The Warwick Framework
A Container Architecture for Aggregating Sets of Metadata

Carl Lagoze[*] **Clifford A. Lynch**[†] **Ron Daniel, Jr.**[‡]

June 28, 1996

1. Executive Summary

This document describes an architecture called the **Warwick Framework**, a result of the April 1996 Metadata Workshop II in Warwick U.K. The purpose of the Warwick workshop was to build on the results of the March 1995 Metadata Workshop in Dublin, Ohio, from which developed the **Dublin Core** metadata set.

The Dublin Core is an attempt to formulate a simple yet usable set of metadata elements to describe the essential features of networked documents -- what the report of the Dublin meeting terms "document-like objects". The focus of the Dublin Core is primarily, but not exclusively, on description of objects. The Core metadata set is intended to be suitable for use by resource discovery tools on the Internet, such as the "webcrawlers" employed by popular World Wide Web search engines (*e.g.*, Lycos and Alta Vista). The thirteen elements of the Dublin Core include familiar descriptive data such as *author*, *title*, and *subject*. In the design of the Dublin Core consideration was given to mappings between the elements of the Core and existing, more specialized descriptive systems such as library cataloging (AACR2/MARC) and the FGDC metadata scheme. A few fields in the Core, such as *coverage* and *relationship*, are less typical of descriptive cataloging, but attempt to generalize aspects of descriptive cataloging practice without making the fine distinctions called for by traditional approaches, which require trained catalogers.

The Warwick Workshop was convened to build on the Dublin core program and provide a more concrete and operationally useable formulation of the Dublin Core, in order to promote greater interoperability among content providers, content catalogers and indexers, and automated resource discovery and description systems. The second workshop also was an opportunity to assess the results of a year of experimentation with the Dublin Core.

While there was general consensus among the attendees that the concept of a simple descriptive metadata set is useful, there were a number of fundamental questions concerning the real utility of the Dublin Core as it was defined at the end of the preceding workshop. Does the very loosely defined Dublin Core really qualify as a "standard" that can be read and processed programmatically? Should the number of the core elements be expanded, to increase semantic richness, or reduced, to improve ease-of-use by authors and/or web publishers? Will authors reliably attach core descriptive metadata elements to their content? Should a core metadata set be restricted to only descriptive cataloging information or should it include other types of metadata such as administrative information, linkage data, and the like? What is the relationship of the Dublin core to other developing work in metadata schemes, particularly in those areas such as rights management information (terms and conditions), which were considered largely outside the scope of the Dublin Core?

---

[*] Digital Library Research Group, Computer Science Department, Cornell University, lagoze@cs.cornell.edu
[†] Office of the President, Division of Library Automation, University of California, clifford.lynch@ucop.edu
[‡] Advanced Computing Lab, Los Alamos National Laboratory, rdaniel@acl.lanl.gov

We concluded that the answer to these questions and the route to progress on the metadata issue lay in our ability to provide a higher-level context for the Dublin Core. This context should define how the Core can be combined with other sets of metadata in a manner that addresses the individual integrity, distinct audiences, and separate realms of responsibility and management that characterize these distinct metadata sets.

The result of the Warwick Workshop is a proposal for a *container* architecture, known as the Warwick Framework. The framework is a mechanism for aggregating logically, and perhaps concretely (through the use of specific data structures), distinct *packages* of metadata. This is a modularization of the metadata issue with a number of notable characteristics.

- It allows the designers of individual metadata sets to focus on their specific requirements and to work within their specific areas of expertise, without concerns for generalization to ultimately unbounded scope.
- It allows the syntax of metadata sets to vary in conformance with semantic requirements, community practices, and functional (processing) requirements for the kind of metadata in question.
- It distributes management of and responsibility for specific metadata sets among their respective "communities of expertise".
- It promotes interoperability and extensibility by allowing tools and agents to selectively access and manipulate individual packages and ignore others.
- It permits access to different metadata sets related to the same object to be separately controlled.
- It flexibly accommodates future metadata sets by not requiring changes to existing sets or the programs that make use of them.

The separation of metadata sets into packages does not imply that packages are completely semantically distinct. In fact, it is a feature of the Warwick Framework that an individual container may hold packages, each managed and maintained by distinct parties, with complex semantic overlap, recognizing the reality of these scoping problems.

2. Organization of this Paper

The remainder of this paper is divided into six major sections. Section 3 summarizes the two metadata workshops relevant to the Warwick Framework; the 1995 Dublin Ohio workshop and the 1996 Warwick U.K. workshop. Section 4 contains a short description of the Dublin Core, and then Section 5 evaluates the Core as defined by the first workshop and notes a number of its limitations. Section 6 sets the context for the remainder of the paper by considering the metadata issues in a broader context. Section 7 is the heart of this paper, containing a description of the Warwick Framework architecture. Section 8 describes a number of open issues which need to be resolved before full implementation of the core. Finally, Section 9 describes four proposed implementations: HTML, MIME, SGML, and one based on a distributed object architecture.

3. The Metadata Workshops

The idea for metadata workshops grew out of an informal meeting at the Second International World Wide Web Conference in Chicago in 1994. The participants at this meeting recognized that the number of resources on the Web was exploding and the ability to find or discover resources was becoming correspondingly complex.

The problems that existed two years ago with resource discovery and location on the Web still plague us. Web documents have no descriptive data, or metadata, associated with them and resource discovery tools, such as Lycos and Alta Vista, have no alternative than to create indexes from the contents of the documents. At best, these tools apply heuristics based on the use of HTML presentation markup as a method of moving beyond pure full-text based indexing. The problems with these full text indexes are apparent when one searches for documents about "Mercury" and find a mixture of pages about the planet Mercury, the element Mercury, the Greek God Mercury, and articles from the *San Jose Mercury-News*. Further, it is clear that the full text indexing approach does not extent reasonably to the growing collection of non-textual data moving onto the network. The full text based indexing approaches are completely useless for non-textual documents such as the plethora of images, audio, video, and even executable programs (accessible through CGI gateways) on today's Web.

While there have been some attempts to apply intellectual classification and cataloging practices to some resources on the net, such as the Yahoo catalog, the costs of employing human labor to construct these catalogs means that they can only provide a guide to a very small part of the rich resources accessible through the network; most commonly they operate on the level of entire web sites rather than at the document level. And the time delay inherent in human review and classification means that the cataloging approach is primarily applicable to relatively stable sites; it cannot be effectively and affordably extended to cover the large amount of relatively ephemeral material on the net.

The first Metadata Workshop, sponsored by the Online Computer Library Center (OCLC) and the National Center for Supercomputer Applications (NCSA), was held in March, 1995 in Dublin, Ohio. The Dublin Workshop convened a mixture of computer and information scientists, librarians, and information providers to attempt to formulate solutions to the growing resource discovery problem.

The goals of the Dublin Workshop were purposely quite modest. The decision was made to set aside the “general purpose” metadata problem, and limit the scope to metadata useful in *describing* what were termed "document-like objects" (DLOs), which correspond generally to a textual WWW page. The scope of the descriptive data was further restricted to what is commonly known as descriptive cataloging and, to a lesser extent, classification -- for example, little consideration was given to evaluative information which might reasonably be part of an object’s description. Furthermore, the focus of the workshop was on conceptual consensus building, to which syntax wars are especially anathema. By agreeing not to argue over syntax, the Workshop participants were limited to enumerating the set of metadata elements that are elemental to the description of DLOs and defining the semantics of those elements.

The primary result of this Dublin Workshop was a set of thirteen metadata elements, named the Dublin Core Metadata Set[1], or Dublin Core. The Dublin Core is intended as a relatively simple set of descriptive data that can be provided by a majority of authors or maintainers of WWW documents, and which will be usable by WWW locator services.

The results of the Dublin Workshop met with a considerable level of interest. The Workshop Report and summary article[2] in d-Lib Magazine[3] attracted substantial readership. A number of prototype implementations using the Dublin Core were undertaken in North America, Europe, and Australia.

To build on these results of the Dublin Workshop, a follow on workshop was held in Warwick U.K. in April, 1996, this time sponsored by OCLC and the United Kingdom Office for Library and Information Networking (UKOLN). The announcement to this second workshop contained two stated goals:

- promote semantic interoperability across disciplines and languages, and
- define mechanisms for extensibility to support richer descriptions and linkages to other description models.

As the participants convened for this second workshop there was consensus that these goals would be more difficult to meet than the conceptual consensus-building goal of the earlier workshop. Interoperability among systems requires a more concrete description of syntax and semantics than that provided by the original Dublin Core. Extensibility entails determining how the Dublin Core interacts with other types of metadata. An examination of interoperability and extensibility issues also involves a detailed consideration of usage scenarios for Dublin Core (and other metadata): how it is created, how it is located and interchanged, and how it is employed by various applications.

4. Summary of the Dublin Core

Readers of the document who are interested in complete details on the Dublin Core are encouraged to read the complete Dublin Workshop Report[4]. This section is intended as a brief summary of the Dublin Core metadata set -- as defined by the Dublin Metadata Workshop -- providing context for the remainder of this document.

Readers should note that there is a terminology problem in talking about the Dublin Core. Part of the work of the Warwick workshop was to reassess the Dublin Core per se, as well as to examine it in the larger context described here. Other parts of the overall workshop report will discuss some changes and refinements to the definition of the Dublin Core as a result of the Warwick meeting; further, some of the open issues covered later in this paper will point towards additional areas of potential change within the Core definition. The focus of this section and the next is to describe and evaluate the state of the Dublin Core as it existed prior to the start of the Warwick meeting.

The goal of the Dublin Core is to provide a minimal set of descriptive elements that facilitate the description and the automated indexing of document-like networked objects. In addition, the core is meant to be sufficiently simple to be understood and used by the wide range of authors and casual publishers who contribute information to the Internet. As stated earlier, the original Dublin Core proposal purposely avoids the question of syntax.

The thirteen elements of the Dublin Core are shown in Figure 1.

Subject	The topic addressed by the object.
Title	The name of the object.
Author	The person(s) primarily responsible for the intellectual content of the object.
Publisher	The agent or agency responsible for making the object available.
OtherAgent	The person(s), such as editors and transcribers, who have made other significant intellectual contributions to the work.
Date	The date of publication.
ObjectType	The genre of the object, such as novel, poem, or dictionary.
Form	The data representation of the object, such as PostScript file.
Identifier	String or number used to uniquely identify the object.
Relation	Relationship to other objects.
Source	Objects, either print or electronic, from which this object is derived.
Language	Language of the intellectual content.
Coverage	The spatial locations and temporal duration characteristic of the object.

Figure 1 - Dublin Core fields

In addition to enumerating these data elements, the Dublin Workshop report specifies a number of underlying principles that apply to the entire core metadata set.

- The core metadata set is extensible to permit inclusion of additional site specific or domain specific data elements .
- All elements in the Core metadata set are optional in any specific description of an object using the Core. The reason for this is two-fold. First, some of the elements of the Core are meaningful for only certain documents – for example, the *coverage* field is useful mainly for geospatial resources. Second, incomplete descriptions are inevitable under precisely those usage scenarios that the Dublin Core was intended to address; description of documents by authors, site

managers, and non-professional publishers or information distributors, rather than professional catalogers and indexers.

- All elements are repeatable allowing, for example, multiple author elements.
- The semantics of each element can be modified by either:
    - the use of qualifiers, borrowed from other existing metadata schemes, which permit more detailed or specific semantics to be imported into the Dublin Core from these pre-existing external metadata schemes, or
    - ad-hoc specializations and extensions developed specifically for use with the Core so as to refine the normal meanings of the core data elements.

  No consideration was given to registry of these extensions in the original Dublin Core document, or to their implications for interoperability and extensibility -- or indeed to the utility of the qualifier provision in the context of typical usage scenarios.

5. Evaluating the Dublin Core

The results of the 1995 Dublin Metadata Workshop were intended as an initial step towards defining a core descriptive metadata set, not as a complete definition of that set. Making progress at the Warwick Workshop required evaluating the Dublin Core definition and determining the work that was necessary to provide a platform for implementation. We summarize that evaluation in this section. Some of the issues described here are addressed by the Warwick Framework. Other issues, such as syntax for Dublin Core elements and guidelines for authors, are addressed by other documents resulting from the Warwick Workshop. A few remain for further work.

It is important to recognize that many of these criticisms are in some real sense unfair. The Dublin workshop deliberately set goals that were not intended to produce a fully defined final product but rather to make progress and build consensus towards the ultimate definition of a final product.

5.1 The Dublin Core is loosely defined.

The authors of the Dublin Core readily admit that the definition is extremely loose. With no definition of syntax and the principles that "everything is optional, everything is extensible, everything is modifiable" the Dublin Core definition does not even approach the requirements of a standard for interoperability. The specification provides no guidance for system designers and implementers of web crawlers and spiders that may use the Dublin Core as the source for resource discovery and indexing. Achieving this level of precision and concreteness was beyond the scope of the Dublin workshop but is essential for further progress.

5.2 Authors and publishers of networked material may not provide the Core information.

The simplicity of the Dublin Core was motivated by the desire to make it useable by the general class of "non-professional" authors and publishers so common on the World Wide Web. Yet, there is some empirical evidence that this class of providers will not even provide the simplest of descriptive information. Furthermore, with only very vague semantics, no controlled vocabulary and no definition of appropriate syntax within the fields of the Dublin Core there is the possibility that any information provided will be questionable or even meaningless; at the very least it may be impossible to process algorithmically. The Dublin Core data elements in effect serve as a transport mechanism and a labeling mechanism for information that is going to be displayed to human end-users; or at best processed heuristically, with some consideration given to the Dublin Core tagging in developing the heuristics.

It is important to recognize here that this is less a critique of the Dublin Core per se than it is a recognition of the problems in developing implementation scenarios that include author-provided descriptive metadata elements. Facilitating the creation of such information as part of authoring and network “publishing” is a key to making progress. A major focus at the Warwick meeting was the development of practical mechanisms to permit authors and site administrators to provide Dublin Core data elements.

5.3 The Dublin Core represents the dominance of descriptive cataloging, avoiding issues of operational and administrative metadata.

The Dublin Core does not represent itself as any more than a method for abbreviated descriptive cataloging. However, it is important to note, and we will return to this subject later, that descriptive cataloging is just one set among the many and varied metadata sets that may be associated with networked objects. Many of these other metadata sets, such as those associated with the management and administration of the object, have equal importance and need to be specified. It is important to note that in the discussion of implementation experience at the Warwick meeting, virtually all implementations found it necessary to work with other types of metadata in addition to descriptive cataloging, and found it expedient to develop packaging, transport and organizational strategies that allowed metadata of different types to be handled consistently.

5.4 While the Dublin Core concentrates on general descriptive cataloging elements, it also includes some domain-specific elements.

Again, in the interest of reaching consensus, the creators of the Dublin Core included a number of elements that diverge notably from the practices of general purpose descriptive cataloging. For example, the *coverage* element is specific to spatial or temporal data and the *source* element is generally meaningful for materials that did not originate in digital form. The *relationship* element has particularly problematic and unclear semantics that are extremely domain dependent. One could argue that other specialized elements should be included in the Core, threatening to explode the definition with all sorts of new metadata.

5.5 But the Dublin Core is an important step toward developing consensus on metadata.

Although many of these criticisms are valid, the Dublin Core is, at the least, a firm basis for further discussion in an important area. Full library-style descriptive cataloging is simply too expensive for the overwhelming majority of networked documents. The need for a cheaper alternative is apparent. The Dublin Core at least addresses this issue and the effort to create it was valuable as a first step in a longer process.

In addition, the use of the Dublin Core in a limited context might produce very positive results. For example, assume a set of "high-integrity sites". Administrators at such sites might tag their documents at these sites with Dublin Core metadata elements using a set of well-specified practices that included relatively controlled vocabulary and regular syntax. Retrieval effectiveness across these high-integrity sites would probably be significantly better (assuming harvesting and retrieval tools that make use of the metadata) than the unstructured searches available now through Lycos and Alta Vista. We can even imagine a market structure where these sites register with the search service providers, for a fee, and that users could restrict their searches to these sites.

6. Metadata Issue in the Broader Context

The organizers of the 1995 Dublin Metadata Workshop intentionally limited its scope, avoiding, as the workshop report states, "the size and complexity of the resource description problem". This limited focus was viewed as necessary to produce concrete results from the Workshop.

By the end of the first day of the Warwick Workshop, it was obvious that this strategy was now an impediment to progress beyond what had been previously accomplished. Three principal questions surfaced, each of which made clear the need to broaden our perspective.

1. *Should the number of elements in the Dublin Core be expanded or contracted?* Some workshop attendees felt that in order for the core to succeed as a tool for authors, its number of elements

should be restricted to only the most basic descriptive elements. Others saw the need for new fields such as *terms and conditions* or *administrator.*

2. *Should the syntax of the Core be strictly defined or left unstructured?* Many attendees wanted to avoid the painful syntax wars that are familiar to those who have participated in standards efforts. However, without a more concrete definition of syntax, the Dublin Core does not provide the level of interoperability for which it was intended.
3. *Should the Core be targeted solely at the existing WWW architecture, or extend that architecture?* There is a strong argument to facilitate easy adoption by specifying a metadata standard that is closely attuned to current practice in the Web environment and that can be easily implemented within the existing World Wide Web framework (browsers, servers, HTML specification, etc.). However, the Web is clearly not the ultimate networked information delivery vehicle , and many of its flaws are the subject of active discussion in the IETF, W3C, and other venues. Many of the Workshop attendees felt that it was important to describe a metadata framework that extends existing WWW technology and is sufficiently flexible to accommodate older but still important networked information models such as FTP archives on one hand, but can also extend to new information delivery environments such as distributed object systems on the other.

We can answer these questions by stepping back from our focus on core metadata elements and examining some of the general principals about metadata.

6.1 Metadata takes a variety of forms, both specialized and general.

Descriptive cataloging is but one of many classes of metadata. Yet, even if we restrict ourselves to this category, we observe that there exists and is legitimate reason for a variety of cataloging methodologies and interchange formats.

The well-established Anglo-American cataloging rules (AACR2) and MARC[5] interchange format (and its numerous variations) is the basis for virtually all existing library systems and has proven effective in creating and encoding descriptions of a great variety of content . However, the very complex rules require extensively trained catalogers for successful application, and the arcane structure of the MARC record requires complex and specialized systems for record creation and interchange. Simpler descriptive rules, such as that suggested by the Dublin Core, are usable by the majority of authors but do not offer the degree or retrieval precision and classification and organization that characterizes library cataloging. Projects such as the Computer Science Technical Reports[6] effort have already proven that simple descriptive rules, when coupled with simple interchange formats that can be created with commonplace text editors, can permit untrained authors or editorial staff to build descriptive records that are of significant value. The Dublin Core builds on this experience. Finally, there are domain-specific formats such as the Content Standard for Digital Geospatial Metadata[7] (CSDGM) format that is the result of work by the Federal Geographic Data Committee[8] (FGDC); the schemes used to describe mathematical software packages; or the schemes used in the health sciences for classification and description. These schemes include considerable descriptive cataloging, and are if anything more complex than MARC/AACR2, offering very precise descriptions of very complex datasets for specific user communities and specific software environments (both for record creation and searching) that can support them.

Real world applications, however, need to make use of a much broader spectrum of metadata than descriptive cataloging. We list below some other metadata types to provide a sense of this range, without suggesting that this list is in any way a comprehensive enumeration

- *terms and conditions* - This is metadata that describes the “rules” for use of an object. Terms and conditions might include an access list of who can view the object, a "conditions of use" statement that might be displayed before access to the object is allowed, a schedule (tariff) of prices and fees for use of the object, or a definition of permitted uses of an object (viewing, printing, copying, etc.).

- *administrative data* **-** This is metadata that relates to the management of an object in a particular server or repository. Some examples of information stored in administrative data is date of last modification, date of creation, and the administrator's identity.
- *content ratings* **-** This is a description of attributes of an object within a multidimensional scaled rating scheme as assigned by some rating authority; an example might be the suitability of the content for various audiences, similar to the well-know movie rating systems used by the MPAA. The technical subcommittee of PICS[9] (Platform for Internet Content Selection) in the IETF is an effort to create a framework for defining such content ratings. Note that content ratings have applications far beyond simple filtering on sex and violence levels, however; they are likely to play important roles in future collaborative filtering systems, for example.
- *provenance* **-** This is data defining source or origin of some content object, for example describing some physical artifact from which the content was scanned. It might also include a summary of all algorithmic transformations that have been applied to the object (filtering, decimation, etc.) since its creation. Arguably, provenance information might also include evidence of authenticity and integrity through the use of digital signature schemes; or such authenticity and integrity information might be considered a separate class of metadata.
- *linkage or relationship data* **-** Content objects frequently have multiple complex relationships to other objects. Some examples are the relationship of a set of journal articles to the containing journal, the relationship of a translation to the work in its original language, the relationship of a subsequent edition to the original work, or the relationships among the components of a multimedia work (including synchronization information between images and a soundtrack, for example). References to related content should be done using some unique persistent identifier such as an ISBN, ISSN, or URN.
- *structural data* **-** This is data defining the logical components of complex or compound objects and how to access those components. A simple example is a table of contents for a textual document. A more complex example is the definition of the different source files, subroutines, data definitions in a software suite.

It should be clear from these examples that we are far from consensus on an overall taxonomy of metadata classes, although considerable work has been done in this area. Consider the murky relationships between structural metadata on one hand and linkage or relationship data on the other, or the questions raised above about the scope of provenance metadata.

6.2 New metadata sets will develop as the networked information infrastructure matures.

The range of metadata needed to describe and manage objects is likely to continue to expand as we become more sophisticated in the ways in which we characterize and retrieve objects and also more demanding in our requirements to control the use of networked information objects. Until the present era of the multimedia WWW and ubiquitous Internet access (*i.e.,* children can access it from the home), who would have imagined the need for content ratings? As another example, as more proprietary content is made available for purchase on the global Internet, there is a growing emphasis on areas such as terms and conditions definitions as a prerequisite to establishing a networked marketplace in information. The architecture must be sufficiently flexible to incorporate new classes of semantics without requiring a rewrite of existing sets (imagine having to go through the net to do a metadata rewrite!!).

6.3 Different communities will propose, design, and be responsible for different types of metadata.

Each logically distinct metadata set may represent the interests of and domain of expertise of a specific community. For example, elaborate descriptive cataloging sets are best created and maintained by librarians and especially catalogers. The contents of terms and conditions metadata sets are best understood by parties with business and legal expertise and background in intellectual property issues. Each community should be able to independently create and maintain the metadata that falls within its "sphere of influence". Some classes of metadata may exist to meet specific legal or regulatory requirements; assertions made within these metadata segments may have particular legal liabilities associated with them, for example.

The separate origin and administration of different metadata sets will result in very divergent syntax and notation. For some types of metadata, such as descriptive cataloging data, static textual representations will be sufficient. Others may be expressible only through more powerful means, such as executable (or interpretable) programs. This is especially true for the metadata that may encode terms and conditions, which may specify negotiation between clients, agents, and outside services (*e.g.,* authentication services and payment services).

6.4 There are many "users" of metadata.

Just as there are disparate sources of metadata, different metadata sets are used by and may be restricted to distinct communities of users and agents. Machine readability may be a high priority for some types of metadata, whereas for others may be targeted for human readability. The terminology in some types of metadata may be domain specific. Each "user" of metadata should be able to directly access that metadata that is relevant to it. From the opposite perspective, there may be reason to selectively restrict access to certain types of metadata associated with an object to certain communities of users or agents. There will be requirements to be able to transport specific metadata sets across systems without permitting those systems access to the content of the metadata set.

6.5 Metadata and data have similar behaviors and characteristics.

Strictly and statically partitioning the information universe into data and metadata is misleading. What may appear to be metadata in one context, may look very much like data in another. For example, some critic's review of a movie qualifies as metadata **-** it is a description of the content, the movie. However, the review itself is intellectual content that can stand alone as data in many instances. Like other data it may have associated metadata and, especially, terms and conditions that protect it as fungible intellectual property. This recursive relationship of data and metadata may nest to an arbitrarily deep level.

6.6 The metadata sets associated with an object may be physically collocated or may be referenced indirectly.

If we are really envisioning a *distributed* information infrastructure, then our notion of distribution should not only be *between* objects, but *within* objects. That is, metadata for an object may be a aggregation of multiple independently managed sets of metadata, each of which may be maintained separately on the network. References to physically separate sets should be referenced using a reliable persistent name scheme, such as that proposed for Universal Resource Names[10] (URNs) and Handles[11].

Indirect reference to metadata sets goes hand-in-hand with sharing of metadata sets. For example, assume a repository with many content objects, a number of which have common terms and conditions for access (*e.g.,* a university digital library with a site license for a set of periodicals). We should be able to express this by linking, using a name reference, the set of objects to one encoding of the terms and conditions. Consequently, we should be able to modify the terms and conditions for the set of objects by changing the one shared encoding. The shared terms and conditions metadata may, in turn, reside separately in a repository managed by an outside provider that specializes in intellectual property management

7. The Warwick Framework Architecture

The result of this analysis at the Warwick Workshop is an architecture, the Warwick Framework, for aggregating multiple sets of metadata. The Warwick Framework has two fundamental components. A *container* is the unit for aggregating the typed metadata sets, which are known as *packages*.

A container may be either transient or persistent. In its transient form, it exists as a transport object between and among repositories, clients, and agents. In its persistent form, it exists as a first-class object in the information infrastructure. That is, it is stored on one or more servers and is accessible from these servers using a globally accessible identifier (URI). We note (and will demonstrate later in the proposed distributed object implementation) that a container may also be wrapped within another object (*i.e.*, one that is a wrapper for both data and metadata). In this case the “wrapper” object will have a URI rather than, or in addition to, the metadata container itself.

Independent of the implementation, the only operation defined for a container is one that returns a sequence of packages in the container. There is no provision in this operation for ordering the members of this sequence and thus no way for a client to assume that one package is more significant or "better" than another.

At the container level each package is an opaque bit stream. One implication of these properties is that any encoding (transfer syntax) for a container must allow the recipient of the container to skip over unknown packages within the container (in other words, the size of each package must be self describing at the container level). This property also permits the contents of individual packages to be encrypted, permitting the transport of metadata across systems that need not have access to specific sets, or that may need to acquire (*i.e.*, purchase) such access. Certain implementations, such as the HTML one proposed later in this paper, may lack the power to fully enforce these abstract properties of containers.

Each package is a typed object; its type may be determined after access by a client or agent. Packages are of three types:

1. *metadata set* - These are packages that contain actual metadata. Some examples of this are packages that are MARC records, Dublin Core records, and encoded terms and conditions. A potential problem is the ability of clients and agents to recognize and process the semantics of the many metadata sets. In addition, clients and agents will need to adapt to new metadata types as they are introduced, at least to the extent of ignoring them gracefully, or perhaps copying them for downstream applications that may know how to process them. Initial implementations of the Warwick Framework will probably include a set of well known metadata sets, in the same manner that most Web browsers have native handlers for a set of well-known MIME types. Extending the Framework implementations to handle an extensible metadata sets will rely on a type registry scheme. We describe this in some greater detail in the implementation section of this document.
2. *indirect* - This is a package that is an indirect reference to another object in the information infrastructure. While the indirection could be done using URLs we emphasize that the existence of a reliable URN implementation is a necessary to avoid the problems of dangling references that plague the Web. We note three possibly obvious, but important, points about this indirection. First, the target of the indirect package is a first-class object, and thus may have its own metadata and, significantly, its own terms and conditions for access. Second, the target of the indirect package may also be indirectly referenced by other containers (*i.e.,* sharing of metadata objects). Finally, the target of the indirection may be in a different repository or server than the container that references it.
3. *container* - This is a package that is itself a container. There is no defined limit for this recursion.

We illustrate in Figure 2 a simple example of a Warwick Framework container. The container in this example contains three logical packages of metadata. The first two, a Dublin Core record and a MARC record, are contained within the container as a pair of packages . The third metadata set, which defines the terms and conditions for access to the content object, is referenced indirectly via a URI in the container. Note that the syntax for terms and conditions metadata and administrative metadata is not yet defined.

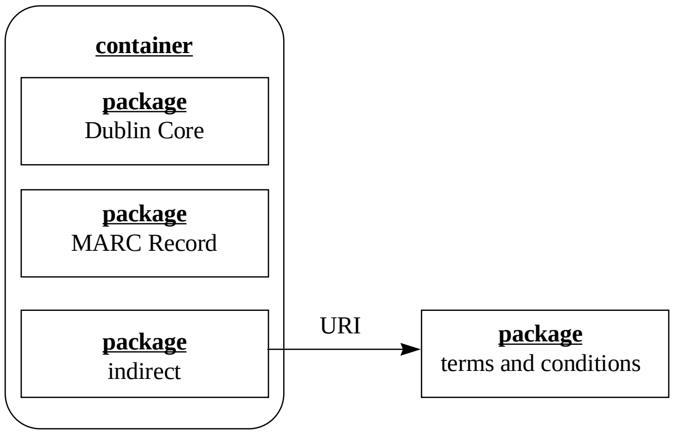


Figure 2 - Metadata container with three packages (one indirect)

The mechanisms for associating a Warwick Framework container with a content object (*i.e.,* a document) depend on the implementation of the Framework. The proposed implementations later in this document illustrate some of the options. For example, a simple Warwick Framework container may be embedded in a document, as demonstrated in the HTML implementation proposal; or an HTML document can link to a container that is stored as a separate file. On the other hand, as demonstrated in the distributed object proposal, a container may be a logical component of a so-called digital object, which is a data structure for representing networked objects.

The reverse linkage, that which ties a container to a piece of intellectual content, is also relevant. Anyone can, in fact, create descriptive data for a networked resource, without permission or knowledge of the owner or manager of that resource. This metadata is fundamentally different from that metadata that the owner of a resource chooses to link or embed with the resource. We, therefore, informally distinguish between two categories of metadata containers, which both have the same implementation.

- An *internally-referenced* metadata container is the metadata that the author or maintainer of a content object has selected as the describing the object. This metadata is associated with the content by either embedding it as part of the structure that holds the content or referencing it via a URI. An internally-referenced metadata container referenced via a URI is, by nature, a first-class networked object, and may have its own metadata container associated with it. In addition, an internally-referenced metadata container may back-reference the content that it describes via a *linkage* metadata element within the container.
- An *externally-referenced* metadata container is metadata that may well be created and maintained by an authority separate from the creator or maintainer of the content object. In fact, the creator of the object may not even be aware of this metadata. There may an unlimited number of such externally-referenced metadata containers. For example, libraries, indexing

services, ratings services, and the like may compose sets of metadata for content objects that exist on the net. As we stated earlier, these externally-referenced metadata containers are themselves first-class network objects, accessible through a URI and having some associated metadata. The linkage to the content that one of these externally-referenced containers purports to describe will be via a *linkage* metadata element within the container. There is no requirement, nor is it expected, that the content object will reference these externally-referenced containers in any way.

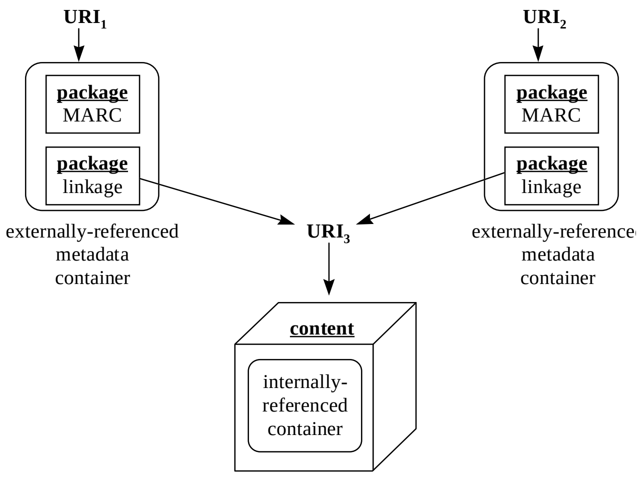

URI1
URI2
package MARC
package linkage
package MARC
package linkage
externally-referenced metadata container
URI3
externally-reference metadata container
content
internally-referenced container

Figure 3 - Relationship between content and metadata

Figure 3 shows an example of this relationship. Three metadata containers are shown. The one internally-referenced metadata container is embedded in the content object (it does not have a URI, nor does it have a linkage package that references the content). The two externally-referenced metadata containers are independent objects. They each have a URI and reference the content object via its URI.

The internally-referenced metadata container in this illustration could also be indirectly referenced by the content. In this case it would have its own URI (say URI_4) and would have a linkage package referencing URI_3 (the content).

8. Open Issues in the Warwick Framework

Time constraints at the Warwick workshop did not permit a full exploration of all of the issues involved in the proposed framework. There are several topics that urgently call for more detailed and extended examination prior to finalizing the framework. Certainly the most fundamental question about the Warwick Framework is the semantic interaction and overlap of the multiple metadata sets that may exist in a container. While packages are to some extent logically distinct, they may have semantics that overlap in complex ways.

This overlap may occur at numerous levels. There is the possibility of *horizontal* semantic overlap between two metadata sets in a container. One example is a container with two descriptive cataloging records; one MARC and one Dublin Core. There is the possibility of *vertical* semantic overlap between two metadata sets, one in a container and the other at some arbitrary level of recursion that descends from that container. (Recall that a container may hold other containers or refer indirectly to other containers.) One example is a complex object with multiple terms and conditions metadata sets as one descends through the structure of the object. Here the problem is the scope of the object or objects to which the metadata carried in packages within the composite object applies.

In the end, the semantics of the metadata associated with an object need to be understood by the "consumers" of metadata - the clients and agents that access objects and the users that configure these clients and agents. We run the danger, with the full expressiveness of the Warwick Framework, of creating such complexity that the metadata is effectively useless. Finding the appropriate balance is a central design problem.

Consider, for example, a common consumer of metadata - the "spider" or "crawler" that tries to gather descriptive metadata for networked objects and then compiles it into a searchable index. Designing this agent is difficult if descriptive cataloging metadata is contained, without concern for consistency, in a number of metadata sets for each object. What are the rules for assembling a usable index from such arbitrarily mixed metadata? What are the semantic transformations that can be made across the multiple metadata sets?

We see similar, if not more complex, problems with overlapping metadata describing access rules or terms and conditions for an object. Client access to a compound object, such as a multimedia document, may require negotiating through numerous sets of terms and conditions at multiple levels of the document (*e.g.*, one set for a particular section of text, another for a particular slice of full-motion video, etc.). Yet, at the top level, all the client is concerned with is whether the object can be accessed and at what cost. This may be difficult or impossible to compute, especially if we consider the possibility of arbitrarily deep recursion or circular references among containers. We assume that market forces, the strong incentive to make object's accessible to clients and agents, may be sufficient to avoid the most complex problems of this nature.

One proposal that was discussed only superficially at the Warwick workshop was the possibility of developing a relationship metadata package, which would externalize and generalize the relationship element from the Dublin Core. Presumably if this approach were adopted the relationship element would be removed from the revised Dublin Core, thus eliminating one source of ambiguity about compound objects. However, since many metadata schemes, not just the Dublin Core, contain concepts of relationship or hierarchy, this does not fully solve the problem.

In addition to this fundamental issue, we list below a number of other implementation issues that need to be examined in the process of implementing the Framework.

8.1 Type Registry

The Framework design requires that packages are strongly typed. By this we mean that an agent or client will be able to determine the type of the metadata in a package; definers of specific metadata sets should ensure that the set of operations and semantics of those operations will be strictly defined for a package of a given type. We expect that a limited set of metadata types will be widely used and that agents, browsers, and clients will be configured to process these metadata types. This is similar to capability of existing WWW browsers to internally process certain MIME types of documents.

The type system must be extensible for, as we state earlier, new sets of metadata or variations of existing sets will appear. How will existing clients and browsers handle these new types? The simplest, solution is that existing software will need to be upgraded as new metadata types come into existence; prior to such upgrading, the container structure would at least permit the software to skip over the new, unknown package type and warn the user that it had done so. Such a solution does not scale well if new types appear often. A better solution would be the development of a network-based, software queriable type registry service. This service could provide the information that would perhaps allow a client to either reconfigure itself to process the new type, or download a network accessible applet or helper application that would assist in processing the new type.

The limitations of such an approach are still real, however, and not well-understood: somehow, the type system would need to convey some sense of the meaning of new types to processing applications. This might allow an application to, at first, determine if it even needs to "care" whether it doesn't know about a new type. Depending on the answer to this, the application could decide whether it needs to download an extension or helper application. For example, current browsers don't care about provenance information, and many users may not understand or care about provenance.

There are also questions about whether there needs to be hierarchy and perhaps some form of inheritance in the type system: for example, do we want a general class of rights and permissions metadata that recognizes that there may be various “dialects” and transfer syntaxes and representations used for this class of metadata?

8.2 Data Encoding

The Warwick Framework presents two data encoding problems. At the container level, what is the syntax for transferring sets of packages? This syntax must be independent from the syntax of the packages themselves, which are opaque at this level (indeed the package itself may even be encrypted). It must allow for type information to be carried. Ideally, we believe it should be relatively simple. We include some proposals for this container syntax, where appropriate, in the following implementation sections.

The more difficult data encoding problems exist at the package level. Here there is no single right answer. Some metadata sets can be adequately expressed in ASCII as a set of attribute/value pairs. Other designers of metadata sets may prefer other structures such as SGML, HTML, or ASN.1. The fields contained in these syntaxes may be much more complex than just strings and integers as well. Consider, for example, rules that describe the terms and conditions for access to an object. In the simplest case these rules may be expressed using an access control list, such as that well established in the operating systems world. However, a complete representation of the current legal and business framework in the digital domain will require rules that account for negotiation, challenge/response, and interaction with third-party services (*e.g.*, payment services, authentication services, and the like). This type of adaptive, interactive metadata is best expressed via some type of executable program or agent.

Our approach of segmenting metadata into discrete packages each with independent syntaxes at least permits progress on this problem; simple classes of metadata can use simple syntaxes, while classes with more complex requirements can explore more complex and powerful syntaxes. But, to return to one of the central issues that motivated the Warwick Workshop, there is a need to agree on one or more syntaxes for the Dublin Core data elements themselves.

8.3 Efficiency

The power of the Warwick Framework lies in its recursive and distributed characteristics. This lends great power to the model, but in an actual implementation may be quite inefficient. Even in the context of the relatively simple World Wide Web, the Internet is often unbearably slow and unreliable. Connections often fail or time out due to high load, server failure, and the like. In a full implementation of the Warwick Framework, access to a "document" might require complex negotiation across distributed repositories. The performance of this distributed architecture is difficult to predict and is prone to multiple points of failure. Efficient operation of this distributed architecture will depend an improved network infrastructure using caching, data or object replication, dynamic load balancing, and other methods being examined in distributed systems research

8.4 Repository Access

It is clear that some protocol work will need to be done to support container and package interchange and retrieval. We foresee the need for various forms of retrieval. The simple case is retrieval of a container for an object. A more complex case is retrieval of only those containers that include packages of a specific set of types. The requirements for this protocol have not been explored in any detail. Some examination of the relationship between the Warwick Framework and ongoing work in repository architectures would likely be fruitful.

9. Implementing the Warwick Framework

Simplicity of design and rapid deployment were primary considerations in the design of the Dublin Core. At first glance it may seem that, with the Warwick Framework, we have forsaken this motivation and have proposed an architecture that does not fit with the current world of HTML, HTTP, and WWW browsers. In fact, the basic notion of the Framework, the ability to place a number of metadata sets in a container, can be expressed in the context of the existing WWW infrastructure. We propose this implementation later in this section.

We miss an important opportunity, however, if we constrain the design and possible implementations according to the existing Web. This infrastructure will surely evolve and may even be replaced by a more powerful information infrastructure. Research and development of such an infrastructure is being undertaken in the DARPA/NSF/NASA Joint Digital Library Initiative[12], other international digital library research projects, and a number of other venues. With an eye towards these developments, we later propose a number of other implementations, with greater power to fully express the Warwick Framework.

9.1 HTML Implementation

Rapid deployment of the Warwick Framework will only occur if the initial implementation requires no change to existing WWW software. We describe in this section a limited implementation of the Framework that conforms to HTML 2.0, and which is transparent to existing browsers, spiders, and HTML authoring tools. Eric Miller[13] of OCLC[14] proposed the initial idea for this implementation. This proposal was presented at the May 1996 W3C-sponsored Distributed Indexing/Searching Workshop[15]. The syntax described below extends the workshop proposal[16] with a provision for indirect reference to metadata sets.

This implementation takes advantage of two tags in HTML 2.0.

- The `META` tag is used to embed metadata with the `HEAD` portion of HTML documents.
- The `LINK` tag provides for both indirect linking to the reference definition for a metadata schema and for indirect linking to a set of metadata.

9.1.1 META Tag

Each `META` tag in HTML 2.0 specifies a name/value pair. This pair is encoded using the `NAME` attribute and the `CONTENT` attribute. The `HEAD` portion of an HTML document may contain multiple `META` elements. For example, a simple example of a `META` tag is:

```
    <META NAME="title" CONTENT="Gone With the Wind">
```

We propose an encoding for the value of the `NAME` attribute that groups a number of `META` tags into a single metadata set. The encoding is as follows:

```
    <META NAME="<schema_name>.<element_name>" CONTENT="string data">
```

In this encoding, the template `<schema_name>` is replaced by a unique prefix for the respective metadata set, and `<element_name>` is replaced by the attribute name in that metadata set. The registration of unique `<schema_name>` encodings is left unresolved at this time. If we assume the `<schema_name>` `DC` for Dublin Core elements, a partial Dublin Core metadata set would be encoded as follows:

```
    <META NAME="DC.Title" CONTENT="HTML 2.0 Specification">
    <META NAME="DC.Author" CONTENT="Tim Berners-Lee">
    <META NAME="DC.Author" CONTENT=Dan Connolly">
```

This same HTML document might contain another set of metadata with another `<schema_name>` prefix. For example, assume some standard for administrative metadata, which we will call the `ADM` set. The set might be coded as follows:

```
    <META NAME="ADM.Administrator" CONTENT="Bill Clinton">
    <META NAME="ADM.Modified_Date" CONTENT="050696">
```

9.1.2 LINK Tag

The following syntax for the `LINK` is used to indicate that a document containing metadata of the type `<schema_name>` is located at `<URL>`.

```
    <LINK REL=META.<schema_name> HREF="<URL>">
```

For example, an indirect link to a set of geospatial metadata, with the `<schema_name>` `CSDGM`, might be:

```
    <LINK REL="META.CSDGM" "HREF=http://meta_repo.ukp.edu/geometa">
```

Another use for the `LINK` tag is to provide a pointer to a human-readable reference definition of a metadata schema. This encoding is motivated by the lack of a central registry of metadata schema. The following syntax for the `LINK` tag indicates that the reference definition of the metadata scheme `<schema_name>` is located at `<URL>`.

```
    <LINK REL=SCHEMA.<schema_name> HREF="<URL>">
```

Thus, the reference definition for the Dublin Core metadata schema might be indicated by the following encoding:

```
    <LINK REL=SCHEMA.dc HREF="http://purl.org/metadata/dublin_core">
```

The combination of these elements in an HTML document is shown in Figure 4.

The HTML implementation of the framework offers backward compatibility with existing browsers, but requires a fair amount of work to determine the container/package structure. Fully implementing the framework in HTML is problematic because of the difficulty of handling nested containers in the simple name/value pairs provided by the META element.

There are at least two potential migration paths for browsers to implement the full capabilities of the framework - MIME and SGML. MIME (Multipurpose Internet Mail Extensions) is the set of Internet standards that were developed to allow email messages to contain rich data formats and structures. Browsers already have limited support for MIME, and their level of support is likely to increase over time. SGML (Standard Generalized Markup Language) is the meta-language that was used to define HTML. Browser add-on products already exist for browsing arbitrary SGML documents, and it is possible that browsers will grow to offer native SGML capabilities. The next two sections discuss how the Warwick Framework can be implemented in MIME and SGML. Following that is a section on implementing the framework using distributed objects, which assumes an information infrastructure far different from the existing WWW.

9.2 MIME Implementation

As mentioned above, MIME is the set of standards (RFC-1522 and others) that were originally created to allow varying content in e-mail messages. Earlier standards, such as RFC-822, dealt with the standardization of mail headers, but not with the structure of the mail message itself. MIME addresses the structure of the message, allowing messages to contain binary data and multiple components or attachments. These capabilities can be used for a straightforward implementation of the

```
<HTML>
<HEAD>
<TITLE>A Sample Document with Mixed Metadata</TITLE>
<META NAME="DC.Title" CONTENT="Sample Document">
<META NAME="DC.Author" CONTENT="Bob Dole">
<META NAME="DC.Author" Content=Bill Clinton">
<META NAME="ADM.Administrator" CONTENT="Bill Clinton">
<META NAME="ADM.Modified_Date" CONTENT="050696">
<LINK REL="SCHEMA.DC" HREF="http://meta.org/meta-reg/Dublin-
Core.html">
<LINK REL="META.CSDGM" HREF="http://meta_repo.ukp.edu/geometa">
</HEAD>

<BODY>
This is the body of the sample document
</BODY>

</HTML>
```

Figure 4 - HTML document with embedded metadata

container/package architecture of the Warwick Framework. This section provides a brief overview of MIME, then shows how the three types of packages in the Warwick Framework - container, indirect, and metadata set - may be represented using MIME constructs. A more detailed paper on implementing the Warwick Framework in MIME is being prepared by Jon Knight and Martin Tomlinson

9.2.1 MIME Overview

MIME defines a simple two-level typing scheme. Each MIME message specifies this type in the `content-type` field. There are seven major types - `text`, `audio`, `video`, `image`, `application`, `multipart`, and `message`. Each of those major types has an extensible set of subtypes. For example, the `text` type has subtypes such as `text/plain`, `text/sgml`, and `text/html`. The `image` type has subtypes such as `image/gif`, `image/jpeg`, and `image/tiff` to deal with different image formats.

The `multipart` type is used for messages that include multiple components, each with a possibly different type. The common example of a `multipart` message is an ASCII e-mail message with attachments such as spreadsheets, graphics, etc. Multipart messages are composed of *body parts*, whose boundaries are flagged by *boundary strings*. Each body part has an associated `content-type.` Multipart messages may contain nested multipart body parts.

There are a number of subtypes for the type `multipart`.

- `alternative` is intended for situations where only one of the alternative body parts should be presented to the user, such as when plain text and formatted versions of a document are sent. Applications that can deal with the formatting should show that alternative. Applications that can't handle the formatting have the `plaintext` as a fallback.
- `parallel` is intended for applications where the different body parts should all be shown.
- `mixed` is for situations that are not as simple as `alternative` or `parallel`.
- `related` is a more recent content type for situations where one body part needs to make references to another body part.

9.2.2 Encoding the Warwick Framework in MIME

The body parts of a MIME multipart message directly correspond to the packages in a Warwick Framework container. This allows a straightforward encoding of metadata packages as a MIME message, as shown in the example in Figure 5 The example shows a container with two packages, each of which is a *metadata set* as defined earlier. The container is typed as `multipart/alternative`, indicating that the two included packages have similar meaning. The text `boundary="######"` at the beginning of the container defines a unique boundary string to delimit each package of the multipart message. The first package is a Dublin Core description encoded in SGML, and the second is a MARC record with the type `application/usmarc`. Systems that could handle USMARC would use it in preference to the Dublin Core description, but the Dublin Core description is there as a usable alternative for simpler systems. Note that the USMARC alternative has a header for the **`Content-Transfer-Encoding`**. This allows special characters to be safely transmitted through mail systems that only handle 7-bit characters.

```
MIME-Version: 1.0
Content-type: multipart/alternative; boundary="######"

--######
Content-type: text/sgml

<!DOCTYPE dublinCore PUBLIC '-//OCLC//DTD Dublin core v.1//EN'>
<dublinCore>
  <title>On the Pulse of Morning</title>
  <author>Maya Angelou</author>
...
</dublinCore>
--######
Content-type: application/usmarc
Content-Transfer-Encoding: Quoted-Printable

... USMARC record appears in here, a quoted-printable
    encoding is used to handle special characters ...
--######
```

Figure 5 - Warwick Framework container encoded as MIME multipart message

The MIME content-type `message/external-body` can be used to implement the Warwick Framework *indirect package*, which defines a package that is external to the container. The example in Figure 6 shows an indirect reference to a MARC package using a URI. Note that the Internet Engineering Steering Group (IESG) has recently approved a draft standard that allows the use URLS as an `access-type` for external body parts.

```
Content-type: message/external-body; access-type=URI;
              name="http://www.foo.bar.com/path/huh.marc"
```

Figure 6 - Indirect reference to a MARC package using a URI

9.3 SGML Implementation

As mentioned above, SGML[17] [18] is the meta-language that is used to define HTML. By this, we mean that SGML is a language that is used to define other languages, typically ones for marking up textual documents. Those languages, such as HTML, are defined by preparing a Document Type Definition (DTD). DTDs are roughly analogous to the schema definitions in relational databases. They define the allowed structure and combinations of structures in a document. A detailed paper on implementing the Warwick Framework in SGML is being prepared by Lou Burnard, C.M. Sperberg-McQueen, Liam Quin, and Eric Miller. This section discusses one way of implementing the framework in SGML that was derived from an early draft of their paper.

Implementing the Warwick Framework in SGML requires a DTD that can handle the container/package architecture, and can deal with indirect packages and metadata sets. This DTD should be capable of including packages that have their own DTDs; for example, the Dublin Core DTD being prepared as one of the results of the Warwick workshop. The framework DTD must also be able to incorporate metadata packages that do not conform to any DTD. This arises when incorporating packages that use a non-SGML notation, such as the USMARC format. Finally, there should be a quick and easy way of representing metadata elements in SGML, but without the potential for collisions between element names that arises when aggregating many DTDs into one document.

The DTD in Figure 7 meets those requirements. The container/package hierarchy is implemented by the `<container>` element and the `%PackageTypes` parameter entity. Parameter entities are essentially text substitution macros for portions of a DTD. Packages which have their own DTDs are easily included using the SGML idiom of overriding the definition of the `%md-set` parameter entity, and by providing the required DTD fragment in the document's declaration subset. Packages in a non-SGML format can be incorporated by use of the `NOTATION` attribute on the `<package>` element. An example of the use of the `NOTATION` attribute is shown in Figure 8. Although we have not done so in that example, we recommend that non-SGML data be included by means of an entity reference, since occurrences of the character "<" inside a package interfere with parsing. The technique in Figure 8 is appropriate when you can guarantee that "<" will not appear in the data. Note that references to external entities are essentially indirect packages, so it may be the case that most non-SGML data should be handled as indirect packages. Finally, the `<metagroup>` and `<metadata>` elements allow for novel elements to be incorporated without more DTD definitions.

Because of DTDs, SGML meets the requirement for strongly-typed packages. However, there is no provision for a registry of elements. Because of this, element names in different DTDs could potentially conflict. There are no truly satisfactory methods for dealing with this problem. The `SUBDOC` feature of SGML can prevent the name space collisions, but it is not widely implemented. The `<metadata>` element discussed in the previous paragraph can be an acceptable method of preventing name space collisions that would result from combining DTDs that define the same element. However, the application is still responsible for knowing how to handle the cases when multiple `<metadata>` elements have the same value for the `NAME` attribute.

A document conforming to the DTD in Figure 7 will have an outer `<container>` element, which must contain one or more elements of the known types of packages. Packages can be another container, an `indirect` package, or a `md-set`. The `md-set` is a parameter entity, which will allow its definition to be easily changed when packages with their own DTDs must be accommodated. The list of package types is also defined as a parameter entity for similar reasons.

```
<!--  Warwick Framework DTD -->
<!-- Override this entity definition to add other metadata packages
     that follow their own DTDs. -->
<!ENTITY % md-set 'DublinCore | package'>

<!-- Override this entity to add other notations for non-SGML data. -->
<!ENTITY % notations 'SGML | USMARC | MIME | RFC-822'>
<!NOTATION SGML    SYSTEM "" -- Default value for package notation -->
<!NOTATION USMARC  SYSTEM "" -- Some known encoding of US MARC -->
<!NOTATION MIME    SYSTEM "" -- An IETF MIME message -- >
<!NOTATION RFC-822 SYSTEM "" -- Simple attribute: value pairs -->

<!ENTITY % PackageTypes  'container | indirect | %md-set'>

<!ELEMENT  container - - (%PackageTypes)+>
<!ATTLIST  container
  Name     CDATA  #REQUIRED  -- Name of container schema --
  Schema   CDATA  #IMPLIED   -- URI for container schema def.--
  Version  CDATA  #IMPLIED   -- Version of the package schema -->

<!-- The <indirect> element refers to packages of metadata held
     externally. The URI attribute specifies the remote package.
-->
<!ELEMENT indirect  - O EMPTY>
<!ATTLIST indirect
  URI      CDATA  #REQUIRED -- The reference to the data --
  Name     CDATA  #IMPLIED  -- Name of ext. package schema --
  Version  CDATA  #IMPLIED  -- Version of the package schema -->

<!-- The MD-SET parameter entity should be overridden when people
     wish to add packages with known DTDs. For now it mentions DublinCore
     and Package. DublinCore has its own DTD, but for this example we
     take the easy way out by declaring its content as #PCDATA.
     The NOTATION attribute allows the Package element to include a
     metadata package that is in a non-SGML format. If people do not
     want to follw a package-specific DTD, they may use the metadata
     and metagrou elements inside a package.
-->

<!ELEMENT DublinCore - - (#PCDATA) -- deal w/ this later -->
<!ELEMENT package  - - (#PCDATA | metadata | metagroup)+ >
<!ATTLIST package
  Name     CDATA                #REQUIRED -- Name of package schema --
  Schema   CDATA                #IMPLIED  -- URI of schema definition --
  Version  CDATA                #IMPLIED  -- Version of package schema --
  Notation NOTATION (%notations) SGML     -- non-SGML formats allowed,
                                             but SGML is the default -->
<!ELEMENT metagroup - - (metadata | metagroup)+ >
<!ATTLIST metagroup
    Name   CDATA  #Required  -- Name of the metadata grouping --
    Type   CDATA  #IMPLIED   -- Categorization of metadata --
    Scheme CDATA  #IMPLIED   -- Reference to controlled vocabulary -->
<!ELEMENT metadata - - (#PCDATA) >
<!ATTLIST metadata
    Name   CDATA  #Required  -- Name of the metadata field --
    Type   CDATA  #IMPLIED   -- Categorization of metadata --
    Scheme CDATA  #IMPLIED   -- Reference to controlled vocabulary -->
```

Figure 7 - Warwick Framework DTD

The `<container>` and `<package>` elements have identical lists of attributes, which let us specify the name of the package, as well as its version and a URI to fetch more information on the package type. The `<indirect>` package element is slightly different, in that its URI attribute specifies the package data to fetch.   One question to be answered is just how much information should be supplied about indirect packages. Should we indicate its datatype?

Note in the DTD in Figure 7, the `<DublinCore>` element is just defined to hold text. This makes the example simpler, but in reality we would use the DTD for the Dublin Core that is currently being developed.

An example of the use of the Warwick DTD is given in Figure 8.

```
<!DOCTYPE container system "warwick.dtd">
<container name="example">
<indirect uri="http://foo.bar.com/huh">
<package name="admin">
<metadata Name="date-created">12/31/95</metadata>
<metadata Name="last-revised">4/5/96</metadata>
</package>
<DublinCore>
  For this example, just some text. A DTD for the Dublin
  core is being developed, and the content here should conform
  to it in real use.
</DublinCore>
<package name="misc" notation="RFC-822">
From: daniel@acl.lanl.gov (Ron Daniel)
Subject: Metadata tagging schemes
</package>
</container>
```

Figure 8 - Warwick Framework container encoded in SGML

9.4 Distributed Object Implementation

An object-oriented implementation of the Warwick Framework is appropriate for a number of reasons.

- Strong Typing - The object abstraction allows us to clearly define containers and packages as entities with fixed attributes and operations. This type structure would allow us, for example, to operationally define the full spectrum of metadata sets – from those which are simply attribute/value pairs (*e.g.*, the Dublin Core) to those with more complex operational characteristics (*e.g.*, terms and conditions metadata sets).
- Information Hiding - This feature of the object model is closely tied to strong typing. The object model restricts access to entities according to their publicly defined interfaces. For example, in a object-oriented implementation a client would not have access to the stream representation of a metadata container (as in simpler implementations such as the HTML example above). Instead, clients would only have access to the defined method of the container; *i.e.*, the operation which returns the *references* to the set of packages in the container. Similarly, since this operation only returns references to the contained packages, access to the actual "data" in the respective package is subject to the operations defined for the package.
- Inheritance Hierarchy - The various types of metadata sets defined earlier in this document fit quite well into an inheritance hierarchy. For example, the various types of descriptive cataloging data have common characteristics that could be expressed through a common super-type. This is also true for other types of metadata. The inheritance hierarchy could be useful for semantic transformations among metadata sets with similar semantics; for example, Dublin Core records and the RFC-1807[19] bibliographic records used in the NCSTRL[20] project. A type hierarchy would also be useful when clients encounter unknown metadata types, allowing them to partially access the metadata through a known sub-type.

Distributed object technology extends the object abstraction by providing non-local access to objects – a client of an object may be located in a different address space or different machine than the server that contains the actual implementation of the object. Some examples of distributed object implementations are the many implementations based on the Object Management Group's[21] CORBA[22] specification, ILU[23] (a CORBA-like implementation from Xerox), and the proposed DCOM extension to OLE[24] from Microsoft.

Each of these implementations either propose or have some preliminary implementation of extensible security frameworks, which allow implementers and server administrator to control access to objects or methods within objects. This is especially relevant to an implementation of the Warwick Framework, and in fact the entire information infrastructure, where access to intellectual content must be limited to authorized parties and under terms defined by terms and conditions associated with the object (as metadata).

The CORBA specification also includes a number of proposed higher level services, two of which are relevant to the metadata and content issues addressed by this document. The interface repository[25] service allows providers to register new object types. Clients may then use the information in the registry to access those types. The dynamic invocation interface[26] service makes it possible for clients to assemble method calls to the operations defined for these new types.

We note that a distributed object implementation of the Warwick Framework pre-supposes an information infrastructure quite different from that which currently exists. There are a number of efforts underway to create such an infrastructure, as exemplified by the Stanford University Digital Libraries Project[27] and the June 1996 joint W3C/OMG Workshop on Distributed Objects and Mobile Code[28].

Figure 9 shows the type hierarchy for an object-based implementation of the Warwick Framework:

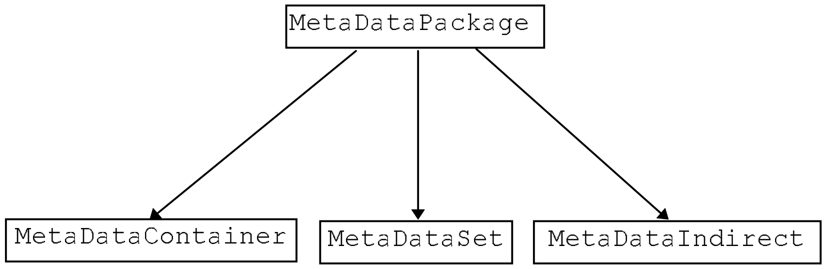


Figure 9 - Class hierarchy for metadata

The classes in this type hierarchy are as follows.

- `MetaDataPackage` - This is the super-class from which all other metadata classes are sub-typed, as shown above. There are no methods defined for this type. Its only serves as the abstract type returned by the `GetPackages` method of the `MetaDataContainer` type. This then allows, at run-time, any of its subtypes to be returned by this method.
- `MetaDataContainer` - This class implements the container abstraction in the Warwick Framework. There is one access method defined for this class, `GetPackages`. This method returns a sequence of references to objects of class `MetaDataPackage`. As stated earlier, the ordering of this sequence has no relevance. The sub-typing of `MetaDataPackage` allows each element of this sequence to be a reference to either a `MetaDataSet`, `MetaDataIndirect`, or `MetaDataContainer`. It is important to note that *references* to packages to MetaDataPackage objects are returned and not the objects themselves. This means that the package objects are, at this level, opaque to the client. The client can only access the package objects according to their operational definitions, and access to these operations may depend on secure access defined for that object. In addition, “skipping” over a package merely entails getting the object reference to the next package in the container. This preserves the modularity of access to containers and packages in the Warwick Framework definition.
- `MetaDataIndirect` - This class implements indirectly-referenced metadata. The required access method for this class is `GetIndirectionID`, which returns a URI of the object indirectly referenced. It would be possible to implement another method, `GetIndirection`, which returns an object of type `MetaDataPackage`, effectively resolving the indirect reference recursively until a “leaf” object is located.

- `MetaDataSet` - This class is the abstract super-class for all of the actual metadata sets; *i.e.*, those objects that actually contain descriptive data such as Dublin Core metadata or terms and conditions metadata. As stated earlier, the different classes of metadata sets naturally fall into an inheritance hierarchy. The creation of such a type hierarchy is fertile ground for future research work in the area of metadata. The metadata taxonomy presented earlier in this document might be a good starting point for this class hierarchy. Figure 10 shows a partial example of such a type hierarchy.

The Kahn/Wilensky Framework[29], a result of the DARPA-funded Computer Science Technical Reports Project[30], proposes a distributed information infrastructure into which object implementation of the Warwick Framework fits. A quick overview of this framework is as follows. The proposed information infrastructure is an open architecture supporting the deposit, storage, dissemination, and management of information in digital form. Information in the system is stored in the form of a *digital object*, which is a content-independent package encapsulating intellectual content, or the *data* of the object, and other associated material (*i.e.*, the metadata that is the subject of this document). An important aspect of the infrastructure is its strict segregation of content-independent concerns, at the digital object level, and content-dependent concerns, at the data level.

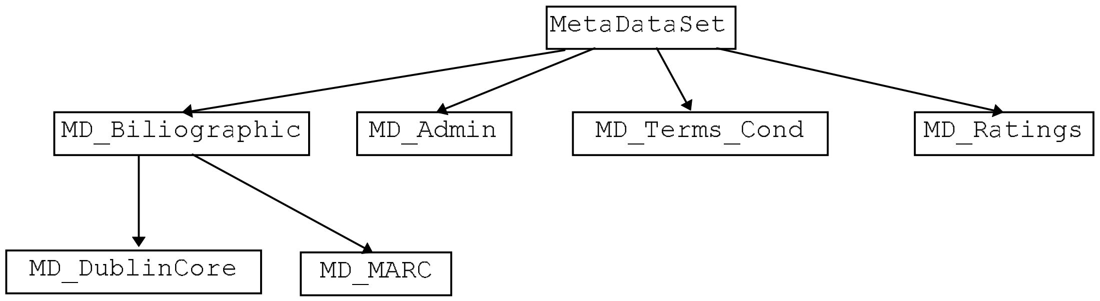


Figure 10 - Class hierarchy for MetaDataSet

The Kahn/Wilensky document does not strictly define the metadata associated with a digital object. It does, however, describe two important types of metadata that are essential for the infrastructure.

1. *Unique-Persistent Name* - The Kahn/Wilensky document describes a URN implementation known as the handle system[31].
2. *Terms and Conditions* - The Kahn/Wilensky Framework places strong emphasis on the legal and social context into which the information infrastructure must fit. Protection of intellectual property is a key requirement for the success of any technology in this context. The framework addresses this issue by prescribing that terms and conditions for access should be embedded or attached to content objects, and that access to these objects be governed by these terms and conditions.

Digital objects are logically stored in *repositories*. Kahn/Wilensky describes the functionality of a basic Repository Access Protocol (RAP), with the following low-level operations:

- `DepositDigitalObject` - Deposit a new digital object in a repository.
- `AccessDigitalObject` - Return a reference to a digital object in a repository.

A repository has the important role of restricting access to these operations subject to the terms and conditions defined for the repository and for the digital object that is the target of the operations.

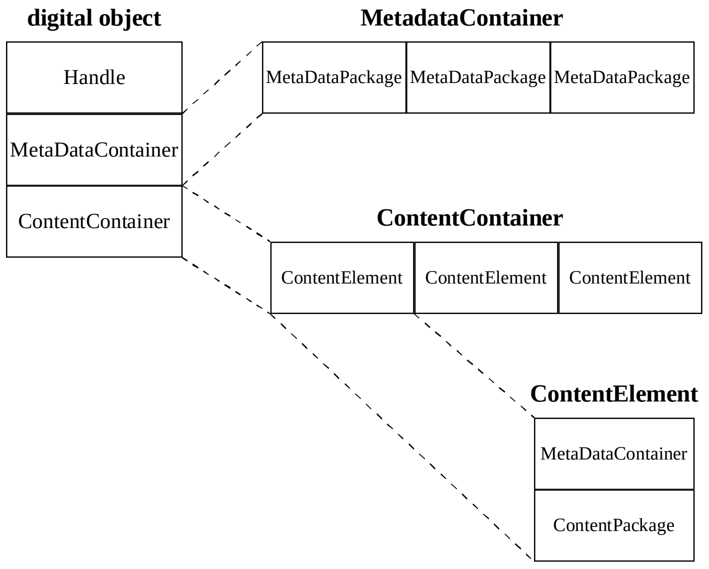
In follow-on work to the original Kahn/Wilensky paper, researchers at the Cornell Digital Library Research Group[32], CNRI, and NCSA examined implementations issues[33] for the framework and developed a design for a distributed object implementation[34] of the Kahn/Wilensky framework. The latter paper also examines the issues related to enforcing terms and conditions in a distributed object implementation.

Work on a distributed object implementation, using ILU, is currently underway at Cornell. This work incorporates the Warwick Framework notion into the Kahn/Wilensky framework. In this implementation a digital object includes three classes of objects:

1. `URN` - The unique identifier for the digital object.
2. `MetaDataContainer` - The container of metadata (*internally-referenced*, in the sense described earlier) for the content of the digital object. A `MetaDataContainer` is a sequence of objects of type `MetaDataPackage`. The full semantics of the Warwick Framework container are supported.
3. `ContentContainer` - The set of content in this digital object. This is a sequence of objects of type `ContentElement`. Each `ContentElement` contains a `MetaDataContainer` and an object of `ContentPackage`. A `ContentPackage` is the primitive super-type for a class hierarchy that matches that of a `MetaDataPackage`. That is, a `ContentPackage` may sub-class to “true” content (*e.g.*, PostScript, GIF image, Java applet), an indirect reference to another digital object, or recursively to another `ContentContainer`.

Note that a digital object contains two sets of metadata containers. One is at the object level, holding metadata relating to the digital object as a whole. Another set is attached to each content element, holding metadata relating to that specific piece of content. The digital object data structure is illustrated in Figure 11.

One additional feature of the class hierarchy, not illustrated here, is that the distinction between metadata



.

Figure 11 - A digital object containing a Warwick Framework container

and data is purely an artifact of how some “content” is packaged in a given digital object. Both the `MetaDataContainer` and `ContentContainer` are hierarchical aggregates with “content” at the leaves. A given type of content, for example, a MARC record, may exist as a leaf descending from the `MetaDataContainer` one of one digital object and as a leaf descending from the `ContentContainer` of another. This is consistent with our earlier observation that there really is no absolute partition between what is metadata and what is data, and that the partitioning between the two depends wholly on the context in which they are used.

In summary, this digital object design permits arbitrary aggregations of metadata and content within a first-class (named) object. Each element of the aggregation may, itself, be a first-class object with independent administration, descriptive data, and rules for access.

10. Acknowledgments

The authors wish to thank the organizers of the metadata workshops, especially S. Weibel, whose efforts provided an essential forum for this and other related work. The ideas here draw extensively from discussions at the Warwick workshop; they also reflect the influence of work done on the still-incomplete White Paper on Networked Information Discovery and Retrieval by C. Lynch, A. Michaelson, C. Preston and C. Summerhill that is being prepared for the Coalition for Networked Information. We would also like to acknowledge the extensive work of E. Miller, J. Knight, M. Tomlinson, L. Burnard, C.M. Sperberg-McQueen, and L. Quin on the HTML, MIME, and SGML implementation proposals described here. Finally, we would like to thank J. Davis and M. Curtin at Cornell for their careful editing.

[1] OCLC/NCSA Metadata Workshop. http://www.oclc.org:5046/oclc/research/conferences/metadata/
[2] Stuart Weibel. Metadata: The Foundations of Resource Description. D-lib Magazine. July, 1995. http://www.cnri.reston.va.us/home/dlib/July95/07weibel.html
[3] Corporation for National Research Initiatives. d-lib Magazine. http://www.dlib.org
[4] Stuart Weibel, Jean Godby, Eric Miller, and Ron Daniel. OCLC/NCSA Metadata Workshop Report. http://www.oclc.org:5046/oclc/research/conferences/metadata/dublin_core_report.html.
[5] The Library of Congress. Machine-Readable Cataloging. http://lcweb.loc.gov/marc/marc.html
[6] CS-TR: Computer Science Technical Reports. http://WWW.CNRI.Reston.VA.US/home/cstr.html.
[7] Federal Geographic Data Committee. Content Standards for Digital Geospatial Metadata. http://geochange.er.usgs.gov/pub/tools/metadata/standard/metadata.html
[8] The Federal Geographic Data Committee, http://fgdc.er.usgs.gov/fgdc2.html
[9] Rating Services and Rating Systems. Internet Draft. ftp://nic.merit.edu/documents/internet-drafts/draft-pics-services-00.txt
[10] Universal Resource Names. http://union.ncsa.uiuc.edu/HyperNews/get/www/URNs.html
[11] Corporation for National Research Initiatives, The Handle System. http://www.handle.net
[12] Digital Library Initiative. http://www.grainger.uiuc.edu/dli/national.htm
[13] Eric Miller. Home Page. http://www.oclc.org:5046/~emiller/
[14] Online Computer Library Center, Inc., http://www.oclc.org
[15] W3C Distributed Indexing/Searching Workshop. http://www.w3.org/pub/WWW/Search/9605-Indexing-Workshop/
[16] Stuart Weibel. A Proposed Convention for Embedding Metadata in HTML. http://www.oclc.org:5046/~weibel/html-meta.html
[17] Beniot Marchal. A Gentle Introduction to SGML. January, 1996. Http:/www.brainlink.com/~ben/sgml.
[18] SoftQuad, Inc. SGML Primer. Http://www.sq.com/products/orders/prorder.html.
[19] R. Lasher, D. Cohen. A Format for Bibliographic Records. RFC-1807. file://nic.merit.edu/documents/rfc/rfc1807.txt
[20] The Networked Computer Science Technical Reports Library. http://www.ncstrl.org
[21] Object Management Group. http://www.omg.org
[22] Information Resources for CORBA and the OBG. http://www.acl.lanl.gov/CORBA/
[23] Xerox Palo Alto Research Laboratory. Inter-Language Unification. ftp://ftp.parc.xerox.com/pub/ilu/ilu.html
[24] Brockschmidt, Craig. Inside OLE 2, Second Edition. Microsoft Press, 1995.
[25] Object Management Group, Object Request Broker 2.0 Extensions Interface Repository RFP, http://www.omg.org/doc/1993/93-09-16.ps
[26] NEC, Dynamic Invocation Interface Example, http://www.omg.org/docs/1993/93-01-02.tar.Z.uu
[27] Stanford Digital Library Project. http://www-diglib.stanford.edu/diglib/
[28] Joint W3C/OMG Workshop on Distributed Objects and Mobile Code. http://www.w3.org/pub/WWW/OOP/9606_Workshop/
[29] Robert Kahn and Robert Wilensky. A Framework for Distributed Object Services. May 13, 1995. http://WWW.CNRI.Reston.VA.US/home/cstr/arch/k-w.html
[30] Corporation for National Research Initiatives. Computer Science Technical Reports Project. http://WWW.CNRI.Reston.VA.US/home/cstr.html
[31] Corporation for National Research Initiatives. Handles and the Handle System. http://WWW.CNRI.Reston.VA.US/home/cstr/handle-intro.html
[32] Cornell Digital Library Research Group. http://cs-tr.cs.cornell.edu/Dienst/htdocs/Info/group.html
[33] Carl Lagoze and David Ely. Implementation Issues in an Open Framework for Digital Object Services. Cornell Computer Science Technical Report TR95-1540.http://cs-tr.cs.cornell.edu:80/Dienst/UI/2.0/Describe/ncstrl.cornell%2fTR95-1540
[34] Carl Lagoze, Robert McGrath, Ed Overly, and Nancy Yeager. A Design for Inter-Operable Secure Object Stores. Cornell Computer Science Technical Report TR95-1558. http://cs-tr.cs.cornell.edu:80/Dienst/UI/2.0/Describe/ncstrl.cornell%2fTR95-1558