flags to the user or proceed with resolution.

The discovery process is shown in Figure 6.2

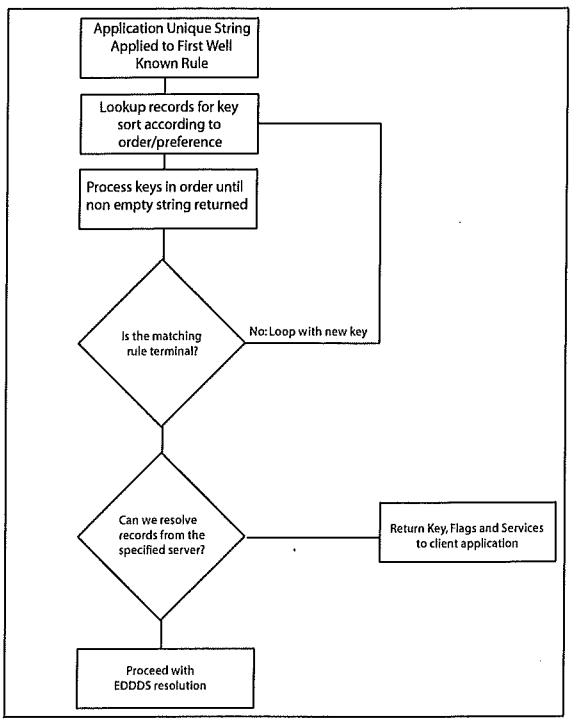

Figure 6.2: The EDDDS discovery algorithm represented as a flow chart

In the discovery process, records are processed until a "DIS" flag is found in a record which satisfies the users service requirements and, where possible, provides the access methods which the user has requested. If the server is a EDDDS resolver server it will have a "NAPTR" flag which informs the client

application that NAPTR records are to be found at the location represented by the key. These records are then processed in a very similar fashion to those above:

1. Query the database represented by the key for ordered NAPTR records (rules).
2. Apply the regular expression rules in order until a non empty key is found which matches user service type requirements.
3. Add the rule which produced the key to a new array and continue processing rules until completed.
4. Sort rule set according to those which best represent user requirements.
5. If selected key is terminal return results to the user otherwise return to step 2 with new key.
6. Upon terminal lookup return key services and flags to the user.

The resolver process is shown in Figure 6.3

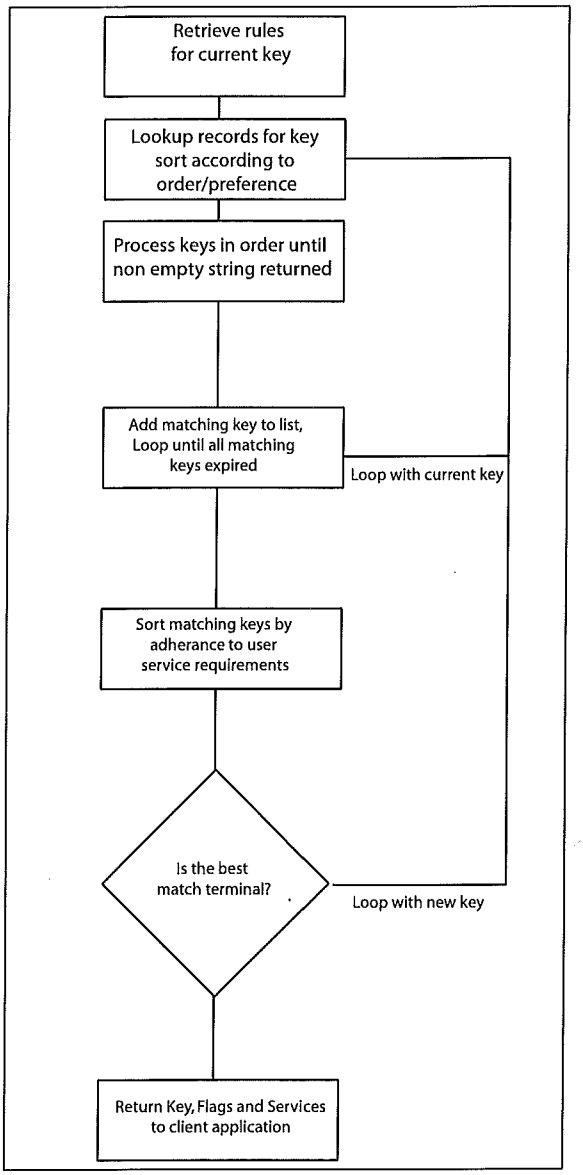

Figure 6.3: The EDDDS resolution algorithm represented as a flow chart

6.3 Evaluation

6.3.1 Usability

Due to the inheritance of functionality from the DDDS, the EDDDS inherits several usability characteristics of the DDDS. The predominant characteristics are the efficiency and security benefits of using the DNS and the challenges of using regular expressions. These characteristics, and the new usability characteristics introduced by the EDDDS, are discussed below from the perspective of the users, publishers and administrators of the resolver.

Users

Were an EDDDS implementation to be introduced at the operating system level, present users of the Internet would notice little difference between the use of URLs and URNs - excepting the persistent availability of the resources they retrieve. Client applications which interact with an operating system EDDDS library, would operate in very similar means to applications which currently use URLs. Examples of such interaction from a user perspective could be the execution of programs which retrieve data files, input as URNs by the user, and the access of resources identified by URNs with tools such as web browsers.

Although the notion of simple resource access is promising, the EDDDS offers client applications and users the potential to specifically address the requirements for the resource they require. Requesting resource information in a specific format, with the optional extended service string, empowers users to select the access method and data format that best suits their needs.

The DDDS, with its simple structure and use of the DNS, proposed a quick means of resolver discovery with the total resolution time reliant on the server

discovered. Although this remains true of the EDDDS in the case of discovery of a non-EDDDS server, complete resolution through the EDDDS never leaves the DNS proving for resource access times comparable to the URL, which also uses the DNS.

Publishers

The EDDDS introduces an opportunity for users to specify much greater detail with regards to the services they require. This consequently allows publishers to tailor their data to the requests of users. Consider, for example, the case of an audio file "xy1" which is available in a smaller sized ".mp3" format and a higher quality, higher sized format ".wav". The use of complex services allows publishers to advertise both formats according to the requirements of the user and order their rules according to their own requirements.

The EDDDS makes extensive use of regular expressions in order to guide resolution. Despite the complications of using regular expressions in the DDDS, the separation of discovery and resolution in the EDDDS introduces an opportunity for the simplification of regular expressions.

Resolver discovery, usually the source of more complex expressions, could quite easily be delegated to the administrators of authoritative URN servers, with the resolution expressions remaining the responsibility of publishers. The advantages of such delegation would be the opportunities for interfaces to be created which accept user input, through means such as forms, and translate their intentions into regular expressions. Such opportunities are increasingly possible as resolver discovery is completed and the size of the potential expression decreases.

Separation of discovery and resolution also leads to a clearer environment for the creation of economic models of URN control. It can be reasoned that publishers buy namespaces, or segments thereof, from administrators. The intro-

duction of the "N2P" service, for the assertion of persistence, would allow publishers to guarantee their resources according to the agreements they have with namespace providers.

Administrators

Other than the scope for the creation of economic models of resolver discovery and/or resource resolution, administrators of URN namespaces are largely unaffected by the changes in the EDDDS. The features of the DNS which allowed them to control server load and improve redundancy in the case of the DDDS still apply and have been furthered with new flexibility in the provision of services. High bandwidth resource applications, such as streaming audio, could be distinguished from lower bandwidth applications by use of the service field and subsequently directed to more appropriate servers.

The notion of complete resource resolution through the EDDDS, instead of discovery and delegation to another system, does provide administrators with both a more predictable idea of traffic requirements, as well as a probable decrease in the bandwidth overheads of URN resolution.

6.3.2 Evolution

Evolution of the EDDDS, as with the DDDS, benefits greatly from the DNS. Scalability and accessibility are helped immensely by the ubiquity and investment made in the DNS by the Internet community.

Scalability has also been revised with the EDDDS. Resource publishers are now able to specify the characteristics of their resources in terms of both their access mechanisms and the content format. This allows for the segmentation of high bandwidth or high demand items to avoid network saturation.

Where the EDDDS differs from the DDDS in terms of design, evolution has been carefully considered. As with all distributed systems, the Internet relies upon standards to ensure interoperability. The new flag and service syntax introduction by the EDDDS mimics that issued by the relevant standards bodies, in this case the IANA and IETF.

The introduction of new resolution services in the EDDDS is very likely. As such the service execution phase of resource retrieval has been separated from the discovery and resolution phases. Implementation of new services simply requires an abbreviated identifier and the population of NAPTR rules.

6.3.3 Security

Given the continued use of the DNS as a database for rule storage, the EDDDS shares the same security considerations as the DDDS. The introduction of data integrity initiatives provides some respite from the challenges discussed previously. However, the security of the EDDDS remains closely reliant on the security of the DNS.

Data integrity in the EDDDS is asserted using the "N2S" service. This service returns the location of digital signature information for the resource being retrieved. "A digital signature of a message is a number dependent on some secret known only to the signer, and, additionally, on the content of the message being signed." (Menezes et al, 1996).

The use of digital signatures relies on the existence of a Public Key Infrastructure (PKI) to enable public key authentication. Public key authentication is an asymmetric authentication process which consists of public and private keys. "The public key defines an encryption transformation Ee, while the private key defines the associated decryption transformation Dd" (Menezes et al, 1996). PKI systems consist of a means to distribute public keys and issue private keys through a "Certificate Authority".

Given a means to distribute public keys to users of a particular namespace, that namespace can begin to assert integrity of its data using digital signatures. Several algorithms for asserting integrity with digital signatures exist, the most common of which are the "RSA" and "DSA" algorithms.

Signing a message requires "transforming the message and some secret information held by the entity into a tag called a signature", according to Menezes et al, (1996). Commonly, a compressed version of the message called a "message digest" is created and signed. This digest is distributed as the digital signature of the original message.

The process of authorisation of a digital signature involves three steps. First, the user computes a message digest of the data they have received. Secondly, the signature is decrypted using the public key of the namespace. Finally, the two message digests are compared. If the digests are the same, the data retrieved is valid and intact.

Providing privacy remains a major challenge in the EDDDS. Namespaces which choose to adopt PKI could in theory use some of the encryption functionality to scramble data transferred between DNS servers and EDDDS clients. Such a system would affect the public and global resolution of URNs, although in some application areas it could be deemed appropriate.

6.4 Summary

This Chapter presented a novel approach to the problem of URN system through extending functionality offered by the DDDS, a mechanism for the discovery of URN resolvers.

Both the EDDDS and DDDDS designs have been evaluated according to their adherence to the URN resolver requirements of usability, security and evolution. However, proof is required that these designs are functional.

Such proof of concept is presented in Chapter 6 with a series of resolution experiments presented for a namespace of language identifiers. The results returned to the user in both the DDDS and EDDDS cases are presented and discussed.

Chapter 7

Experiments and Results

7.1 Overview

The previous two Chapters in this thesis outlined the DDDS, an algorithm for discovering URN resolvers, and the EDDDS, a series of proposed extensions to this algorithm to enable resource resolution. This Chapter presents an example implementation of both the DDDS and the EDDDS for a URN namespace of language identifiers in order to provide proof of concept for these designs.

7.2 The PARADISEC URN namespace

The PARADISEC URN namespace seeks to provide the Pacific and Regional Archive for Digital Sources In Endangered Cultures (PARADISEC) organisation with a persistent means of identifying resources stored within its archives. By applying for an IANA registered URN namespace, this thesis has been able to prescribe the syntax and resolution mechanisms that are required to access data identified with PARADISEC URN identifiers. URN namespaces are issued upon submission of a namespace application document to the IETF. Proposed namespaces are published as Internet-Drafts while accepted namespaces are issued as Internet-Standards, or RFCs.

PARADISEC is a partnership between four major Australian universities which exists to ensure the long term survival of languages and cultures that may have otherwise been forgotten by history. "Over 2000 of the world's 6000 different languages are spoken in Australia, the South Pacific Islands (including around 900 languages in New Guinea alone) and Southeast Asia" (N/A, 2005). It is claimed that "within the next century this number is likely to drop to a few hundred" (N/A, 2005).

Due to the increasing number of research papers being produced citing information within the PARADISEC archive, it is important to develop an identifier scheme which encourages the persistence of these citations and, therefore, the communities ability to access them.

The PARADISEC URN syntax is comprised of three important fields - the collection, item and name. Each of these fields are assigned unique identifiers by PARADISEC curators when a resource is submitted to the archive. The collection is represented by the initials of the contributor, the item is an incrementing number for each item submitted and the name is the filename of the resource. These result in a URN in the form shown by Figure 7.1

```
urn:paradisec:⟨collection⟩:⟨item⟩:⟨name⟩
```

Figure 7.1: PARADISEC identifier syntax

7.2.1 urn:paradisec:AB1:001:A

To illustrate the process by which the DDDS and EDDDS can be applied to PARADISEC URNs, the resolution of URN `urn:paradisec:AB1:001:A` will be described. In both cases, the user is required to submit two pieces of information to the URN resolver - the URN to be resolved and the services required. In this case, an "N2L" URN to URL service will be illustrated for both

the DDDS and EDDDS, with extra parameters incorporated into the EDDDS service specification.

The testbed environment used is outlined in Figure 7.2. This environment consists of a network of Berkley Internet Names Database (BIND) DNS servers referring to two resource repositories, located at the University of Sydney and the Australian Partnership for Advanced Computing. This design was intended to show the potential for delegation between resource locations and the institutions that host them. Whilst these location names were chosen for illustrative purposes, the implementation was conducted on DNS servers at the ANU aliased with these domain names.

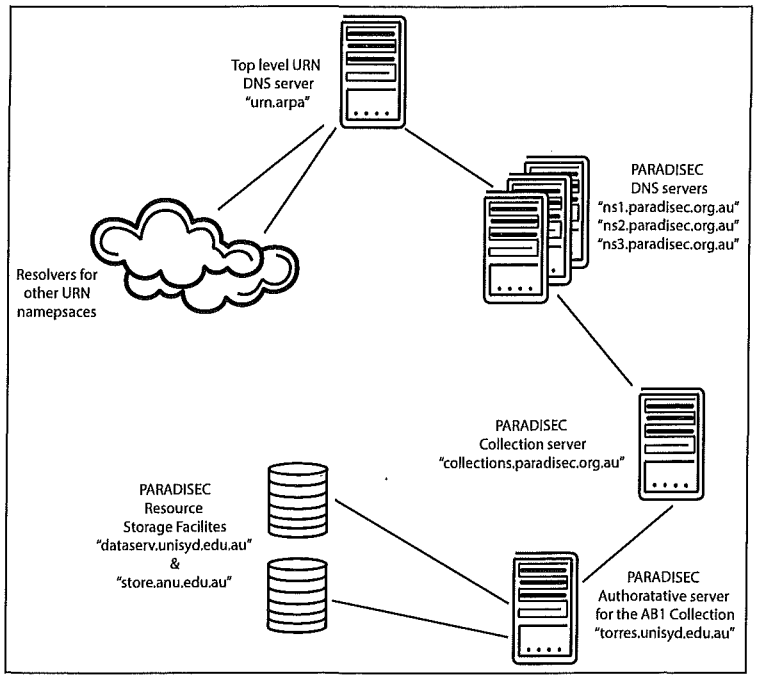

Figure 7.2: An example environment for PARADISEC URN identifiers

In the POSIX regular expression library, case insensitivity is expressed by providing both the lower and upper case values for each character. For example, matching "urn:paradisec" would involve separating each character into its up-

per and lower case values. An expression such as "[uU][rR][nN]:[pP][aA][rR][iI][dD][sS][eE][cC]" results. Therefore, although the PARADISEC namespace is specified as case-insensitive, all examples in this thesis are assuming user input in lower case for the purpose of readability.

7.3 DDDS Implementation

This section outlines the discovery of an authoritative server for the PARADISEC URN "`urn:paradisec:AB1:001:A`". The matching rules processed by the application at each stage of the process are displayed and discussed.

In order to enable resolver discovery, the DDDS algorithm was implemented in a "ddds" Java application. The sequence diagram shown in 7.3 shows the interactions between this application, the "resolverClient" client application and the DNS.

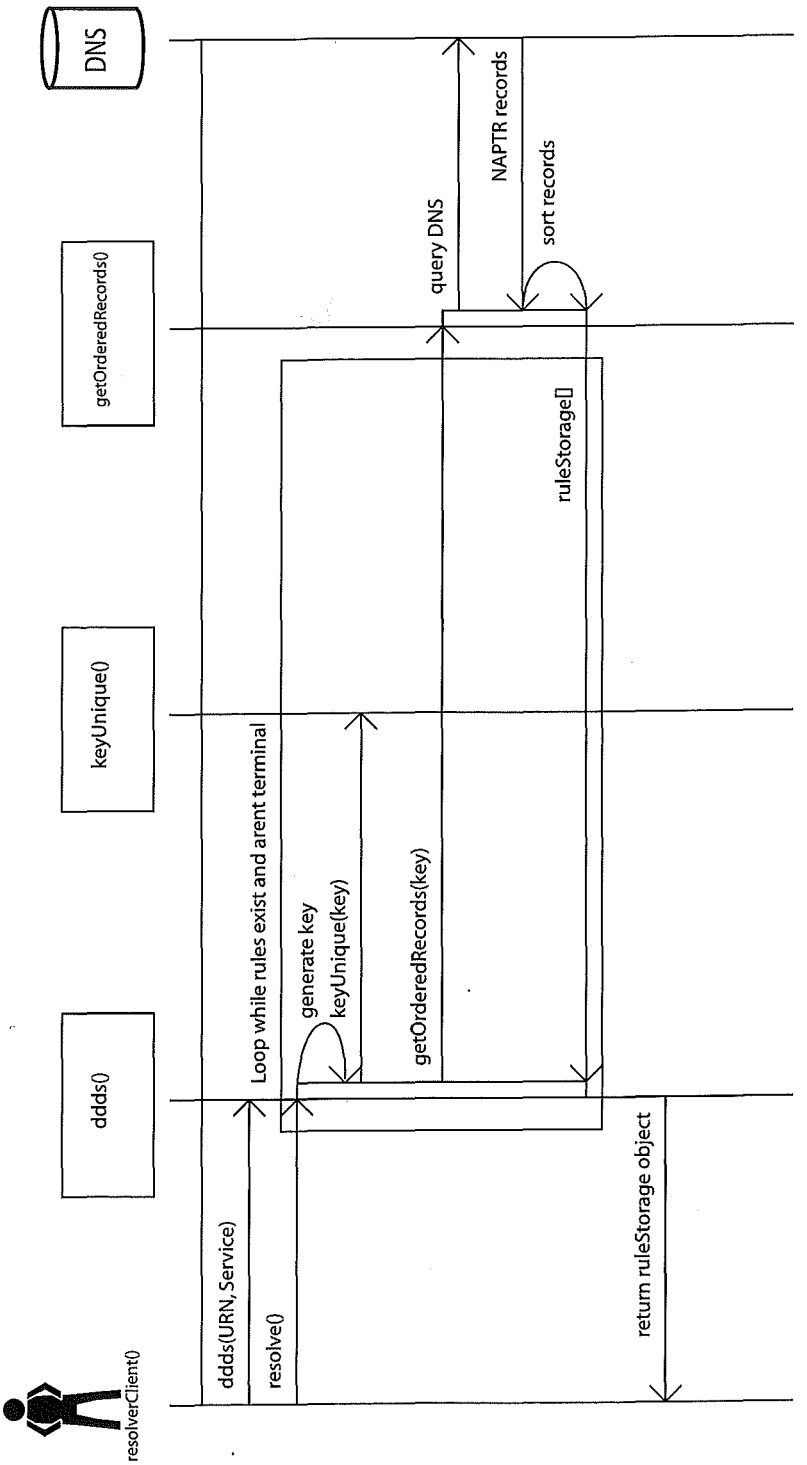

Figure 7.3: UML diagram describes the DDDS java implementation

7.3.1 Discovery

The first query to the DNS seeks all records at the root level of the URN tree. It is envisaged that this would be directed toward a root server at the IANA, as is the case with Internet domain names at present. There exists a urn.arpa domain, administered by the IANA, which could be used for this purpose. The resolver will process each of the rules until it finds a match of service and rule. In the paradisec case, the successful match would be the rule shown in Figure 7.4.

```
IN NAPTR "100" "10" "A" "N2L""[urn:paradisec:[.[^\"&<>[]\^`{\|}~]]+:[.[^\"&<>[]\^`{\|}~]]""ns1.paradisec.org.au"
```

Figure 7.4: PARADISEC namespace NAPTR record

The second query to the DNS is for a resolver for the AB1 collection, now that the authoritative PARADISEC namespace resolver has been located. The rule which matches this requirement is shown in Figure 7.5. Although continued matching of rules for this type of resolver could be expected, it is assumed that the resolver for this collection is authoritative for all items in the AB1 collection. Therefore, the final query, which should be expected to be a terminal query, will result in the specification of an access method for the resolver of this collection. The flag which prescribes this access method, one of either S, A or U, will require different actions from the user.

```
IN NAPTR "100" "10" "" "N2L""urn:paradisec:ab1:[.[^\"&<>[]\^`{\|}~]+]:[.[^\"&<>[]\^`{\|}~]+]"ab1.collections.paradisec.org"
```

Figure 7.5: PARADISEC AB1 collection record

7.3.2 Results

Now that an appropriate resolver for this URN has been located, the DDDS returns one of several types of rules to the user as a successful result. These rules, their format and usage are presented below.

SRV resolution

In the case of a SRV record being returned, the user must be able to further resolve DNS SRV records, as specified in (Esibov et al, 2000). These records provide users with a domain name and a port number on which to access resources. An example rule which would return a SRV record is shown in Figure 7.6.

```
IN NAPTR "100" "10" "S""N2L""urn:paradisec:ab1:001:a""_http._tcp.paradisec.org.au"
```

Figure 7.6: PARADISEC example SRV resolution

SRV records provide a further level of abstraction to the user as they prescribe the service provided to the user according to its definition as managed by the IANA, not according to a port number as in the case of a URL. Although the use of port numbers in URLs give a user a reasonable idea of the service to expect, the use of SRV records assures the users understanding and subsequent use of the correct access protocol. SRV records afford the user the clearest idea of what step to take next in resource resolution however they do not specify syntax for resolution to continue. Therefore, the user or client application can reliably access the server specified, but the query process to resolve the cited URN is unclear and not suggested by the rule returned.

URL resolution

A U flag returned to the user indicates a URL has been returned. This provides information for accessing the resolver cited but, as mentioned, no guarantees on protocol conformity. Such a URL record returned would appear as shown in the replacement field of Figure 7.7. In this instance, the user has been returned what appears to be a website, however, there is no means of specifying the resource access methods required.

```
IN NAPTR "100" "10" "U""N2L" "urn:paradisec:ab1:001:a" "http://www.paradisec.org.au/resolver/"
```

Figure 7.7: PARADISEC example URL resolution

One solution to this is to incorporate a complete or partial substring of the first well known rule into the result through use of a back reference. A back reference is a regular expression concept which involves taking part of the matched string portion and incorporating it into the replacement expression. Incorporated in this manner, the URL resolution approach is of some use to the user, provided the URL scheme returned is actionable by the client application.

A resolution

The most simplistic return value, an A record, simply provides the user with an IP address of a host, which should be responsible for resource resolution. A rule which would produce such a record is shown in Figure 7.8. The domain name represented in the replacement field would be returned as a key and queried by the client resolver for an IP address.

This is the least useful of the various return values as the user is left to guess what the next point of interaction with the resolution process should be. No access protocol or port number are provided.

```
IN NAPTR "100" "10" "A""N2L" "urn:paradisec:ab1:001:a" "resolver.paradisec.org.au"
```

Figure 7.8: PARADISEC example A record resolution

7.4 EDDDS Implementation

As with the DDDDS resolver, the EDDDS resolver developed was implemented as a Java object which can be instantiated with a URN string and a user defined service flag. This implementation is modelled in Figure 7.9.

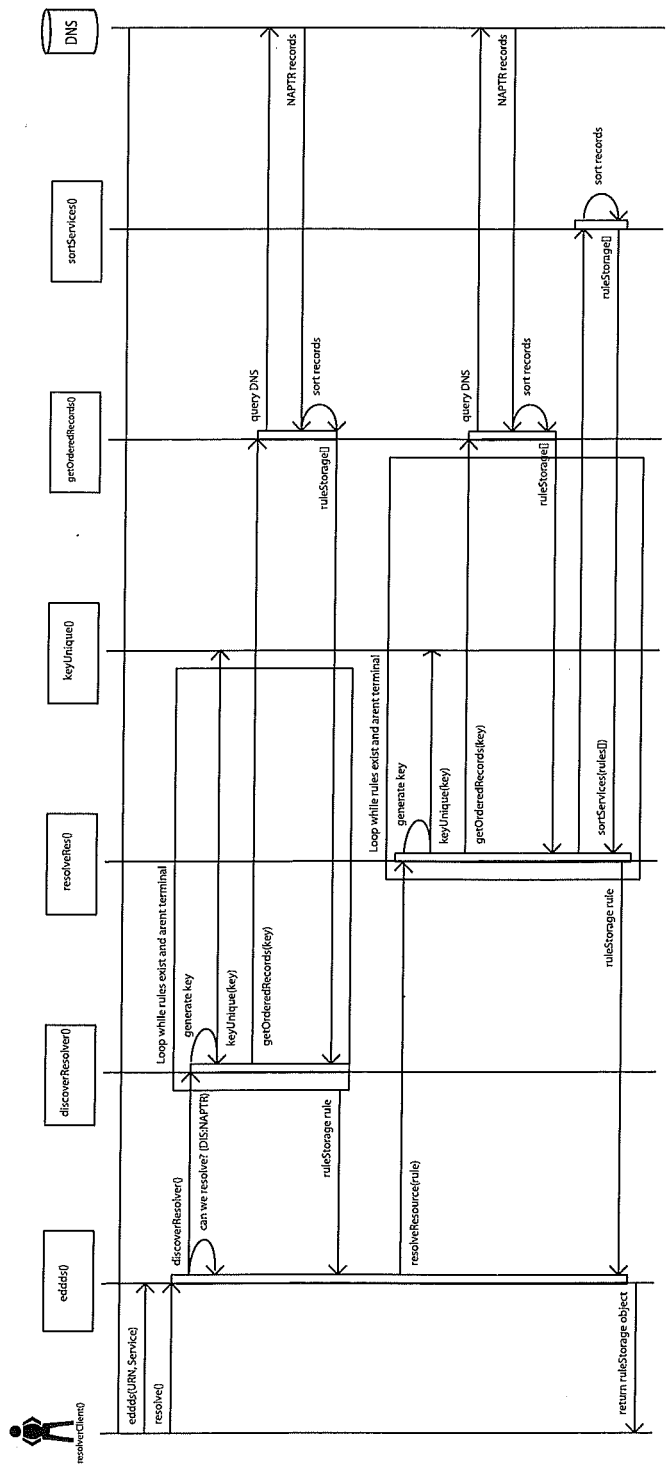

Figure 7.9: UML diagram describes the EDDDS java implementation

In this example, the user has specified the urn "`urn:paradisec:AB1:001:A`" and the flags "N2L:audio/mpeg". The resolution of this identifier follows the discovery and resolution of this resource and presents the various service results that the user can expect.

7.4.1 Discovery

The first rule processed by the EDDDS accepts the URN entered by the user as input and returns a key upon finding a match. As the example URN meets the syntax requirements, the key produced will be the location of the root DNS server for URN records. Traditionally, root servers are managed by the IANA and in this example the server address "urn.arpa" is used. This server is queried for NAPTR records and produces rules for each of the urn namespaces registered. The records shown in Figure 7.10 represent a match for the paradisec namespace.

```
IN NAPTR "100" "10" ""N2L+N2P+N2C+N2S""urn:paradisec:[.[^\"&<>[]\^`{\|}~]]+:[.[^\"&<>[]\^`{\|}~]]+" ns1.paradisec.org.au
IN NAPTR "100" "10" ""N2L+N2P+N2C+N2S""urn:paradisec:[.[^\"&<>[]\^`{\|}~]]+:[.[^\"&<>[]\^`{\|}~]]+" ns2.paradisec.org.au
IN NAPTR "100" "10" ""N2L+N2P+N2C+N2S""urn:paradisec:[.[^\"&<>[]\^`{\|}~]]+:[.[^\"&<>[]\^`{\|}~]]+" ns3.paradisec.org.au
```

Figure 7.10: The top level or root PARADISEC resolver records

As there are three records which match the paradisec namespace, all identically ordered and offering identical services, the EDDDS will use the first record it finds. This record will be different for each resolution attempt. This "round robin" approach to DNS server location is commonly used for high demand servers such as web servers and has the effect of load balancing queries.

Assuming the use of "ns1.paradisec.org.au", the EDDDS algorithm loops and queries this key for a set of rules. Once ordered, the rules appear as shown in Figure 7.11. In this case, given the request for the collection "AB1" the

"ab1.paradisec.org.au" record will match.

```
IN NAPTR "100" "10" "" "N2L+N2P+N2C+N2S" "urn:paradisec:ab1:[.[^\"&<>[]\^`{\|}~]]+" ab1.collections.paradisec.org.au
IN NAPTR "200" "10" "" "N2L+N2P+N2C+N2S" "urn:paradisec:tk1:[.[^\"&<>[]\^`{\|}~]]+:[.[^\"&<>[]\^`{\|}~]]+" tk1.collections.paradisec.org.au
IN NAPTR "300" "10" "" "N2L+N2P+N2C+N2S" "urn:paradisec:jc1:[.[^\"&<>[]\^`{\|}~]]+" jc1.collections.paradisec.org.au
```

Figure 7.11: Ordered rules for a subset of PARADISEC collections

In the final EDDDS loop, the server responsible for the AB1 collection is queried for rules. The rules returned will appear as shown in 7.12. Despite all rules matching the requirements for this query, the lowest ordered rule is terminal and as such, is returned to the user. This record specifies the location of more NAPTR records and therefore can continue to be processed inside the EDDDS.

```
IN NAPTR "100" "10" "DIS:NAPTR" "N2L+N2P+N2C+N2S" "urn:paradisec:ab1:[.[^\"&<>[]\^`{\|}~]]+" ab1.arts.unisyd.edu.au
IN NAPTR "200" "10" "DIS:A" "N2L+N2P+N2C+N2S" "urn:paradisec:ab1:[.[^\"&<>[]\^`{\|}~]]+:[.[^\"&<>[]\^`{\|}~]]+" 150.203.0.178
IN NAPTR "300" "10" "DIS:SRV" "N2L+N2P+N2C+N2S" "urn:paradisec:ab1:[.[^\"&<>[]\^`{\|}~]]+" _tcp._http.handle-srv.unisyd.edu.au
```

Figure 7.12: Discovered resolvers for the AB1 collection

Had the AAA, A, SRV or URL records been ordered higher the user would receive a rule specifying the location of an authoritative server to pursue outside of the EDDDS, as is common in the traditional DDDS. At present, it is not possible to provide further guidance to the user after this value is returned. With some consideration to the way in which records are resolved with systems such as Handle and Purl this would be feasible.

7.4.2 Resolution

Given the key "ab1-res.paradisec.org.au" returned in the final step of the discovery phase had a NAPTR flag, it is now possible to attempt resource resolution using the EDDDS. The client service requirements specified for this process were "N2L:audio/mpeg" and the discovered rule states that N2L is an available service. The following steps involve finding the best fit for these service requirements.

The PARADISEC namespace organises data in a hierarchical fashion according to collection, item and name. The PARADISEC collection AB1 represents several items. This hierarchical structure introduces potential for resolver delegation, as shown in Figure 7.13.

```
IN NAPTR "100" "10" """N2L""urn:paradisec:ab1:001:[.*]"torres.unisyd.edu.au"
IN NAPTR "200" "10" """N2L""urn:paradisec:ab1:012:[.*]"torres-data.paradisec.org.au"
IN NAPTR "300" "10" """N2L""urn:paradisec:ab1:002:[.*]"torres.store.anu.edu.au
```

Figure 7.13: Resolver servers available for the 001 item

The "001" item, and the torres.unisyd.edu.au server responsible for its data storage have several records which match the resource name "A". These records, shown in Figure 7.14, present various different resources identically ordered with preferences used to reflect the more common requests expected by the curators.

```
IN   NAPTR  100 10 "RES:http" "N2L:text/html" "urn:paradisec:ab1:001:a"   http://dataserv.unisyd.edu.au/ab1/001/a.html
IN   NAPTR  100 10 "RES:http" "N2L:audio/wav" "urn:paradisec:ab1:001:a"   http://store.anu.edu.au/ab1/001/a.wav
IN   NAPTR  100 10 "RES:ftp" "N2L:audio/mpeg" "urn:paradisec:ab1:001:a"   http://store.anu.edu.au/ab1/001/a.xml
```

Figure 7.14: Resolution options for the urn:paradisec:AB1:001:A resource

Although all of these records match the key, and are subsequently stored for

service processing, the service request is not matched completely by any of them. Therefore, each of the records are sorted by their adherence to the service "audio/mpeg". The best match is found by performing a regular expression match between the string presented in the service field of the rule and the service specified by the user.

In this case, the best service match is the "N2L:audio/wav" service offered by a resource stored at the ANUSF. As the flag field of this record, "RES:http" is terminal , the EDDDS returns with this rule.

7.4.3 Results

Whereas in the DDDS the address of a discovered server is offered, successful resolution of EDDDS identifiers provides the user with a location address and a statement of the services offered at that location. It is important to note that the execution of services will vary greatly between namespaces - therefore, inclusion of service execution in an EDDDS resolver would greatly restrict usability.

In this example discussed the most simple example of resource resolution, N2L. The record set presented in Figure 7.15 shows an example of the other services which might be offered by a server responsible for this URN. The process of service execution for each of these services is described below.

```
IN   NAPTR  100 10 "RES:http" "N2C" "urn:paradisec:ab1:001:a"   http://dataserv.unisyd.edu.au/ab1/001/a.xml
IN   NAPTR  100 10 "RES:http" "N2S:RSA" "urn:paradisec:ab1:001:a"   http://store.anu.edu.au/ab1/001/a
IN   NAPTR  100 10 "RES:ftp" "N2L:audio/mpeg" "urn:paradisec:ab1:001:a"   http://store.anu.edu.au/ab1/001/a.mpeg
IN   NAPTR  100 10 "RES:http" "N2P" "urn:paradisec:ab1:001:a"   http://policies.paradisec.org/
```

Figure 7.15: Alternative service options for the urn:paradisec:AB1:001:A resource

N2L

In the N2L case, a URL is returned which represents the resource which was queried. In the PARADISEC example, this resource is held by a server which runs a HTTP, FTP or RTSP service.

As all of these resources are actionable, there are several methods by which information can be returned to the user. The most simple method, as used in this example, would be to return the URL only - leaving the task of gathering the resource up to the user. Alternatively, the service could use a platform specific tool, such as GNU "wget" for UNIX systems, to retrieve the resource. Ideally, URN resolution could be implemented as a native or "plugin" function

for web browsers. This would enable users to very simply access resources identified by URNs in a similar fashion to the way web resources are accessed.

In other namespaces such simple resolution might not be appropriate. Location addresses returned may be a subset of an image which is to be retrieved and processed according to further guidelines. Alternatively, they could be email addresses which are to be transparently used by a Message Transfer Agent (MTA) to send a clients email and could even be the location of a web service to be invoked by a client application.

N2C

In the simplest example, metadata associated with a resource could be retrieved in one of the three methods mentioned above. As metadata in the PARADISEC namespace is actionable via HTTP, the process of returning a resource location is sufficient.

URC information can be stored in several formats from informal means such as plain text files, through to established standards, such as the Dublin Core Metadata Initiative (DCMI). Although these formats are all representable through a URL, applications may retrieve, interpret and process metadata quite differently.

N2S

Digital signatures in the EDDDS provide a means for users to ensure that the data they retrieve is valid. Digital signatures were discussed in Chapter 6.

The PARADISEC implementation returns only a URL location of a digital signature for the record listed. Matching and processing this signature against a hash of the resource retrieved is achievable through various libraries for vari-

ous programming languages.

Distinguishing between signatures encrypted with the DSA or RSA encryption algorithms is done through use of the service flag. Implementation of a public key encryption system, necessary for the distribution of public and private keys to perform signature checks, is outside of the scope of the EDDDS.

N2P

There are several ways in which persistence can be asserted by a resource. In the case where a user is simply using the persistence value as a guide to the longevity of their resource, a textual representation may be appropriate. In other examples, the date format may need to follow a machine consumable format such as the Julian date format.

Furthermore, functionality exists to use the Time To Live (TTL) value in the DNS. This value, commonly used for caching purposes, could be interpreted as a guaranteed time frame for the resource information to remain static by either the user or the client application.

In the PARADISEC example, users would use the persistence value as a guide to estimate the longevity of their citations. Given this direct user interaction, dates are displayed in a simple "DDMMYYYY" format.

In the PARADISEC namespace example, persistence is stated as a best effort estimation of the life of the identifier. In other examples, this assertion may be guaranteed by various administrators and enforced by way of economic agreements with publishers.

7.5 Summary

The implementation of the DDDS and EDDDS systems proves that the algorithms suggested in Chapters 5 and 6 can solve the problem of URN discovery and resolution. Furthermore, the successful processing of rules toward resolution justifies the selection of the DNS as a database.

The demonstrated resolution of the PARADISEC identifier `urn:paradisec:AB1:001:A` illustrates the various strengths of the DDDS and EDDDS systems, most notably concerning the opportunities for providing for and asserting the persistence of resources via flexible NAPTR records. Any of the rules traversed in this example could be extensively restructured, and the resource locations subsequently moved, without any changes to the structure of the URN identifier. This would not have been possible with the URL without employing specific technologies such as web-server redirection.

Implementation of the DDDS provides the user with a very simple set of results, as specified by the DDDS design. Whilst in theory the goal of resolver discovery has been reached, a major obstacle exists regarding the clear definition of the interactions between the DDDS and the resolvers discovered. This problem is not solved in the design or implementation of the EDDDS as it requires consensus between all other URN systems on a standard for querying their resolvers. The availability of a global discovery system does, however, provide these groups with a means for implementing their own EDDDS systems which incorporate a namespace and resolver specific means of dealing with the resolver servers discovered.

The EDDDS implementation discussed in this Chapter provides a means for users to access a variety of services identified by URNs. Although simple discovery is an option in this implementation, complete resolution is available with various options for delegation and delivery of user requirements.

The various features of the DDDS and the EDDDS have previously been critiqued in accordance with the aims of usability, security and evolution. This implementation provides proof of functionality for these algorithms.

Chapter 8

Conclusion

This work sought a means of persistent identification which, unlike the systems currently available, was accessible and "actionable" in a uniform and global manner.

Resources are presently identified on the Internet through use of a URL. This work proved the unsuitability of the URL for persistent resource access. Resolving URLs on the Internet is made possible through use of the DNS. This work sought to explore current approaches to partial URN resolution with the DNS and suggest extensions which would provide for complete URN resolution.

8.1 Contributions

- Does the proposed extension to the DDDS provide an adequate resource resolution system for the URN?

The resolution system proposed in this work, the EDDDS, achieves both targets of being actionable and persistent while conforming to many of the requirements outlined in (Arends et al, 2005).

The use of the URN for persistent identification has always seemed optimal. A comparison of the current approaches to Internet identification against a list

of desirable characteristics justifies this assumption.

Given the use of such an established identifier, the current attempts at resolution were examined and found to be lacking in several areas. Most notable was the inability of any of the current URN systems to provide globally actionable identifiers without the use of a proxy resolution service. Proxy resolution - a process usually as simple as encapsulating a URN into a URL - undermines the goal of persistence by making identifiers reliant on the proxy and the technology the proxy implements.

The DDDS, a URN system capable of partial URN resolution, was found to be quite useful in a number of respects. Its design, once implemented and evaluated, was found to be suited to the goals of this work. The DDDS did not, however, completely satisfy the goal of actionable identification.

In order to satisfy this goal, a number of extensions to the structure of the DDDS were proposed and named the EDDDS. These extensions were implemented and proved to provide an actionable identifier in the form of the URN.

- Can the URN resolver developed be used to resolve resources for a URN namespace.

The proof of concept implementations of both the DDDS and EDDDS achieved several outcomes. First, technical feasibility of the algorithms proposed was demonstrated. This demonstration included proof of successful interactions with the NAPTR DNS record and successful results in varied resolution scenarios. Secondly, this work demonstrated the ease by which a group can deploy persistent means of identification. The PARADISEC namespace implements its own naming hierarchy to meet the goals of identification in the language community.

It has been understood throughout this research that an actionable identifier

is just one of the many factors that affect the persistence of a resource on the Internet. That said, the flexibility introduced by the EDDDS encourages persistence by virtue of its design. Requiring only suitable management by publishers and administrators of URN identifiers to ensure persistence.

Whilst there are several questions to be answered and policies to be developed before the EDDDS could be implemented on a wide scale, the design proposed provides a strong foundation upon which the URN identifier can reach its potential on the Internet.

8.2 Future Work

Although the EDDDS provides a means for groups to start resolving their own URNs now, there are several considerations for future work to ensure its successful implementation on a wider scale.

Usability

At present, the EDDDS provides an efficient means of retrieving resource information, but no means of efficient publishing. The use of regular expressions requires unrealistic technical proficiency on the part of the user. One possible solution is a system of forms where most expressions could be simplified on a per-namespace basis and incorporated into a web interface to the DNS for use by publishers.

Although the various EDDDS implementations will be subject to their own performance requirements, the ability of the DNS to rapidly return queries affects all Internet users. Incorrect use of the NAPTR record, being either the use of exceedingly long fields or simply too many records, could adversely affect DNS servers - especially the root servers. This problem may be addressed with either replication of servers or controls on NAPTR format, however, modelling will be necessary to devise acceptable use.

Security and Privacy

Despite proven and reliable means of integrity assertion within EDDDS, the DNS remains flawed in relation to both security and privacy. Future work in this field will remain hampered by the need to maintain interoperability between DNS servers across the world.

Evolution

It is unreasonable to expect that universal adoption of the EDDDS for resource

resolution will occur. Therefore, interaction with other resolvers, such as those discussed in this work, will need to be improved. At present, authoritative resolvers can be located regardless of type - however, there is no means for these servers to assert their capabilities or the means by which they can be accessed.

Policy

Given the amount of administrative control used by governments and industry bodies over assigned Internet domain names and URL namespace identifiers, it is naive to imagine URNs, whose namespace identifiers represent similar interests as domain names, will be without such control.

It remains to be seen what level of control will be required in order to maintain operation of a global network of URNs. At the very least restriction of the IETF policy on assigning URN namespace identifiers can be anticipated.

In line with this development in policy, an economic model for the issuing of URN namespace identifiers, maintenance of authoritative servers and issuance of individual resource identifiers may be deemed necessary.

Appendix A

Java Implementation

A.1 Client Application - "resolverclient.java"

```
/*
 * Application creates and invokes an DDDS resolver object.
 * @author Luke Brown luke@bur.st
 * @version 0.1
 */
public class resolverClient {

  public static void main(String args[]) {

    String applicationUnique = "urn:paradisec:ab1:001:a";
    String serviceRequired = "N2L";


    /* Attempt resolution using the DDDS */
    try {

      /* Create DDDS Resolver object */
      ddds d = new ddds(applicationUnique, serviceRequired);
      /* Set Debug Flag */
      d.setDebug();
      /* Fill storage object */
      ruleStorage dddsResults = d.resolve();
      /* Print retrieved rule and generate final key */
      System.out.println("Rule returned: \n"
                + "Order = "
                + dddsResults.order
                + "\n"
                + "Preferences = "
                + dddsResults.preference
                + "\n"
                + "Flags = "
                + dddsResults.flags
                + "\n"
                + "Services = "
                + dddsResults.service
                + "\n"
                + "Expression = "
```

```
                + dddsResults.regexp
                + "\n"
                + "Replacement␣=␣"
                + dddsResults.replacement
                + "\n\n"
                + "Final␣key␣=␣"
                + applicationUnique.replaceAll(dddsResults.regexp,
                                dddsResults.replacement));
    } catch (Exception e) {
      System.err.println(e.getMessage());
      System.out.println("Error␣resolving␣URN!");
    }

    /* Attempt resolution using the EDDDS */
    try {

      /* Service requirements extended */
      serviceRequired = "N2L:audio/mpeg";

      /* Create EDDDS Resolver object */
      eddds e = new eddds(applicationUnique, serviceRequired);
      /* Set debug flag to true */
      e.setDebug();
      /* Fill ruleStorage object test with resolved ruleset */
      ruleStorage edddsResults = e.resolve();
      /* Print retrieved rule */
      System.out.println("Rule␣returned:␣\n"
                + "Order␣=␣"
                + edddsResults.order
                + "\n"
                + "Preferences␣=␣"
                + edddsResults.preference
                + "\n"
                + "Flags␣=␣"
                + edddsResults.flags
                + "\n"
                + "Services␣=␣"
                + edddsResults.service
                + "\n"
                + "Expression␣=␣"
                + edddsResults.regexp
                + "\n"
                + "Replacement␣=␣"
                + edddsResults.replacement
                + "\n");

    } catch (Exception e) {
      System.err.println(e.getMessage());
      System.out.println("Error␣resolving␣URN!");
    }
  }
}
```

A.2 Storage Class - "rulestorage.java"

```
/*
 * This class provides a rule storage structure for NAPTR records
 * @author Luke Brown luke@bur.st
 * @version 0.1
 */

public class ruleStorage {

  int order;
  int preference;
  String flags;
  String service;
  String regexp;
  String replacement;

  /*
   * ruleStorage constructor, builds a ruleStorage object to represent
   * the given values of a NAPTR record.
   */
  public ruleStorage(int o, int p, String f, String s, String reg, String rep) {

    order = o;
    preference = p;
    flags = f;
    service = s;
    regexp = reg;
    replacement = rep;

  }
}
```

A.3 DDDDS object - "ddds.java"

```
mport javax.naming.*;
import javax.naming.directory.*;
import java.util.*;
import java.util.regex.*;
import java.util.Enumeration.*;

/*
 * This class provides an implementation of the DDDS URN resolution algorithm.
 * @author Luke Brown, luke@bur.st
 * @version 0.1
 */
public class ddds {

  final String firstKnown = "urn:[\\w&&[^#%/]]+:.*";
  String userService = "";
  String appUnique = "";
  String key = "";
  Vector v = new Vector();
  boolean debug = false;

  /* Class Constructor
   * @param Accepts a String URN and a String of the required Services
   */
  public ddds(String aUni, String usrSvc) {

    /* Check for valid arguments */
    if (Pattern.matches(firstKnown, aUni)) {
      this.userService = usrSvc;
      this.appUnique = aUni;
      key = aUni.replaceAll(firstKnown, "urn.arpa");
    } else {
      /* Notifies Client app */
      System.err.println("Error:_Malformed_URN");
      System.exit(0);
    }
  }

  /*
   * @param Method takes URN supplied through constructor and attempts resolution
   * @return A string array of results is returned
   */
  public ruleStorage resolve() {

    ruleStorage result = new ruleStorage(0, 0, "", "", "", "");

    boolean rewrite = true;
    boolean terminal = false;

    /* we have a key, we have a aus, loop commences here: */
    main_while:
    /* Continue while rewrite rules are present and no terminal flags are found */
    while (rewrite && !terminal) {

      /* if key is unique add to list */
      if (keyUnique(v, key)) {
```

```
      v.addElement(key);
      /* else exit with loop condition */
    } else {
      System.err.println("Error: Key Unchanged, Loop Detected");
      rewrite = false;
      break main_while;
    }

    /* Get sorted record set for current Key */
    if (debug) {
      System.out.println("Debug: Looking up key: " + key);
    }
    ruleStorage[] ruleSet = getSortedRecords(key);

    /* Process keys returned in order */
    for (int i = 0; i < ruleSet.length; i++) {

      if (debug) {
        System.out.println("Debug: Record: " + i + " Expression: "
            + ruleSet[i].regexp + " Replacement: "
            + ruleSet[i].replacement + " Flags: "
            + ruleSet[i].flags);
      }

      /* A successful rule Must: match, produce a non
      empty string and have valid (or no) flags */
      if (Pattern.matches(ruleSet[i].regexp, appUnique)
          && (appUnique.replaceAll(ruleSet[i].regexp,
              ruleSet[i].replacement) != "")
          && ruleSet[i].service.equalsIgnoreCase(userService)) {

        /* Rule acceptable, generate new key */
        key = appUnique.replaceAll(ruleSet[i].regexp,
            ruleSet[i].replacement);
        if (debug) {
          System.out.println("Debug: Generated New Key!: " + key);
        }

        /* Check if this rule is terminal */
        String flag = ruleSet[i].flags.substring(0,1);
        if (flag.equalsIgnoreCase("A") || flag.equalsIgnoreCase("U")
          || flag.equalsIgnoreCase("S") || flag.equalsIgnoreCase("P")) {
          terminal = true;
        }

        /* Terminal records are returned to the user */
        if (terminal) {
          if (debug)
            System.out.println("Debug: Terminal Flags Found: "
                + ruleSet[i].flags);
          result = ruleSet[i];
        }
        /* End Loop, Terminal Flags located */
        break;
      }
    }
  }

/* Return successful rule to user */
```

```
  return result;
}

private ruleStorage[] getSortedRecords(String key) {

  int numNaptrs = 0;
  ruleStorage[] ruleSet = new ruleStorage[100];

  try {
    //define DNS server environment
    Hashtable env = new Hashtable();
    env.put("java.naming.provider.url", "dns://127.0.0.1/");
    env.put("java.naming.factory.initial",
        "com.sun.jndi.dns.DnsContextFactory");
    DirContext DnsRes = new InitialDirContext(env);
    //perform lookup
    if (debug) {
      System.out.println("Debug: Performing Lookup with key: " + key);
    }
    Attributes attr = DnsRes.getAttributes(key,
        new String[] { "NAPTR" });
    NamingEnumeration attr1 = attr.getAll();

    if (attr1.hasMore()) {
      //grab csv string of records
      String nextKey = ((attr1.next()).toString());
      //break records into individual strings
      String[] nextArr = nextKey.split(", ");
      ruleStorage[] temp = new ruleStorage[(nextArr.length) - 1];
      //foreach string build a ruleStorage object
      for (int i = 0; i < nextArr.length; i++) {
        //split em with regex
        String[] tempArr = nextArr[i].split(" ");
        int buffer = 0;
        //if the first element is the label
        if (tempArr[0].equals("NAPTR:")) {
          buffer = 1;
        }

        ruleStorage napStruct = new ruleStorage(Integer
            .parseInt(tempArr[0 + buffer]), Integer
            .parseInt(tempArr[1 + buffer]),
            tempArr[2 + buffer], tempArr[3 + buffer],
            tempArr[4 + buffer], tempArr[5 + buffer]);
        ruleSet[i] = napStruct;
        numNaptrs++;
      }
    } else {
      System.err.println("Error: No DNS Records Returned");
      System.exit(1);
    }
  } catch (Exception e) {
    System.err.println("Error!: " + e);
    System.exit(1);
  }

  /* sort rules by order then preference ... */
  for (int z = (numNaptrs - 1); z >= 0; z--) {
    for (int j = 1; j <= z; j++) {
```

```
        /* If we're more important than the next up, transfer */
        if (ruleSet[j - 1].order > ruleSet[j].order) {
          ruleStorage tempOrder;
          tempOrder = ruleSet[j - 1];
          ruleSet[j - 1] = ruleSet[j];
          ruleSet[j] = tempOrder;

        }
        /* If we're more important AND a higher preference, transfer */
        else if (ruleSet[j - 1].order == ruleSet[j].order
            && ruleSet[j - 1].preference > ruleSet[j].preference) {
          ruleStorage tempPref;
          tempPref = ruleSet[j - 1];
          ruleSet[j - 1] = ruleSet[j];
          ruleSet[j] = tempPref;
        }
      }
    }

    ruleStorage[] ruleSetTidy = new ruleStorage[numNaptrs];
    System.arraycopy(ruleSet, 0, ruleSetTidy, 0, numNaptrs);
    return ruleSetTidy;
  }

  /*
   * Method checks keys are unique
   * @param Vector of seen keys and current key
   * @return Returns True is key is unique, false if key has been seen.
   */
  private boolean keyUnique(Vector v, String key) {

    boolean found = true;
    Iterator vI = v.iterator();

    while (vI.hasNext()) {

      if (vI.next() == key) {
        found = false;
      }
    }
    return found;
  }

  /*
   * Method sets debug flag for verbose output
   */
  public void setDebug() {
    debug = true;
  }
}
```

A.4 EDDDS object - "eddds.java"

```
import javax.naming.*;
import javax.naming.directory.*;
import java.util.*;
import java.util.regex.*;
import java.util.Enumeration.*;

/*
 * This class provides an implementation of the DDDS URN resolution algorithm.
 * @author Luke Brown, luke@bur.st
 * @version 0.1
 */
public class eddds {

  final String firstKnown = "urn:[\\w&&[^#%/]]+:.*";
  boolean debug = false;

  String userService = "";
  String key = "";
  String appUnique = "";

  /* Class Constructor checks URN syntax conformance and generates first key
    * @param String representations of the application unique string (URN entered)
    * and the firstKnown rule
    */
  public eddds(String aUni, String usrSvc) {

    /* Check for valid arguments */
    if (Pattern.matches(firstKnown, aUni)) {
      this.userService = usrSvc;
      this.appUnique = aUni;
      this.key = appUnique.replaceAll("urn:([\\w&&[^#%/]]+):.*", "urn.arpa");
    } else {
      /* Invalid URN rejected */
      System.err.println("Error:␣Malformed␣URN");
    }
  }

  /*
   * Method resolve calls the discovery and resolution
   * phase methods to guide resource resolution
   */
  public ruleStorage resolve() {

    ruleStorage result = discoverResolver();
    if (result.flags.equalsIgnoreCase("DIS:NAPTR")) {
      result = resolveResource(result);
    }

    return result;

  }

  /*
   * Resolver Discovery method, implements an
   * extended DDDS URN resolver discovery algorithm.
   */
```

```
private ruleStorage discoverResolver() {

  ruleStorage resultDis = new ruleStorage(0, 0, "", "", "", "");
  Vector vDis = new Vector();
  boolean rewrite = true;
  boolean terminal = false;


  /* we have a key, we have a aus, loop commences here: */
  main_while:
    /* Continue while rewrite rules are present and no terminal flags are found */
    while (rewrite && !terminal) {

      /* if key is unique add to list */
      if (keyUnique(vDis, key)) {
        vDis.addElement(key);
        /* else exit with loop condition */
      } else {
        System.err.println("Error: Key Unchanged, Loop Detected");
        rewrite = false;
        break main_while;
      }

      /* Get sorted record set for current Key */
      if (debug) {
        System.out.println("Debug: Looking up key: " + key);
      }
      ruleStorage[] ruleSet = getSortedRecords(key);

      /* Process keys returned in order */
      for (int i = 0; i < ruleSet.length; i++) {

        if (debug) {
          System.out.println("Debug: Record: " + i + " Expression: "
                    + ruleSet[i].regexp + " Replacement: "
                    + ruleSet[i].replacement + " Flags: "
                    + ruleSet[i].flags);
        }

        /* A successful rule Must: match, produce a non empty string and have valid service type*/
        if (Pattern.matches(ruleSet[i].regexp, appUnique) &&
          (appUnique.replaceAll(ruleSet[i].regexp, ruleSet[i].replacement) != "") &&
          ruleSet[i].service.substring(0,3).equalsIgnoreCase(userService.substring(0,3))) {

          /* Rule acceptable, generate new key */
          key = appUnique.replaceAll(ruleSet[i].regexp, ruleSet[i].replacement);
          if (debug) {
            System.out.println("Debug: Generated New Key!: " + key);
          }

          /* Check if this rule is terminal,
            * non empty strings (literal "" excepted) are terminal
            */
          terminal = (ruleSet[i].flags.equalsIgnoreCase("")
          || ruleSet[i].flags.equalsIgnoreCase("\"\"")) ? false:true;
          /* Terminal records are returned to the user */
          if (terminal) {
            if (debug) {
              System.out.println("Debug: Terminal Flags Found: "
```

```
                    + ruleSet[i].flags);
            }
            resultDis = ruleSet[i];
            return resultDis;
            /* End Loop, Terminal Flags located */
          }
          break;
        }
      }
    }

    /* Return successful rule to user */
    return resultDis;
  }

  /*
   * Method accepts a terminal resolver discovery rule, finds the resolver
   * and queries it for resource information.
   * @param a ruleStorage object which describes the discovered resolver.
   */
  private ruleStorage resolveResource(ruleStorage disRule) {

    ruleStorage result = new ruleStorage(0, 0, "", "", "", "");
    ruleStorage[] matched = new ruleStorage[100];
    int matchedKeys = 0;
    Vector vRes = new Vector();
    boolean rewrite = true;
    boolean terminal = false;

    key = appUnique.replaceAll(disRule.regexp, disRule.replacement);

    /* we have a key, we have a aus, loop commences here: */
main_while:
      /* Continue while rewrite rules are present and no terminal flags are found */
      while (rewrite && !terminal) {

        /* if key is unique add to list */
        if (keyUnique(vRes, key)) {
          vRes.addElement(key);
          /* else exit with loop condition */
        } else {
          System.err.println("Error: Key Unchanged, Loop Detected");
          rewrite = false;
          break main_while;
        }

        /* Get sorted record set for current Key */
        if (debug) {
          System.out.println("Debug: Looking up key: " + key);
        }
        ruleStorage[] ruleSet = getSortedRecords(key);
        System.out.println("ruleset length is " + ruleSet.length);
        /* Process keys returned in order */
        matchedKeys = 0;
        for (int i = 0; i < ruleSet.length; i++) {

          if (debug) {
            System.out.println("Debug: Record: " + i + " Expression: "
                    + ruleSet[i].regexp + " Replacement: "
```

```
                + ruleSet[i].replacement + "␣Flags:␣"
                + ruleSet[i].flags);
        }

        /* A successful rule Must: match, produce a non
            empty string and have valid (or no) flags */
        if (Pattern.matches(ruleSet[i].regexp, appUnique)
          && (appUnique.replaceAll(ruleSet[i].regexp,
                    ruleSet[i].replacement) != "")
          && ruleSet[i].service.substring(0,3).equalsIgnoreCase(userService.substring(0,3))) {
          matched[i] = ruleSet[i];
          matchedKeys++;
        }
      }

      if (matchedKeys == 0) {

        System.err.println("No␣matching␣records␣for␣Key:␣" + key);
        break main_while;
      }

      ruleStorage[] matchedTidy = new ruleStorage[matchedKeys];
      System.arraycopy(matched, 0, matchedTidy, 0, matchedKeys);
      matchedTidy = sortService(userService, matchedTidy);

      /* Rule acceptable, generate new key */
      key = appUnique.replaceAll(matchedTidy[0].regexp,
                matchedTidy[0].replacement);
      if (debug) {
        System.out.println("Debug:␣Generated␣New␣Key!:␣" + key);
      }

      /* Check if this rule is terminal */
      terminal = (matchedTidy[0].flags.equalsIgnoreCase("") ||
            matchedTidy[0].flags.equalsIgnoreCase("\"\""))
        ? false:true;

      /* Terminal records are returned to the user */
      if (terminal) {
        if (debug)
          System.out.println("Debug:␣Terminal␣Flags␣Found:␣"
                + matchedTidy[0].flags);
        result = matchedTidy[0];
        /* End Loop, Terminal Flags located */
        break;
      }

    }
  /* Return successful rule to user */
  return result;
}

/*
 * Method orders services by best effort match
 * @param service flags and unordered array
 * @return ordered array of ruleStorage objects
 */
private ruleStorage[] sortService(String services, ruleStorage[] rules) {
```

```
String userContent = "";
String userDelivery = "";

/* Ascertain User requirements */
String[] userServices = services.split(":");
if (userServices.length == 2) {
  String[] userServiceContent = userServices[1].split("/");
  userContent = userServiceContent[0];
  userDelivery = userServiceContent.length == 2 ? userServiceContent[1]:"";
}

System.out.println("Debug:␣Sorting␣Array:␣User␣values:\n" + userContent + ",␣" + userDelivery);

for (int z = (rules.length - 1); z >= 0; z--) {
  for (int j = 1; j <= z; j++) {

    /* Get Content and Delivery strings for both current and next record up */
    String tempLowerContent, tempUpperContent, tempLowerDelivery, tempUpperDelivery;

    String[] tempLowerServices = rules[j].service.split(":");
    tempLowerServices = tempLowerServices[1].split("/");
    tempLowerContent = tempLowerServices[0];
    tempLowerDelivery = tempLowerServices.length == 2 ? tempLowerServices[1]:"";

    String[] tempUpperServices = rules[j-1].service.split(":");
    tempUpperServices = tempUpperServices[1].split("/");
    tempUpperContent = tempUpperServices[0];
    tempUpperDelivery = tempUpperServices.length == 2 ? tempUpperServices[1]:"";

    System.out.println("Debug:␣Sorting␣Array:␣\nUpper␣Values:␣"
              + tempUpperContent + ",␣"
              + tempUpperDelivery
              + "\nLower␣Values:"
              + tempLowerContent + ",␣"
              + tempLowerDelivery);

    /* if this rule meets service CONTENT and DELIVERY
     * (and above doesnt) transfer
     */
    if (tempLowerContent.equalsIgnoreCase(userContent) &&
      tempLowerDelivery.equalsIgnoreCase(userDelivery) &&
      !tempUpperContent.equalsIgnoreCase(userContent) &&
      !tempUpperDelivery.equalsIgnoreCase(userDelivery) ||
      /* if this rule meets service CONTENT and DELIVERY
      * (and above only meets content) transfer
      */
      tempLowerContent.equalsIgnoreCase(userContent) &&
      tempLowerDelivery.equalsIgnoreCase(userDelivery) &&
      tempUpperContent.equalsIgnoreCase(userContent) &&
      !tempUpperDelivery.equalsIgnoreCase(userDelivery) ||
      /*
       * OR this rule meets service CONTENT (and above doesnt)
       * transfer
       */
      tempLowerContent.equalsIgnoreCase(userContent) &&
      !tempUpperContent.equalsIgnoreCase(userContent) ||
      /*
       * OR this rule meets service CONTENT
       * and DELIVERY (and above DOES) and we
```

```
         * have a lower order, transfer
         */
        tempLowerContent.equalsIgnoreCase(userContent) &&
        tempLowerDelivery.equalsIgnoreCase(userDelivery) &&
        tempUpperContent.equalsIgnoreCase(userContent) &&
        tempUpperDelivery.equalsIgnoreCase(userDelivery) &&
        rules[j].order < rules[j-1].order ||
        /*
         * OR this rule meets service CONTENT
         * (and above DOES) and we have a lower
         * order, transfer
         */
        tempLowerContent.equalsIgnoreCase(userContent) &&
        !tempLowerDelivery.equalsIgnoreCase(userDelivery) &&
        tempUpperContent.equalsIgnoreCase(userContent) &&
        !tempUpperDelivery.equalsIgnoreCase(userDelivery) &&
        rules[j].order < rules[j-1].order ||
        /*
         * OR this rule meets service CONTENT
         * and DELIVERY (and above DOES) and we
         * have a lower preference, transfer
         */
        tempLowerContent.equalsIgnoreCase(userContent) &&
        tempLowerDelivery.equalsIgnoreCase(userDelivery) &&
        tempUpperContent.equalsIgnoreCase(userContent) &&
        tempUpperDelivery.equalsIgnoreCase(userDelivery) &&
        rules[j].preference < rules[j-1].preference ||
        /*
         * OR this rule meets service CONTENT
         * (and above DOES) and we have a lower
         * preference, transfer
         */
        tempLowerContent.equalsIgnoreCase(userContent) &&
        !tempLowerDelivery.equalsIgnoreCase(userDelivery) &&
        tempUpperContent.equalsIgnoreCase(userContent) &&
        !tempUpperDelivery.equalsIgnoreCase(userDelivery) &&
        rules[j].preference < rules[j-1].preference
        ) {

        /* debug message */
        System.out.println("Debug: attempting transfer");
        /* perform transfer operation */
        ruleStorage tempTransfer = rules[j -1];
        rules[j -1] = rules[j];
        rules[j] = tempTransfer;
      }
    }
  }

  return rules;
}

/*
 * Method retrieves records for a given key
 * @param key to form the next eddds database query
 * @return ordered array of ruleStorage objects
 */
private ruleStorage[] getSortedRecords(String key) {
```

```
int numNaptrs = 0;
ruleStorage[] ruleSet = new ruleStorage[100];

try {
  /* Configure DNS parameters */
  Hashtable env = new Hashtable();
  env.put("java.naming.provider.url", "dns://127.0.0.1/");
  env.put("java.naming.factory.initial",
      "com.sun.jndi.dns.DnsContextFactory");
  DirContext DnsRes = new InitialDirContext(env);

  /* Query DNS with supplied key */
  if (debug) {
    System.out.println("Debug: Performing Lookup with key: " + key);
  }
  Attributes attr = DnsRes.getAttributes(key,
                      new String[] { "NAPTR" });
  NamingEnumeration attr1 = attr.getAll();

  /*  If we got results process them */
  if (attr1.hasMore()) {
    /* csv string of records */
    String nextKey = ((attr1.next()).toString());
    /* break records into individual strings */
    String[] nextArr = nextKey.split(", ");
    ruleStorage[] temp = new ruleStorage[(nextArr.length) - 1];

    //foreach string build a ruleStorage object
    for (int i = 0; i < nextArr.length; i++) {
      //split em with regex
      String[] tempArr = nextArr[i].split(" ");
      int buffer = 0;
      //if the first element is the label
      if (tempArr[0].equals("NAPTR:")) {
        buffer = 1;
      }

      ruleStorage napStruct = new ruleStorage(Integer
            .parseInt(tempArr[0 + buffer]), Integer
            .parseInt(tempArr[1 + buffer]),
            tempArr[2 + buffer], tempArr[3 + buffer],
            tempArr[4 + buffer], tempArr[5 + buffer]);
      ruleSet[i] = napStruct;
      numNaptrs++;
    }
  }

  /* We didn't get any records, return failed */
  else {
    System.err.println("Error: No DNS Records Returned");
    System.exit(1);
  }
} catch (Exception e) {
  System.err.println("Error!: " + e);
  System.exit(1);
}

/* sort array of results by order then preference */
for (int z = (numNaptrs - 1); z >= 0; z--) {
```

```
      for (int j = 1; j <= z; j++) {
        /* If we're more important than the next up, transfer */
        if (ruleSet[j - 1].order > ruleSet[j].order) {
          ruleStorage tempOrder;
          tempOrder = ruleSet[j - 1];
          ruleSet[j - 1] = ruleSet[j];
          ruleSet[j] = tempOrder;
        }
        /* If we're more important AND a higher preference, transfer */
        else if (ruleSet[j - 1].order == ruleSet[j].order
            && ruleSet[j - 1].preference > ruleSet[j].preference) {
          ruleStorage tempPref;
          tempPref = ruleSet[j - 1];
          ruleSet[j - 1] = ruleSet[j];
          ruleSet[j] = tempPref;
        }
      }
    }

    /* Copy working array to correct length array of ruleStorage items */
    ruleStorage[] ruleSetTidy = new ruleStorage[numNaptrs];
    System.arraycopy(ruleSet, 0, ruleSetTidy, 0, numNaptrs);

    /* Return ruleStorage array */
    return ruleSetTidy;
  }

  /*
   * Method checks keys are unique
   * @param Vector of seen keys and current key
   * @return Returns True is key is unique, false if key has been seen.
   */
  private static boolean keyUnique(Vector v, String key) {

    boolean found = true;
    Iterator vI = v.iterator();

    while (vI.hasNext()) {

      if (vI.next() == key) {
        found = false;
      }
    }
    return found;
  }

  /*
   * Method sets debug flag for verbose output
   */
  public void setDebug() {
    debug = true;
  }

}
```

References

Author Not Available (2005) **PARADISEC information leaflet.** [on-line] Available WWW: http://www.paradisec.org.au/Paradisec_PR04.pdf

Arends, A., Austein, R., Larson, M., Massey, D., & Rose, S. (2005) **DNS security introduction and requirements.** [on-line] Available WWW: http://www.ietf.org/rfc/rfc4033.txt

Berners-Lee T. (1998) **Uniform resource identifiers (URI), generic syntax.** [on-line] Available WWW: http://www.ietf.org/rfc/rfc2396.txt

CENDI. (2004) **Persistent identification: A key component of an e-government infrastructure.** [on-line] Available WWW: http://cendi.dtic.mil/publications/04-2persist_id.html

Connolly, D. & Berners-Lee, T. (1993) **Web naming and addressing overview.** [on-line] Available WWW: http://www.w3c.org/Addressing/

Esibov, L., Gulbrandsen, A., & Vixie, P. (2000) **A DNS RR for specifying the location of services.** [on-line] Available WWW: http://www.ietf.org/rfc/rfc2782.txt

Falstrom, P. & Huston, G. (2004) **A survey of internet identifiers.** [on-line] Available WWW: http://www.ietfreport.isoc.org/idref/draft-iab-identifies/

Fielding, R., Berners-Lee, T., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., & Leach, P. (1999) **Hypertext transfer protocol - HTTP/1.1.** [on-line] Available WWW: http://www.w3.org/Protocols/rfc2616/

Hakala, J. & Walravens, H. (2001) **Using international standard book numbers as uniform resource names.** [on-line] Available WWW: `http://www.ietf.org/rfc/rfc3187.txt`

Kunze, J. (2003) **Towards electronic persistence using ARK identifiers.** [on-line] Available WWW: `http://dot.ucop.edu/home/jak/arkcdl.pdf`

Markheim, S. (2004) **The defence virtual library.** [on-line] Available WWW: `http://dvl.dtic.mil/`

Mealling, M. (1999) **URI resolution services necessary for URN resolution.** [on-line] Available WWW: `http://www.faqs.org/rfcs/rfc2483.html`

Mealling, M. (2002) **The dynamic delegation discovery system.** [on-line] Available WWW: `http://www.ietf.org/rfc/rfc3305.txt`

Mealling, M. & Denenberg, R. (2002) **URNs: Clarifications and reccomendations.** [on-line] Available WWW: `http://www.ietf.org/rfc/rfc3305.txt`

Menezes, A., Oorschot, P., & Vanstone, S. (1996) **Handbook of Applied Cryptography.** CRC Press; USA.

Moats, R. (1997) **URN syntax.** [on-line] Available WWW: `http://www.ietf.org/rfc/rfc2141`

Hewlett Packard (2003) **DSpace: Preserving digital data for the ages.** [on-line] Available WWW: `http://www.hpl.hp.com/news/2003/july_sept/dspace.html`

Paskin, N. (2004) **The digital object identifier.** [on-line] Available WWW: `http://www.doi.org/overview/sys_overview_021601.html`

Shafer, K., Weibel, S., Jul, E., & Fausey, J. (N/A) **Introduction to persistent uniform resource locators.** [on-line] Available WWW: `http://purl.oclc.`

org/docs/inet96.html

Sollins, K. (1998) **Architectural principles of uniform resource name resolution.** [on-line] Available WWW: http://www.cis.ohiostate.edu/htbin/rfc/rfc2276.html

Sollins, K. & Masinter, L. (1994) **Functional requirements for uniform resource names.** [on-line] Available WWW: http://www.ietf.org/rfc/rfc1737.txt

Sun, S., Lannom, L., & Boesch, B. (2003) **Handle system overview.** [on-line] Available WWW: http://www.handle.net/rfc/rfc3650.html

Yin, R & Campbell, D (2002) **Case Study Research: Design and Methods** Sage Publications; USA.