Supporting Relationships in Digital Libraries

Trond Aalberg

Department of Computer and Information Science
Norwegian University of Science and Technology
April 23, 2003

URN:NBN:no-3403

URN:NBN:no-3403

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for the doctoral degree “doctor scientiarum”. The work reported has been carried out during the time period 1998–2003 at the Information Management Group, Department of Computer and Information Science.

First of all I want to thank my supervisor, Professor Ingeborg Torvik Sølvberg, for providing me with a fine combination of freedom and guidance in the shaping of this work. The discussions we have had and the feedback I have received throughout these years have been invaluable for this work. I also want to thank my colleagues at the Department of Computer and Information Science for the friendly and helpful environment that has enabled me to do this work.

Particular thanks to BIBSYS for allowing me to use their database in the application that was developed as a part of this work.

Finally, I want to thank a very patient and understanding family.

Trond Aalberg
April 23, 2003

URN:NBN:no-3403

URN:NBN:no-3403

Abstract

The motivation for this work is based on two recognized research issues for digital libraries. One is the need for interlinked and semantically rich information spaces where relationship information is of particular importance. The other is the service-oriented architecture of digital libraries. The digital libraries of the future will consist of smaller and independent systems that each will provide different functionality and access to different contents.

This work defines and explores a service for managing and using explicit relationships – the Digital Library Link Service. The service implements an instance-oriented approach to relationships that enables any kind of typed relationship to be created among the information objects of digital libraries. The service can be used to create consistent information spaces on top of digital library repositories and enables an associative organization and retrieval of information objects.

This work shows that the use of a fine-grained relationship model implemented as distributed objects enables distribution of the relationship network while still being able to support constraints and maintain consistency. The cost of this, however, is a complexity that can reduce performance and scalability due to the call latency of network communication. A prototype is developed that utilizes caching in order to solve this. Tests conducted show that this technique significantly contributes to the scalability and efficiency. This is particularly important when the relationship information is distributed across different processes with high call-latency in between.

The work further presents a prototype application for enhancing bibliographic catalogues with the rich set of relationship types defined in the bibliographic information model proposed by the International Federation of Library Associations and Institutions – the FRBR model. The Digital Library Link Service is used to implement an index that facilitates the navigation of bibliographic relationships in order to explore bibliographic entities along the paths laid out by the FRBR model. This demonstrates the applicability of the service as a flexible tool for associative organization of information objects.

The main applications of the service are limited to systems with a relaxed requirement in terms of automatic processing of larger sets of relationships. The main access paradigm explored for interacting with relationships is by navigation. The need for automatic and efficient processing of a large relationship network, e.g. for the purpose of indexing, can be supported by extending the system with additional functionality. Another recognized problem is that the use of CORBA references to address long-term persistent information can cause referential integrity problems. One possible way to solve this is to assigning objects globally unique identifiers that later can be used to recover from referential integrity problems.

URN:NBN:no-3403

URN:NBN:no-3403

Table of Contents

URN:NBN:no-3403

List of Figures

1 Introduction

The process of tying two items together is the important thing.

(Vannevar Bush, As We May Think, 1945)

1.1 Problem Statement

The motivation for this work is based on two recognized research issues for digital libraries. One is the need for interlinked and semantically rich information spaces where relationship information is of particular importance. The other is the service-oriented architecture of digital libraries. The digital libraries of the future will consist of smaller and independent services that each will provide different functionality and access to different contents. These two highly orthogonal research issues lead up to the specific problem statement explored in this research which is:

> *How to provide a service for using and managing relationships in digital libraries.*

1.2 Approaches and Objectives

This work explores several interrelated approaches to the problem statement:

- *Relationships are viewed as a generic type of information:* The use of a common data model for this kind of information can enable uniform access, use and management of semantically and structurally different relationships.
- *Relationships as explicit information:* Implementing relationships as first-class objects enables the creation of relationships between third-party entities. Furthermore, it can enable access to relationship information by third-party applications.
- *Build upon existing technology and experience:* Although current support for relationship information in information systems is based on diverse models and techniques, there are established conventions with respect to issues like relationship semantics and relationship constraints that need to be supported.
- *A distributed solution:* If digital libraries are envisioned as a distributed information environment, a service for relationship management and use needs to be implemented as a distributed application as well.

This leads to the following objectives for this work:

- To propose a model for relationship information in digital libraries.
- To design and evaluate an open solution for representing, managing and using relationship information in digital libraries.
- To explore relationship information in a distributed context.

1.3 Contributions

The major contributions of this thesis are:

- The thesis contributes to the understanding of relationships in digital libraries by reviewing how relationships generally are understood and supported in information technology. A general conclusion is that existing solutions for representing and processing relationship information are highly diverse.
- An abstract model of explicit relationships is specified that provides a flexible solution for representing relationship information in a uniform way. This shows that a generic data model can be used to capture the full range of relationship types that are relevant for digital libraries. This solution, however, requires a relationship typing scheme to ensure logically correct relationships.

- The thesis specifies and describes the Digital Library Link Service – an instance-oriented solution for managing and using relationships in digital libraries. The use of an object-oriented solution enables a coherent and flexible solution for the data, behaviour and constraints of explicit relationships. By separating the typing of relationships from the implementation a reusable and dynamic service for relationship support in digital libraries is achieved. The DL-LinkService can be used to create complex relationship networks.
- This work shows that the use of a fine-grained relationship model implemented as distributed objects enables distribution of the relationship network while still being able to support constraints and maintain consistency. The cost of this, however, is a certain complexity that can reduce performance and scalability due to the call latency of network communication. A prototype is developed that utilizes caching in order to solve this. Tests conducted show that this technique significantly contributes to the scalability and efficiency. This is particularly important when the relationship information is distributed across different processes with high call-latency in between.
- The fine-grained relationship object model that is deployed enables distribution along many axes. The service can be used in a client/server fashion or it can be used for peer-to-peer collaborative construction of a consistent, coherent and highly distributed relationship networks. A general conclusion is that relationship information can be highly distributed if this is needed. Implementing long-term persistent information as distributed objects, however, is a challenging issue due to the possible lack of referential integrity that may occur. This work suggests the use of globally unique identifiers as the main enabler for implementing solutions to solve and prevent such problems.
- The proposed solution supports interactive traversal of the relationship network. However, the main access paradigm explored in this work is user-initiated navigation with a relaxed requirement for processing capacity. The need for automatic and efficient processing of a large relationship network, e.g. for the purpose of indexing, can be supported by extending the system with additional functionality.
- Furthermore, a prototype application for enhancing bibliographic catalogues with a rich set of relationship types is implemented. This demonstrates the applicability of the service as a flexible tool for associative organization of information spaces and illustrates the potentially rich information structures that relationships can enable in digital libraries.

Two papers reporting this work have been presented at the European Conference on Research and Advanced Technology for Digital libraries (ECDL) [1, 2].

1.4 Outline of the Thesis

The remaining of this thesis is laid out as follows:

- Chapter 2 is an introduction to digital libraries and discusses the major characteristics of digital library systems. This chapter further focuses on the information that digital libraries maintain and discusses the benefits of a more relationship-centric approach to digital libraries. The last part discusses digital libraries as a distributed and open environment for sharing and using information and functionality, and outlines how a relationship service for digital libraries can be modelled and implemented.
- The topic of Chapter 3 is to explain what relationship knowledge is. This is accomplished by describing selected theories about knowledge, the mathematical relation, and relationship modeling. The last part of this chapter emphasizes relationship semantics and describes the various types of relationships that are recognized in different fields related to digital libraries. Chapter 4 continues this discussion by reviewing and discussing solutions for relationship mechanisms with a particular focus on object-oriented systems, hypermedia link services and formats for descriptive relationship information. Chapter 5 summarizes the aspects of relationship representation that this work emphasizes, and introduces an abstract model that provides a flexible solution for representing relationship information in a uniform way.
- Chapter 6 introduces and describes the core design of the Digital Library Link Service (DL-LinkService) – the system that the remaining of the thesis explores. Various application issues related to the service is described in Chapter 7, such as how the service can be integrated into digital library systems and how the design can be extended to support features like relationship attributes, and how transactions and security can be used to solve particular application issues.
- Chapter 8 describes the specific prototype that is developed with a particular focus on the performance optimization techniques that are implemented. This chapter further describes various tests that have been performed which are used to explore possible problems and limitations of the system.
- Chapter 9 presents a prototype application that uses the DL-LinkService to implement a semantically rich relationship-based index over bibliographic records.
- Chapter 10 evaluates the main features of the DL-LinkService
- Finally, Chapter 11 concludes this work with a summary of major contributions, limitations and future work.

2 Digital Libraries

2.1 Defining Digital Libraries

In the last decade, digital libraries have grown to be a common field of interest for researchers, developers and users from a broad range of disciplines. Digital libraries do not, however, have any single precise definition, and the many disciplines and practitioners involved tend to emphasize different aspects when engaging in research and development. In their introduction to the first issue of the *International Journal on Digital Libraries*, Adam and Yesha explain the main theme of digital libraries with the following [3]:

> *Digital Libraries are concerned with the creation and management of information sources, the movement of information across global networks and the effective use of this information by a wide range of users.....*

This definition captures the motivation for many research and development efforts, but is highly general in the sense that it does not clearly distinguish digital libraries from other kinds of information systems. After all, not all information systems are digital libraries. A more specific definition is given by Borgman et al. in [20]:

> *Digital libraries are a set of electronic resources and associated technical capabilities for creating, searching, and using information. In this sense they are an extension and enhancement of information storage and retrieval systems that manipulate digital data in any medium (text, images, sounds; static or dynamic images) and exist in distributed networks. The content of digital libraries includes data, metadata that describe various aspects of the data (e.g., representation, creator, owner, reproduction rights), and metadata that consist of links or relationships to other data or metadata, whether internal or external to the digital library.*

Stephen Griffin defines digital libraries with the following [74]:

> *...digital libraries provide for collection development, organisation, access, annotation and preservation, and deal both with information in digital form as well as digital management of information residing on physical media.*

A different description is used by the Association of Research Libraries in [11] based on a review of digital libraries by Karen Drabenstott [58]:

- The digital library is not a single entity.
- The digital library requires technology to link the resources of many.
- The linkages between the many digital libraries and information services are transparent to the end users.
- Universal access to digital libraries and information services is a goal.
- Digital library collections are not limited to document surrogates: they extend to digital artefacts that cannot be represented or distributed in printed formats.

One problem when discussing digital libraries is that they exist in the crossroads between many disciplines and interests. This inevitably leads to many views on what a digital library actually is. In the book "From Guthenberg to the Global Information Infrastructure" [19], Christine Borgman discusses two substantially different meanings of digital libraries:

- Digital library as an organization
- Digital library as a service

The viewpoint of *digital libraries as an organization* is often promoted by the practitioners of the field, based on the assumption that traditional library institutions are the organizations in charge of information management and that the main concern is to enable these organizations to store, manage, and make available digital content in a networked world. On the other hand, research communities – in particular in the field of computer science – often have an initial focus on enabling technologies like databases, information retrieval, and network architectures, and promote the view of *digital library as a service* for collecting and organizing content on the behalf of users, by the use of information technology.

This work is mainly concerned with digital libraries as a specific kind of information system or service. For this purpose it is useful to refer to the description of digital libraries found in [54], which defines digital libraries using three key characteristics that distinguish them from other systems:

- *Functionality:* It offers integrated services to a comprehensive digital collection of cultural or scientific information that is available primarily for reading and secondarily for expanding upon as well as annotating.
- *Purpose:* It is mainly used for learning and research.
- *Lifetime:* It provides access to information whose value is preserved across long periods of time.

Furthermore, some of the features that are particularly emphasized in digital library information systems are:

- Rich information needs
- Multiple sources of related information
- Heterogeneous information
- Rich data sources
- Multimedia information
- Defined user populations
- Motivated users
- Task-orientation
- Domain-orientation
- Cross-lingual access
- Collaboration.

This work particularly focuses on support for relationship information in digital libraries and in this context, the following selected characteristics are important:

- Digital libraries are associated with repositories of long-term persistent information and services for managing, organizing and using the information that exists within these repositories.
- The contents of digital libraries are primarily read-only, but digital library systems should provide tools and services for expanding and enriching content in ways that increase the access to and usability of the content.
- Digital library systems represent multiple sources of related information rather than islands of information. Relationships may exist between information in the same repository, and relationships may exist across system and organizational boundaries.
- Digital libraries exist in an environment that is characterized by collaboration and integration.

2.2 Digital Library Information

Even though the content of digital libraries can be highly heterogeneous, there is a certain shared understanding that underlies most digital library systems. Just as traditional libraries by many are associated with information sources like books, journals, etc., it is natural to assert that digital libraries are associated with similar information sources that exist in the digital realm – digital document-like objects. These entities are often organized in collections, and each "document" is described by metadata records.

Information models for digital libraries have been explored in several digital library research projects [9, 10, 115, 145, 161]. In this work, an abstraction of the information in digital libraries will be used that reflects the various concepts that are commonly used in digital libraries. In its simplest form, the model states that everything in a digital library should be considered an *information object* and that the main mechanism for expanding upon these objects and communicating about them is through the use of names or identifiers.

2.2.1 The Primary Content of Digital Libraries

The primary content of digital libraries can, to some extent, be compared with the established convention of a *document*. Documents can be characterized as knowledge- or information-carrying artefacts, and since the invention of writing, such artefacts have been a major convention for exchanging and storing information. The written language has enabled mankind to capture and store facts and ideas and is fundamental for advanced reasoning as well as the transfer of knowledge from one generation to the next generation. An intuitive understanding of the document concept is naturally based on the written word. In the modern world, however, we are not limited by the pencil and the paper, but can choose between many different technologies for recording ideas and events. For that reason the notion of a document can be expanded to include all artefacts that later can be used as a source for information and knowledge, such as still images, moving images, and sound recordings. Even items like the ones we find in museums can be informative and should be included in this extended notion of a document [26].

Although digital libraries imply the use of digital technology, this does not limit the scope of digital library content to documents encoded and transferred as binary data. Digital representation of information is a rather new invention, and the body of knowledge recorded by humans throughout its history is still dominated by physical artefacts like printed books, manuscripts, and physical images. These physical artefacts may exist in different shapes and may be included in digital libraries by any digital surrogate. A digital image of a manuscript stored in a library or a physical object kept in a museum can be sufficient for the information needs of the

end user. Other artefacts may be represented through other surrogates like a metadata record or merely an identifier. Such surrogates can still be useful in the initial steps of acquiring, discovering, evaluating and selecting information, and should be considered equal to digital content objects when discussing digital libraries.

Information in digital form is characterized by the ease with which it can be changed or altered. For this reason, the scope of digital libraries is naturally extended to include dynamic information as well, such as documents that are updated continuously – like patient records – or information sources that disseminate information influenced by the time it is viewed or other input values – like today's weather forecast. A distinction can, however, be made between the content of digital libraries which is document centric, and the content of database applications which typically is highly structured and based on a predefined knowledge of users and tasks [54].

A major challenge for digital library systems is to be able to deal with a heterogeneous set of documents within the same environment. The content itself can be highly diverse and include different media like text, images and video. Additionally, there are numerous formats in use for storing and exchanging information for each of these media types. Nevertheless, a common denominator for the content of digital libraries is that they are persistent sources of information and that they have some kind of identity that enables the same information to be accessed and reused across time. Despite the variety of media and the formats that are in use, the content of digital libraries should be perceived as a uniform set of *content objects*. This abstraction is needed in order to be able to manage, access and extend upon this information in a uniform way. Typing of information and the type specific processing of this information needs to be supported in a transparent way, such as the way we deal with documents on the World Wide Web where media types [64] are used to transparently communicate information about the format of the transferred document.

2.2.2 Metadata

Bibliographic data and catalogues have for centuries been the main tool for libraries to create order in the universe of their holdings, and a corresponding practice can be found in other domains as well. Descriptive cataloguing is considered by many to be the true art of library practice, and many of the services that libraries provide are centred on the bibliographic records of the library catalogue. The purpose of the catalogue, as stated by Charles C. Cutter in 1904 [43], is to offer the user a variety of approaches or access points to the information contained in the collection of the library by enabling a person to find any work when one of the following is known:

- The author
- The title
- The subject

An additional objective for the catalogue is to assist the user in the choice of a work and to show what the library has:

- By a given author
- On given and related subjects
- In a given kind of literature

In most modern libraries, the card catalogue of earlier centuries is replaced by a database and machine-readable bibliographic records. Databases provide for a more flexible use of bibliographic records, and current online systems allow for searching and browsing the catalogues in a number of ways.

In digital libraries, *metadata* has emerged as the preferred term when referring to information that describes other information. Although some delimit metadata to descriptions of networked resources[1], a more general and inclusive interpretation of metadata is usually implied, as exemplified in the following definition by Dempsey and Heery [56]:

> *Metadata is data associated with objects which relieves their potential users of having to have full advance knowledge of their existence or characteristics.*

A main advantage with the metadata term is that it is a domain independent and common term that covers all the description formats that can be found in any domain. If a digital library as a term is used to denote a highly heterogeneous set of information systems, then metadata should be the preferred term because it is not loaded with the practice and standards of a specific discipline.

Metadata can essentially be interpreted as a surrogate for the actual content object. The information included in a metadata record may serve purposes like:

- Searching, discovering, or selecting information.
- Accessing or retrieving information.
- Identification or verification.
- Contextual information about the information.
- Management and preservation.

1. E.g. ODLIS: Online Dictionary of Library and Information Science, http://www.wcsu.ctstateu.edu/library/odlis.html

Metadata is an important element of digital libraries. Some of the objectives of manually generated metadata can be complemented or replaced by full-text indexing and automatically generated metadata, but certain aspects of descriptions that are based on natural, human language are hard to replace. Using natural language is often the preferred solution of users to express their information need in the search and retrieval process. It will be difficult to replace manually generated language-based metadata, which describes the content of images or videos, by automatic feature extraction techniques simply because the perceived information of such resources relies on human interpretation. Another important feature of metadata is that it enables explicit knowledge about content objects to be stored and maintained. Not all information that is required to manage, organize, locate, and use information, is an implicit part of the actual information. As for content objects, digital library systems utilize a heterogeneous set of metadata formats encoded in a number of storage formats.

2.2.3 Collections

Collections have traditionally been one of the basic structuring paradigms in libraries, archives, and museums. The holdings of a library are often referred to as the collection of that library, but additional collections are frequently used within the library to physically organize and maintain the items of the library – either based on the requirements for certain material to be maintained and processed in a specific way, or because the collection itself represents some logical organization that is convenient in management and use. A collection is additionally an intellectual resource. It represents a selection based on an evaluation of which items are most valuable and have the required potential for reuse. Some collections can be based on the notion of full or partial completeness, and intend to be an as complete as possible selection of items that share a set of characteristic features.

A comparable use of collections can be found in digital libraries. The holdings of the digital library system can be viewed as a collection, or the digital library can contain multiple collections. A collection can represent a specific scope or view that represents a reasonable set of items to browse through or search within, or the collection may reflect a set of objects with the same distinguishing characteristics like a specific subject or the same author.

Collections in digital libraries do not necessarily have to be based on the ownership of information. Information that originates in different repositories can be organized into collections that reflect a virtual organization rather than a physical one. Virtual collections have been explored e. g. in [12, 68].

2.2.4 Compound Objects

A different structure that often is found in digital library is that content (and metadata) consists of a set of subparts. This is slightly comparable to the concept of collections, but whereas collections usually aggregate a set of objects that plays the same role, the subparts of a compound object may be of a different kind and serve different purposes. This compound structure may be either a logical or physical structure:

- The *logical structure* is the structure as it appears to the user of the information. A book may be divided in chapters, subsections, illustrations and an index. An issue of a journal may contain a table of contents and a set of articles. Logical structures can be found in movies, sound recordings and other information sources as well. The logical structure is based on the concept that the structure is an intentional one and that the structure remains intact, independent of the physical representations of the information object. The same structure appears regardless of whether the content is viewed on screen or is printed out.
- The *physical structure* is a different kind of structure that is based on the physical organization of the content. Digital information is often managed and stored as files, and the information that logically appears as a distinct work on screen or when printed out can be stored as different files that are integrated whenever the work is retrieved.

2.2.5 Identity

A main feature in digital libraries is that the information has identity. The identity of information may either be based on the set of distinguishing characteristics – like metadata – or it can be based upon the use of tokens that we associate with entities with the sole purpose of serving as identifiers. Many different identifier schemes for a variety of domains and resource types are in use in both digital libraries and physical libraries. The ISBN and ISSN numbers have been in use since the 1960s to identify books and serials [107, 111]. Another example is the SICI identifier scheme for the identification of specific contributions within serial publications [5].

Identification, addressing and naming are fundamental issues in all information systems, whether local or distributed. The distinction between these terms is not always clear, and different technologies and communities use different vocabularies in this matter. A convenient understanding, in the context of distributed information systems, is promoted by the Uniform Resource Identifier specification [13]. This specification defines identifiers as strings that identify abstract or physical resources. Locators are identifiers that identify resources via a representation of their primary access mechanism, for example network location. Names are

characterized by their purpose of providing logical and persistent identifiers independent of physical location. The basic notion of locators like the web address "http://www.idi.ntnu.no" is comparable to what others would call a communication identifier [41] or address, and the meaning of these terms are fairly intuitive. Names and identifiers, however, are more ambiguous and interchangeable terms. Both terms reflect the general concept of symbols that denotes conceptual and physical entities. The term *name* often implies a symbol that is interpretable by humans (and machines). The term *identifier* implies that the symbol uniquely identifies something.

One solution for identification is to use Universal Unique Identifiers (UUID) [124, 137, 195]. A UUID is an identifier generated and represented according to a scheme that ensures uniqueness across both space and time. UUIDs do not require a registration authority or any other dedicated centralized coordination point. Instead, it uses the network address of the host and a reliable time-stamp from the system clock to generate a unique 128 bit long sequence that ensures uniqueness across both space and time. It can be used for multiple purposes, from tagging objects with a short lifetime, to reliably identifying objects across the network in an infinite time perspective.

The move towards digital information distributed over the Internet has resulted in an extensive use of locator based web addresses (URLs) as the main identification scheme for information. However, this is not a reliable mechanism since the web address for a specific document may change or the content of the web address may change. A far better solution is to use identifier systems that enable a persistent, reliable, and location independent identification of networked, digital content objects. A system that has been developed for this purpose is the Digital Object Identifier [6] that is based on the Handle System developed at CNRI [189, 190]. Another important initiative under development is the URN scheme [44, 140], which is an attempt to establish a global and uniform scheme that can be used to encode all persistent identifiers that exist.

The identity of metadata records and collections is somewhat less evident in the management of information. Metadata records stored in a database usually have some kind of record identifier, but this identity is often not revealed to end users or other systems. Identifiers are, however, equally important for metadata records and collections to provide for exchange and reuse in digital library systems. One example on the use of metadata identifiers is the URI based identification of metadata in the Open Archives Initiative Protocol for Metadata Harvesting [157].

2.2.6 A Digital Library Information Model

Primary content, metadata, collections and compound objects together make up the "business model" of digital libraries. These entities are merely conceptual classes,

and within each of these there will be numerous formats and subtypes in use. In this thesis the entities are further abstracted into the rather general and neutral concept of *information objects*. The *object* part of the expression emphasizes that digital libraries contain units (objects or entities) with a certain identity of their own. The *information* part of this expression emphasizes the fact that these objects are informative – they are capable of being used as a source of information or knowledge. The term *information object* is additionally quite domain independent and will most likely imply comparable entities in different application domains.

To be able to extend upon and enrich the information objects of digital libraries it is required that we are able to communicate and refer to these entities, and the use of identifiers is the primary mechanism for this.

Figure 2.1 summarizes the various entities that have been discussed so far as a class diagram in the Unified Modeling Language [176]. The information object is the most generic concept, and other objects are considered as subclasses of the information object. The only shared feature that is introduced for the information object super-type is that it has identity in the form of an identifier that is associated with the object.

The relationship between metadata and the information object is illustrated by the use of a "description" relationship between metadata and information object. This includes a metadata record which describes another metadata record, or a metadata record which describes a collection or compound document.

As a computational structure, the collection is simply an aggregated set of objects where each item in the collection is a subpart of the parental collection. Collections may very well be present in the digital library as objects in their own rights, and as such they can be considered as a specific kind of information object that aggregates other information objects. Collections do not have to be collections of content objects, but may as well be collections of metadata records (catalogues), etc.

Compound information objects can be modelled using an aggregation relationship equal to the modeling of a collection. A main difference between a compound and a collection is that a collection is constrained to aggregate objects of the same type, whereas a compound may aggregate objects of different type, such as a metadata record combined with a content object.

The model does not focus on the typing of information objects to expresses different media, storage and interchange formats, but is based on the assumption that this is transparently managed by the underlying infrastructure of the digital library.

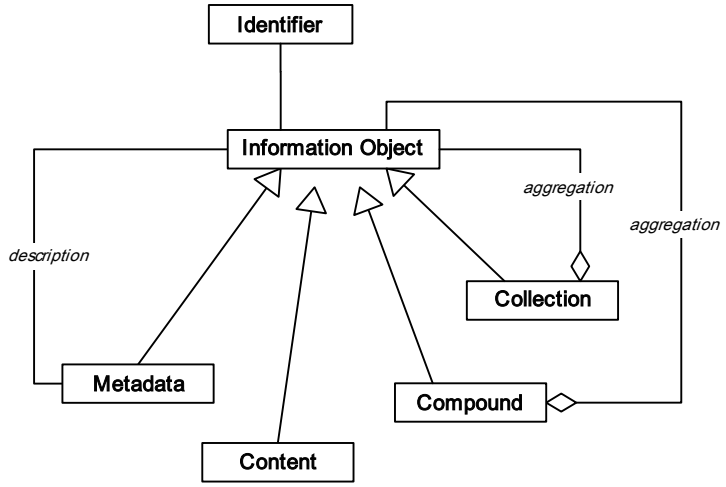


Figure 2.1: A digital library information model.

2.3 A Relationship Centric Approach

2.3.1 Motivation

A relationship centric approach to information management is not a new idea in the context of libraries, whether they are digital or not. In the 1945 essay "As We May Think", Vannevar Bush urges scientists to engage in research and development of technology that would make the increasing body of recorded knowledge more easily available. Vannevar Bush criticized the artificiality of the rigid hierarchical classification indexes of libraries. He suggested the use of associations as the main organizing mechanism when filing and retrieving records of information, and described a future system based on the use of associative "trails" to retrieve information – the memex [29].

Vannevar Bush is often referred to as one of the early pioneers of what later emerged into hypertext systems and the World Wide Web. One of the main reasons for the success of the World Wide Web is the ability to create links between resources and in this way enable the creation of an integrated information space from the resources of many. In hypermedia, links are often said to assert or express relationships and can be considered as the implementation of a relationship instance.

The need for relationship technology in future digital libraries is quite evident in the library and digital library community through various requirements for

relationship support and linking technology. The International Federation of Library Associations and Institutions – IFLA – has suggested a new bibliographic information model in order to support increasing user expectations and needs [100]. The model defines the set of entities that are of main concern in the bibliographic universe and includes a rich set of relationships that can exist among them:

> *In the context of the model, relationships serve as the vehicle for depicting the link between one entity and another, and thus as the means of assisting the user to "navigate" the universe that is represented in a bibliography, catalogue, or bibliographic database.*

The Dublin Core metadata element set is another contribution and recognizes relationships to be a basic element in the description of a resource [60]. The "relation" metadata element is used to express the relationships that may exist between the described resource and other resources. A comparable solution can be found in the IEEE Learning Object Metadata standard [99], as well as in other metadata formats.

The requirement for relationship and linking technology can be derived from many of the issues that are raised in digital library development and the projects that this community is engaged in. One of the main themes that Dagobert Soergel identifies as part of the overall framework for digital library research is concerned with links [185]:

> *Digital libraries need linked data structures for powerful navigation and search.*

Soergel argues that digital libraries need semantic structure. The use of links is considered as a special case of this and can be used to support a range of usage scenarios for the contents of digital libraries. Furthermore, Soergel emphasizes the importance of link semantics and suggests the development of link taxonomies. He concludes by stating that:

> *..there should be links across disciplines and across digital libraries.*

A comparable recognition of linking as a main research topic can be found in [54], which identifies linking as one of the key research topics for collection building.

2.3.2 Relationships Among Information Objects

The abstract information model presented in Chapter 2.2.6 mainly focuses on the core entities of digital library information. The model includes certain basic relationships that are used to express the structure of the model:

- *The inheritance relationship;* which is used to show that the information object is a general super-type for the other entities.

- *The description relationship*; which is the relationships that exist between metadata and the information object that is described by the metadata.
- *The aggregation relationships*; which are the relationships that express the more structural aspects of how the information objects of a digital library are organized.

In addition, numerous other relationships will exist among information objects in digital libraries, and various categories identified are further described in Chapter 3.6. The use of relationships to extend and enhance digital library collections can be characterized as a *relationship centric approach* to digital libraries and the promises of such an approach are many:

- *Relationships enrich the information objects in digital libraries.* Information objects are not atomic entities. In the real world, we rather find that information objects are related to each other in a number of ways as well as being related to a number of other entities. The actual usage of information is heavily depending on the relationships that exist among the entities of information. Semantically rich and inter-linked information spaces require relationships.
- *Information discovery by navigation*. Relationships are a main contributing element for enabling navigational facilities in digital libraries. Navigation is information seeking that proceeds incrementally based on feedback from the system [133]. In the context of information discovery (or information seeking), navigation can be considered an alternative to the set-based query and retrieve paradigm, but in most cases these strategies are complementary rather than competitive.
- *Relationships may cross boundaries.* Relationships are not bound by collection and information ownership, but may cross boundaries. A research article in a scientific journal can be related to many other articles because it directly cites or contains references to other articles. The article can further be related to other information, such as the data sets from experiments or surveys. These interrelated entities can be managed by different organizations and can be stored in different repositories.
- *Relationships as a tool for integration*. Integration is a key issue for digital library systems and can be considered along two different axes – the vertical or horizontal integration axis – depending on how the information sources are aligned. A vertical integration promotes a heterogeneous view on possibly homogenous information. Horizontal integration is a different approach to integration that implies a view on heterogeneous resources as complementary resources that each contribute with different information and functionality to support complex tasks, such as searching for metadata using one service,

using the metadata to locate and retrieve the information from another service, and finding complementary information using a third service.

2.3.3 Relationship Support in Digital Libraries

The availability of relationships is what makes a digital library a rich and interactive information space, and relationship mechanisms are already an element in many digital library systems. Facilities like reference linking [32, 123, 187] and the interactive use of information based on hypermedia technology [4, 42, 211, 213] can both be seen as applications of relationship mechanisms in digital libraries.

One example that illustrates the use of relationships to achieve horizontal integration of multiple repositories is the digital library of the National Centre for Biotechnical Information[1] [148]. This digital library combines the resources of various databases like GenBank – the primary repository for DNA sequences, PDB – the protein databank, and Medline – a reference database for medical publications. One type of relationship that can be found within this system is the relationships that exist between entries in the GenBank or PDB, another kind of relationships exists between the abstracts in Medline that describe articles and the entries in the GenBank or PDB that these articles are related to. These and other relationships that are available in the system enable a rich and interactive information space that adds a significant value to the information they relate.

The main theme, however, when developing digital libraries is the search and retrieve paradigm with an emphasis on manual or automatic indexing of information – metadata versus information retrieval techniques. The support for relationships that can be found in many digital library systems is more in the shape of an added feature or extension of other information objects, and the way it is implemented is highly heterogeneous. Different solutions for relationship representation can broadly be categorized as:

- *Relationships expressed in metadata.* The Dublin Core metadata format contains different elements that can be used for this purpose (like source and relation). Other metadata formats include elements with other syntax and semantics that can be used for the same purpose.
- *Relationships as part of the content.* Documents may contain a rich set of relationships as part of the content. Relationships can be expressed as text, such as the references in scientific articles, or they can be expressed using specific purpose structural elements of a document, such as the links of hypermedia documents. Whereas markup languages like HTML and XML have standardized structures for expressing links, other formats like Micro-

1. http://www.ncbi.nlm.nih.gov/

soft Word, Adobe's Portable Document Format use more proprietary solutions for the same kind of information.

- *Relationships can be external to both metadata and content.* Relationship information may even exist as first class information in its own rights – as explicit relationship information detached from both metadata records and content. This can be implemented by the use of internal ad hoc data structures or open information structures like RDF, Topic Maps, hypermedia links, etc.

2.3.4 Explicit Relationships

The heterogeneous relationship representation used in current digital library technology is a fundamental problem in the development of service-oriented digital libraries. It discourages the development of reusable software solutions for navigation and disables horizontal integration based on relationships. For this reason, this work focuses on the support for relationships in digital libraries by promoting *explicit* and *generic* relationship information.

An explicit relationship is characterized as being a first class object – a distinct unit of information that has an identity of its own. The main advantage of explicit relationships can be summarized as the ability to support relationships in a flexible way. Some advantages can be:

- By revealing the relationships as explicit information independent of the related information objects, they can be processed without the additional overhead of accessing and interpreting other information objects.
- Explicit relationships can be managed and processed as an independent source of information. Different users can access and view different subsets of relationships, and relationships can be interpreted differently in different contexts.
- Explicit relationships are inherently multi-way and can be accessed and navigated in both directions.

A different motivation for the work presented in this thesis is the focus on a generic relationship solution. A generic solution focuses on the core and common aspects of all kinds of relationships, and such a solution will enable:

- *Software reuse.* A generic solution for relationships enables the use of the same software solution in a range of applications. Different types of relationships within the same application can be managed and processed by the same software. Furthermore, different applications can be developed by the use of the same software component for the management and processing of relationships.

- *Interoperability and integration.* The use of a common format for relationship information will be a main enabler for interoperability and integration along the horizontal axis, comparable to how common metadata formats have enabled interoperability and integration along the vertical axis.
- *Information reuse.* Generic relationship support additionally enables reuse of relationship information across applications. Relationships that are captured and stored with a particular usage in mind can easily be accessed and reused in other contexts. This also enables a more generic solution to digital libraries by enabling repositories to store and maintain a structured information space that can be used in different end user applications.

In the abstract information model for digital libraries, the explicit relationship object can be depicted as a subtype of information object, as shown in Figure 2.2 where the relationship object is modelled using the association class construct of UML.

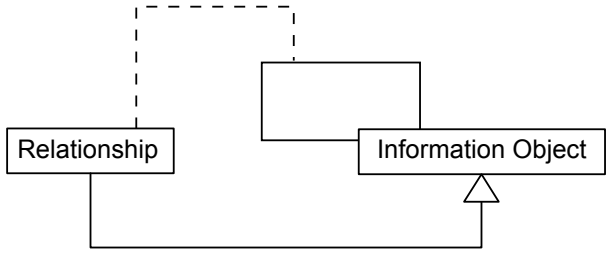


Figure 2.2: Explicit relationships.

The use of explicit relationships is explored in many disciplines such as hypermedia and object-oriented systems, and a motivation for this work is to adapt such technology to the digital library environment as a general purpose relationship service. Explicit relationships and the management and use of this information is an immature area in digital libraries. The support is often closely connected to specific applications and the various solutions already deployed are based on many different technologies. This work rather emphasizes a uniform and flexible solution. A uniform solution is needed to enable reuse and interoperability between different digital library systems and applications. A flexible solution is needed to enable the use of the uniform solution in different applications. This can be achieved if the relationship support is a lower level structuring facility that easily can be adapted to a broad range of relationship types as well as a broad range of relationship applications.

2.4 Digital Library Services

2.4.1 The Global Information Infrastructure

Support for cooperation and integration is a major concern in digital library research and development, and the digital library is by many envisioned as a *global information infrastructure* [19, 178].

Sharing and reuse of resources is already an established practice in traditional libraries, and the need for users to access remote services predates the invention of digital libraries. The Library of Congress started as early as 1901 to offer their bibliographic descriptions to other libraries. The motivation was to provide bibliographic information to libraries that could not afford the creation of such information themselves, and a significant advantage of this was the uniformity of the same bibliographic description in different catalogues. During the 1960s the Machine Readable Cataloguing format (MARC) became the default format for exchange of bibliographic records [108]. The need for remote access to catalogues is another requirement that was early recognized in the library world, resulting in the standardized information retrieval protocol Z39.50 [110].

Today, digital information communicated over the Internet is the main enabler for information sharing and reuse in digital libraries. The Internet, however, is more than a medium for users to retrieve information from remote servers in the traditional client/server setup. The widespread use of the Internet protocols enables more complex applications of distributed digital library systems based on the vision of world-wide integration and cooperation between the various vendors and users of digital library content and services.

2.4.2 Digital Library Architectures

The vision of the global information infrastructure emphasizes digital libraries as distributed and open information systems. A distributed system can be defined as a collection of autonomous computers linked by a network, with software designed to produce an integrated environment. A distributed system, in general, enables cooperation and sharing of resources, and a well designed distributed system should perceive a single, integrated computing facility [41, 192]. Openness implies that system use established and well-defined protocols. A distributed digital library should be considered a distributed application or a specific type of distributed information system. It uses the underlying distributed system of networks, machines and other resources to provide an integrated environment where data and functionality are located on multiple computers that communicate through a network but appear as a coherent resource.

According to [54], the current typical client-server and 3-tier architectures frequently found in digital library systems are not adequate to provide the functionality required to achieve the high-level vision for the digital libraries of the future. The basic system architecture should rather explore open architectures that are *component-based* and *multi-tier:*

> *The Digital Libraries of the future will be ever-expanding systems. An open architecture implies that the overall functionality of the Digital Library will be partitioned into a set of well-defined services. A Digital Library will consist of smaller independent systems that will each provide different functionality or access to different contents.*

Digital library architectures have been explored in many different projects. The University of Michigan Digital Library focused on the use of cooperating agents in a heterogeneous digital library system. A different metaphor was investigated in the Stanford Digital Library Project, by the definition of a set of protocols commonly referred to as the Infobus [160, 172]. The Dienst protocol [51] and its descendants like NCSTRL [128] and OpenDLib [35] are other systems for distributed digital libraries, and more recently the Open Archives Initiative [120] has emerged as a possible framework for sharing and reuse of metadata in a range of applications.

2.4.3 Components and Services

Researchers in digital library architectures often interchangeably use the terms *component* and *service,* although they actually denote quite different concepts. A component is typically a software artefact, whereas a service is a resource that a client can make use of in a dynamic way.

The component expression has a background in reusable and replaceable units of software or hardware. A *reusable* component is an asset that can be deployed as a subpart of different composites [191]. A *replaceable* component is a component for which another component can be substituted without substantial modification to the new component or existing system [180].

Service, on the other hand, is a more ambiguous term. In the client-server model of distributed systems a server is the process or machine that manages and makes available shared resources. In this context, a service can be interpreted as equivalent to a server process, but most uses of the term “service” imply a decomposition of a larger application into smaller units for the purpose of reuse and dynamic integration. Software components promote reuse and substitution of software artefacts, whereas services promote the reuse of network accessible resources or functionality.

Service-oriented computing has recently emerged as the new computing paradigm for a networked world. According to the call for papers for the First International Conference on Service Oriented Computing[1], service-oriented computing is:

> *..the new emerging paradigm for distributed computing and e-business processing that has evolved from object-oriented and component computing to enable building agile networks of collaborating business applications distributed within and across organizational boundaries.*

Steve Vinoski defines service-oriented architectures in [204] as a three-step interaction model:

> *... the concept is actually quite trivial: A service with a well-defined interface and data interchange characteristics advertises itself in a distributed directory service where applications can look to find the details for interacting with the service.*

Although the notion of service-orientated digital libraries tends to be more ambiguous, research projects that are exploring service-oriented digital libraries are based on the same core idea of dynamic discovery and integration of services.

The most typical service a digital library provides is that it enables the users to search and retrieve information, but a more complete list needs to include many other services as well. Examples on services relevant in digital libraries can be:

- Search services that facilitate search for information and returning the result of a search.
- Browsing and navigation facilities that provide for information discovery based on informal and heuristic information seeking strategies.
- Name resolution, which can facilitate the transparent resolution from location independent and persistent names to locators that end users will retrieve the content from.
- Retrieval services that facilitate the evaluation of search results and/or the retrieval of the actual content from repositories.
- Authorization and access management, which is related to the authorization of users and the verification of their access rights for the use of the service or the retrieval of content.
- Personalization services that can support the user with services to maintain personal collections, personal profiles, etc.
- Services for building digital library collections by harvesting and indexing available information.

1. http://www.unitn.it/convegni/icsoc03.htm

- Archiving, storing, and conversion of information to facilitate long-term persistence.
- Automatic or manual generation of metadata, including services to convert between metadata formats.

Reuse of services is a natural consequence of the general evolution of distributed systems. Today, a vast number of machines are connected to the Internet which provides a convenient platform for reuse of resources and functionality. Just as software components are meant for composition into software artefacts, services promote composition or extension of applications through the runtime integration of network accessible services. The combination of services into a specific digital library application can either be preconfigured to solve a well-defined need, or the services can dynamically be discovered at runtime and support adaptive applications that will meet the changing and individual needs of users. A service-oriented architecture, however, requires an underlying infrastructure that supports this modularity in a plug-and-play fashion.

2.4.4 Middleware

Middleware is a common term used to refer to a broad range of software and associated protocols that provide platform independent development and deployment of distributed applications. Middleware is connectivity software that allows multiple processes running on one or more machines to interact across a network [102]. Middleware additionally enables the development of applications that will run on multiple platforms, and they include high-level services that mask much of the complexity of networks and distributed systems [14]. In essence, middleware is the software that resides above the network and below the business aware application [201]. In the context of digital library architectures, the middleware is an important part of the infrastructure that enables easy and dynamic integration and deployment of services without requiring extensive implementation efforts. However, middleware is a vague term that includes solutions based on different computing paradigms:

- *Message oriented middleware;* which is based on the use of asynchronous and indirect calls between the client and server applications – messages.
- *Remote procedure calls;* which implement network communication in a way that resemble the procedural calls of programming languages.
- *Remote data access;* which provides protocols and APIs for communicating with database servers by sending data manipulation language statements and receiving the results.

- *Distributed objects;* which are based on the object-orientated paradigm and implements distributed software as objects that are accessible over the network.
- *Distributed transaction processing;* which implements client/server interaction with transactional execution semantics.

Orthogonal to the above categories is the need for additional general purpose functionality that is common for many distributed applications, such as directories that can be used to discover and locate available resources. Such general purpose services are not bound to specific applications and can even be independent of the computing paradigm of various middleware solutions.

Available middleware solutions can be categorized as middleware components, middleware environments or compound middleware environments [201]:

- *Middleware components* are solutions (products) that provide only one service, such as a solution for security or a naming or directory service.
- *Middleware environments* provide an integrated environment that includes both the protocols and APIs for network communication and the set of services needed for distributed applications. One example is CORBA which is an integrated complete solution for distributed applications. It includes a basic platform for inter-object communication (the Object Request Broker) and a suite of generic services needed for developing and deploying distributed applications.
- *Compound middleware environments* are frameworks for distributed computing that combine different protocols and middleware components into a single framework for distributed application development and deployment.

Any distributed solution, however abstract and implementation independent it initially is intended to be, is bound to be influenced by the underlying middleware it is built upon. Categories of middleware solution appear quite different in their abstraction of the distributed system, and different middleware solutions are often not interoperable. A main problem for digital library interoperability is the heterogeneous usage of middleware in current digital library architectures.

2.4.5 The World Wide Web

The World Wide Web (WWW) was initially designed as a global hypertext system by Tim Berners-Lee, and since its introduction it has gained enormous popularity and has greatly influenced the way information is made available in the global network of the Internet [30, 126]. As a distributed application, the World Wide Web is initially information centric rather than functionally oriented. It is motivated by the goal of providing location-transparent access to information for a global

community. The World Wide Web is defined as the abstract space of information that is interconnected by the use of links. The success of the World Wide Web is based on the use of:

- The Internet as the “backbone”
- The Hypertext Transfer Protocol
- The Hypertext Markup Language document format
- The addressing mechanism of URI
- The web browser as a universal client

During the last decade, the World Wide Web has grown exponentially, and this growth is not only in terms of the number of web sites, web pages, and web users. The World Wide Web has additionally expanded in terms of web-related technology and web-enabled applications. The World Wide Web as an information space and application environment is by no means limited to HTML documents accessed over the HTTP protocol. Through the support for forms in HTML and the data passing mechanisms of URI and HTTP, numerous interactive services have been developed that extend the initial information centric nature with service capabilities.

Service-oriented computing is considered by many as a further extension to current web technology – Web services. The core of Web services is the use of XML based network protocols like SOAP – a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment [225]. Additional facilities are the XML-based Web Service Description Language [228] and the UDDI[1] directory service for locating web services [154].

Current digital library systems are often well integrated with the World Wide Web in different ways:

- Digital libraries use the web-protocol HTTP and the hypertext markup language HTML as the main communication medium between client applications (web-browsers) and the digital library. Additionally, many distributed digital libraries use the HTTP protocol as a communication channel for multi-tiered solutions.
- The World Wide Web may in itself be viewed as a digital library. Users are able to access and browse the interlinked web pages that are made available on web servers, and they can search for information using the numerous web indexes that are available.

A main feature of the World Wide Web is the linking capabilities of HTML, which is the glue that creates a coherent information space out of the web pages provided

1. Universal Description, Discovery and Integration.

by numerous web servers. Current digital library systems are faced with the problem that ordinary HTML linking is a shallow solution for relationships in information management. The one-way embedded links of the Web are well suited for the presentation of interlinked information in user interfaces, but do not address the need for relationships between a rich and heterogeneous body of information formats and the reuse of this information across applications. The use of web links is mainly for processing in a web browser and is limited to media that are capable of containing such links.

2.4.6 CORBA

CORBA is a middleware that provides for a mature and stable architecture and infrastructure for distributed applications [82]. CORBA is an abbreviation for the *Common Object Request Broker Architecture*, and its specifications are developed by the Object Management Group (OMG). CORBA is object-oriented, and the objects of CORBA are conceptually comparable to the objects of programming languages like Java and C++. A main difference, however, is that the CORBA middleware (the ORB – Object Request Broker) and the standardized protocol IIOP, allows for CORBA objects to interoperate over the Internet regardless of programming language, operating system, or hardware.

What constitutes an object in CORBA is a design decision similar to the design of other object-oriented applications. A text document or an image can be modelled and implemented as an object as well as a search system for a bibliographic database. CORBA objects are typed by the use of OMG's Interface Definition Language (IDL) – a strongly typed declarative language. IDL is used to specify the interface of objects, which in essence is a formal description of the methods of an object. In practice IDL also serves other purposes. When developing a client application the IDL file is used as input for the automatic generation of the programming language code that lets the client communicate with server objects as if they were local objects.

CORBA provides for more or less the same environment for distributed computing as Web services. The IIOP network protocol of CORBA is comparable to the SOAP protocol and IDL is the CORBA equivalent to WSDL. CORBA provides for directory and naming services through the CORBA Trading Object Service. A major difference is that Web service is based on the use of XML and may for that reason be easier to integrate with web-applications. At current, CORBA is on the other hand a more mature solution that is highly integrated with programming environments.

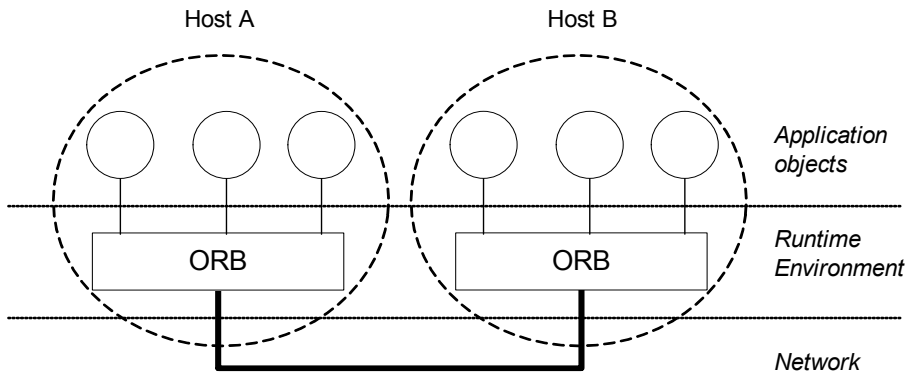


Figure 2.3: The Common Object Request Broker Architecture.

2.5 A Relationship Service for Digital Libraries

This work envisions a service oriented view on relationship information in digital libraries. Relationships are utilized in many of the tasks conducted in digital libraries, and this work explores the support for managing, making available and interacting with relationship information. A relationship service tailored to the environment of digital libraries can enable solutions that otherwise are difficult to support in an interoperable way due to the heterogeneous ways relationships are represented at current.

As described in Chapter 2.3, the use of relationships is a major element in the structuring of digital library information. By exposing this information through a specific purpose service that implements a uniform view on relationship information, the structural aspects of digital library contents can be made available and in this way promote integration along axes that are less supported in current digital libraries.

A general relationship service can be depicted as an intermediate level between the application dependent presentation and processing of relationships and the lower level repository dependent data structures that are used to represent relationships at the data storage level, as illustrated in Figure 2.4.

The lower level of this model is the actual storage of relationship information. Current solutions for representing relationships include a range of solutions. Relationships can be explicit and stored as part of the metadata, as part of the content objects, or separate from both. Other relationships can be implicit and require processing to be discovered, e.g. the topical equality of documents that can be discovered by information retrieval techniques.

The depicted service layer can be used to represent a uniform view on relationships by implementing a common logical information model for relationship information. This is required in order to represent a generic view on the lower level heterogeneous relationship data storage level. An additional requirement for the relationship service level is to provide for a uniform processing of relationships. Both these aspects are required to support a generic solution that can promote interoperability.

At the top level is the application specific processing and presentation of relationships. Relationships are used in many different tasks and appear on the user interfaces of digital libraries in many different shapes. The service layer should not presuppose a specific application or presentation of relationships, but rather be based on a generic model and a set of lower level behaviours that can be used to implement support for relationships in user interfaces and other processes that are clients of the relationship service.

This work uses CORBA as the basis for the architecture of a relationship service for digital libraries. The main objectives for this solution can be summarized by the following:

- CORBA is a mature and stable platform for developing distributed applications, and it is readily available for the most common software and hardware platforms.

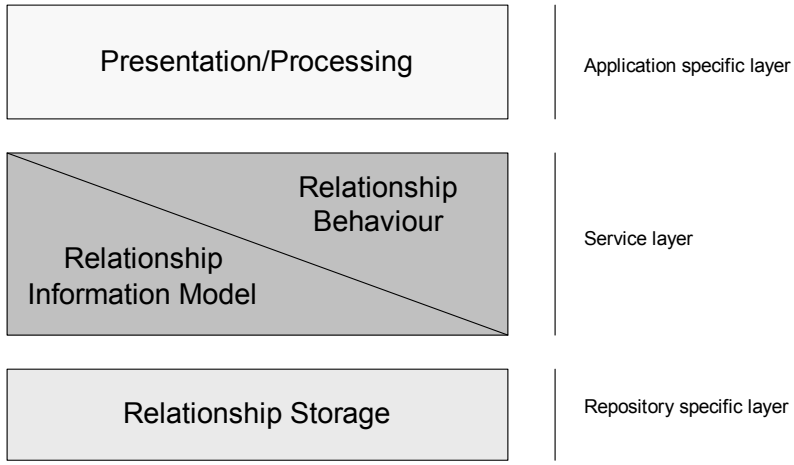


Figure 2.4: A service for relationships.

- The object-oriented nature of CORBA provides for a clear conceptualizing method for any domain of discourse – objects and the behaviour of objects. Explicit relationships are essentially distinct information units with their own identity, and the use of objects is an approach for modeling and implementing such information.
- CORBA supports the notion of persistent objects, which is a necessity if relationship information is to be represented as distributed "information objects".
- CORBA is tightly integrated with the programming environment. The development of CORBA based applications enables a focus on the application rather than on network messaging. This is an important feature in a research context (as well as in ordinary development), because it conveniently enables a high level of iterative development and experimenting with different solutions.
- The network transparency of CORBA enables a high level of location transparency and basically supports any degree of distribution, which corresponds well with this work's objective of exploring the distribution of relationship information and service.
- The CORBA environment has defined a set of services that are common across many different distributed applications, which means that commonly needed basic level functionality like transactions, security, etc., can be available "off the shelves" as components.

Although CORBA has been used as the infrastructure of several digital library projects, such as [160, 162, 217], the possible use of CORBA as the underlying infrastructure for digital libraries is not very well explored.

3 Relationship Knowledge

3.1 The Term and the Concept

When using the term *relationship* we assume that there is an implicit and common understanding of what this term actually means. Although most people have a certain understanding of this term, we nevertheless find that it is difficult to be specific when explaining its meaning. The actual understanding is often vague and sometimes even diverse. When exploring relationships – what they are and how they can be represented – there is however a need to be precise by using definitions or explanations that express or describe the essential nature of the concept.

When examining the actual meaning of a term, it is often useful to refer to Ogden and Richards' triangle of meaning depicted in Figure 3.1 [156]. The triangle of meaning is a model that distinguishes between *symbol*, *thought or reference*, and *referent*. The model states that the connection between a symbol and what the symbol refers to, is an indirect one that depends on the meaning that each person associates with the symbol.

A symbol has both intension and extension [17]. The extension of a symbol means the referent – the object or the set of objects in the real world to which the symbol indirectly refers. The intension of a symbol is its sense: that which a person normally understands by the expression. A major problem when discussing relationships in information technology is that neither the intension nor the extension is clearly defined. The term *relationship* is merely a symbol that denotes the idea of relationships – an idea that can be highly personal and domain dependent. The extension of this term is all the possible relationships that may exist – a rather large set since almost anything can be related to everything else in the real world. For this reason, an extensional analysis of relationships is a difficult task. An analysis of relationships rather has to be based upon the intension of the term – what is generally understood by this term and how this influences the expressing and processing of relationships in information systems.

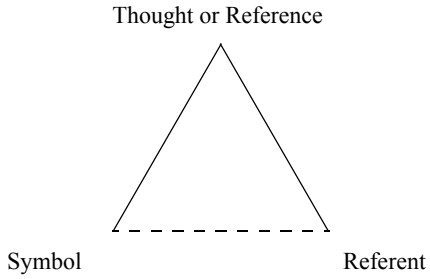


Figure 3.1: The triangle of meaning.

A main problem in human discourse is that we use different terms with the same intension (synonyms), or that we use the same term but with different intension (homonyms). This problem is highly relevant when examining literature and theories concerned with relationships. Terms like *association*, *relation,* and *relationship* denotes, in many cases, the same concept, while in other cases they represent different views or different concepts. When referring to ideas and solutions from different domains, this work will conveniently use the original authors' terminology, despite the problem that two domains may use different terms for the same concept. When referring to theories and solutions developed in this work, the term *relationship* is the preferred one. Correspondingly, the term *related* is used to describe the idea of relationship participation. If a there is a relationship between two objects, these objects are related to each other. The term *link* often occurs in computer science, and this work interprets *link* as the implementation of a relationship instance.

3.2 Relationship as Knowledge

Understanding human knowledge and reasoning has occupied scientists for centuries, but despite the efforts to understand knowledge and cognition there is no consensus understanding in this field. Rather, there is a set of theories emphasizing the various capabilities of human cognition and discourse where each theory is coloured by the context it is developed in and the purpose it serves. The concept of relationships is, however, present in many of these theories.

In the field of science philosophy as it is presented by M. Bunge in [27, 28], knowledge and meaning are defined by the notion of *concepts.* Concepts can be defined as a unit of thought, and accordingly the theory of concepts should be the

philosophical equivalent to the atomic theory. Like for the atoms of atomic theory, we are not able to see concepts, but concept theory exists because it is a convenient model of the world of human knowledge according to available evidence. Bunge makes a further distinction between different concept types and includes relation concepts as one of the four categories:

- Individual concepts
- Class concepts
- Relation concept
- Quantitative concepts

Individual concepts apply to individuals, whether definite (specific) or indefinite (generic), like the concept of the definite person "Einstein" or the concept of the mathematical indefinite symbol "x".

Relation concepts apply to relations among objects and can further be divided into *comparative* and *non-comparative* relations. A relation pairs the elements of two (or more) sets and comparative relations, like <, are relations that reflect a specific ordering of elements; 2 *is less than* 3, John *is higher than* Paul, based on some measurable quality of the individuals.

A comparable model of knowledge can be found in the field of artificial intelligence exemplified by Han Reichgelt's two main categories of knowledge in [170]:

- Domain knowledge
- Strategic knowledge

Domain knowledge is the information about a domain that a computer needs to manipulate to reason about this particular domain. Strategic knowledge, on the other hand, is knowledge about how to use the domain knowledge to solve particular problems in the domain. One can further distinguish between two kinds of domain knowledge:

- Structural knowledge
- Relational knowledge

Domain knowledge often has a very specific structure that consists of entity types possibly arranged in a classification hierarchy – structural knowledge. Relational knowledge is concerned with the relations between the entities that are distinguished in the structural knowledge.

Statements comparable to Bunge's analogy to the atomic theory can be found in the artificial intelligence discipline as well. According to Davis et al. in [52], all knowledge representations are surrogates of the real world. Reasoning, whether it is in a program or in the human mind, is an internal activity, while most things that are

reasoned about exist only externally. Representations are imperfect approximations to reality, and each approximation attends to some things and ignores others. In selecting representation, we are necessarily making a set of decisions about how and what to see in the world.

The above examples from science philosophy and artificial intelligence both indicate that relationships are a fundamental category of knowledge that is different from the knowledge we have about atomic entities. Ronald W. Langacker argues in [122] that this distinction is reflected in language itself by the distinction between:

- Nominal predications which designate things and correspond to nouns.
- Relational predications which designate states and processes and correspond to adjectives, adverbs, prepositions, and verbs.

Relationships may be considered a fundamental aspect of knowledge in the sense that it enables higher-level, coherent and complex knowledge to be constructed from the knowledge we have about singular entities. The existence of relationships as a specific category of knowledge, however, is not based on physical laws or neutral observations, but is determined by our inclination to categorize and distinguish between different types – in this case different types of knowledge. The distinction between relationships and the entities they relate is fundamental in many fields of computer science. This is reflected in information and knowledge representation models like:

- The nodes and arcs of semantic nets
- The terms and propositions of the first order predicate calculus
- The entities and relationships in data modeling
- The objects and relationships in object-oriented modeling

This is equally important for the field of digital libraries where we may distinguish between the basic entities that we deal with and the numerous relationships that may hold between them. Relationships are the main enablers for building larger knowledge or information structures from the more atomic entities of documents and other content objects.

3.3 Where Do Relationships Come From?

Despite this presence of relationships in knowledge models, there is no well-accepted account for the origin of relationships. What are relationships made of? How are they made? This is emphasized by Gasser et al. in [67] by the presentation of five common facts about how relationship knowledge is adapted and interpreted by humans:

- *Fact 1: Language matters.* The relational knowledge we develop is influenced by language. The fact that different languages contain different spatial relation concepts is one proof of this. The German language distinguishes between *an* and *auf* where the English uses the single term *on*. The Korean language, on the other hand, uses the concept of tight fit and loose fit. These and similar discrepancies between the relational terms of different languages show that the various relational concepts are learned, and it seems clear that the individual languages determine the possible relations that can be expressed.
- *Fact 2: Object categories are easier and earlier than relational categories.* In the process of language acquisition children tend to learn object terms earlier and more easily than relation terms. This is not evident in all languages, but all in all there is a bias towards learning nouns over relational terms in early word learning. Studies on other aspects of language acquisition show that common categories are for the most part trivial for children to acquire whereas relational terms exhibit a protracted and erroneous course of development.
- *Fact 3: Understanding relations is dependent on the specific objects entering into those relations.* Children's attention to relations is at first highly dependent on the objects involved but becomes less so with development. The understanding of relational terms is an understanding that develops from a specific similarity-based knowledge at an early stage and then towards a more abstract understanding across diverse kinds of objects and settings.
- *Fact 4: Object properties are relevant to understanding relations.* Studies have shown that object properties matter when people make relational judgments. The real-world use and recognition of relations requires an understanding of the objects that participate in the relationship.
- *Fact 5: Relational concepts have a category structure.* Research indicates that relational concepts seem to be like object concepts in having a graded similarity structure. Studies of how both children and adults perceive relations like higher and lower show that some instances of a relation are better instances, even with respect to relations that do not appear to be graded.

Together, these facts reflect some general observations of how relationships are learned and interpreted by humans. One general conclusion is that there is no predefined set of relationships that are globally understood. Relationships are highly influenced by learning which for instance should imply that relationship mechanisms need to be support ways in which users can learn the meaning of unfamiliar relationships. Another observation is that the understanding of a relationship is depending on the participants of the relationship, which should imply

that first-class relationships need to include relevant information about the participants if this information is unavailable by other means.

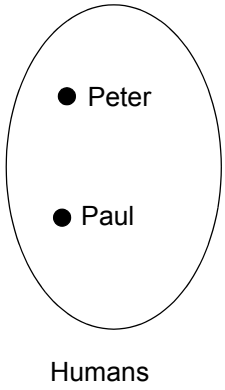
3.4 The Mathematical Relation

One contribution that strongly has influenced the notion of relationships in computer science is the mathematical relation which can be viewed as a formal abstraction of order among things. In contrast to other approaches, we find that in mathematics the relation primitive is defined in terms of other more fundamental elements in the mathematical set theory [206].

A relation is a set of ordered sets. In this context, ordered means that the elements of the set are positioned. The first coordinate stands in a particular order to the second etc. Ordered sets are usually referred to as tuples, and the size of the tuple is referred to as the degree. A binary relation is a set of ordered pairs, a ternary relation is made up of ordered triples, etc. For sets $A, B \subseteq U$, any subset of the Cartesian product $A \times B$ is called a relation from A to B. Any subset of $A \times A$ is called a binary relation on A. Whereas the Cartesian product represents all possible unique combinations with one element from each set, the relation is a subset that reflects the actual truth conforming combinations. A relation corresponds to the extension of a predicate. Given a domain of discourse containing all humans, called H, the Cartesian product $H \times H$ is formed. The predicate $F(x,y)$, interpreted as *x is the father of y,* is true for certain ordered pairs in $H \times H$ and false for others. The set of ordered pairs for which $F(x,y)$ are true is called the extension of the predicate $F(x,y)$ or the relation of fatherhood in *H*. A function is a specific kind of relation. A relation F in $A \times B$ can be called a function when every member of A occurs exactly once as the first coordinate in the set of ordered pairs of the relation from A to B.

A mathematical relation does not necessarily correspond to a relationship as a conceptual knowledge primitive. The coordinates of a tuple may be attribute values from different domains used to describe an entity, in which case each tuple represents an entity, alternatively the coordinates may be keys – entity identifiers – in which case the relation reflects a relationship. This is quite evident in the implementation of data using database relations. The database relation is a lower level structure that can be used to represent many kinds of information in an organized way.

In mathematics a relation is said to specify the set of tuples for which the relation holds. Each tuple can be interpreted as instances of the relation. In mathematics, however, the set *is* the relation. When discussing relationships in the context of information systems a comparable distinction can be made between the relationship



The Cartesian product:

Peter	Peter
Peter	Paul
Paul	Paul
Paul	Peter

$C = H \times H$

The relation of fatherhood:

Peter	Paul

$Fatherhood \subseteq H \times H$

Figure 3.2: Relation and Cartesian products.

as a class-level definition or set and the instances of the set – relationship instance. The Unified Modeling Language defines the individual connection among two or more objects as a *link* – a tuple of object references [176]. This corresponds to how a link generally is understood in other disciplines such as hypermedia. Links, however, are not always explicitly belonging to a particular class or set other than the universal set of all links.

3.5 The Relationship as a Modeling Primitive

The introduction of the relationship as a distinct modeling primitive in computer and information science can be traced back to the emerging of the semantic data modeling languages of the 1970-80's. Unlike the earlier data models, such as the hierarchical data model and the network model, which mostly were used to specify the structure of data for storage in record based file formats, the essence of semantic modeling is that the model should reflect the way humans conceptualize a domain. To achieve this many modeling languages have introduces a relationship modeling primitive, and the ability to describe relationships is by several described as one of

the main characteristics of semantic data models [164, 166]. A major contribution was given by P. Chen with the introduction of the Entity-Relationship model [37], which distinguishes between entities and the relationships they participate in.

In semantic modeling languages and the later object-oriented modeling languages, relationships are supported by specific primitives. One example is the Unified Modeling Language which defines different relationship primitives including aggregation and generalization. The expression of user defined relationships is supported by the generic association primitive. Relationship modeling follows a comparable pattern in many languages. Aspects of a relationship commonly expressed in models may include:

- *Related entities;* Entities or classes are in most cases prior and the relationships are specified between specific entities or classes.
- The relationship *degree* is the number of participating entities or objects of a relationship. A relationship between 2 entities – by far the most frequently occurring incident – gives a relationship with the degree of two; a binary relationship. Relationships, as well as mathematical relations, can however include more than two entities, e.g. a ternary relationship that relates three entities. Any degree higher than two is often referred to as relationships of n-ary degree.
- *Cardinality*, also called multiplicity, is another characteristic used when modeling relationships. The cardinality constraint indicates the number of relationship instances that a given object is allowed to participate in. If an object only can participate in one relationship of a specific kind, the cardinality constraint is 1. Other numbers may be used to specify other cardinality constraints. If the number is undefined, the cardinality constraint is often said to be N.
- *Participation* is a constraint that is used to express whether the entity is required to participate in a relationship or may exist within the system without participating in a relationship. It can either be expressed using specific notational symbols for this purpose, or it can be combined with the cardinality constraint by the specification of a cardinality ratio. The cardinality ratio of 1..N expresses that an object has to participate in at least one relationship. The participation constraint 0..N defines that relationship participation is optional.
- *Names* are often used to express the meaning of a relationship. A single name for the relationship can be used and rolenames can additionally be used to enrich the relationship with additional semantics that describes the participant of the relationship.

- Relationship may additionally be defined with *class-like capabilities*. The difference between a relationship and an entity is not a very distinct one, and intermediary solutions may include the modeling of relationships that include attributes and methods of their own.

A more comprehensive support for relationship modeling can be found in other languages like the Referent language for conceptual modeling [186]. This language supports the formal definition of a rich set of conceptual structures and includes the ability to model relational compositions and derived relationships.

In the context of modeling languages, the relationship primitive is merely one out of several primitives. Together these primitives make up the vocabulary that is used when capturing and describing the domain of discourse in the software engineering process. The actual usage of these primitives can, however, be diverse. A marriage may be defined as a relationship between the two involved persons, but a marriage may also be defined as a class. Choosing between the relationship and the class primitive to represent a specific kind of information is, in this case, a design issue.

A more extreme approach can also be used in modeling by rephrasing other commonly used constructs as relationships. Compound objects and collection objects like lists, bags or sets can be interpreted as specific relationship constructs, with each object as a participating member in a relationship. The objects participating in an aggregation are typically of different types, whereas for a collection object the participating objects are typically of the same type.

3.6 The Meaning of Relationships

The emphasis so far has been on the general concept of relationships. However, in real-world applications the focus is on specific relationships with specific semantics. The mere notion of things being related to each other is quite useless without knowing the meaning of the relationship, as pointed out by Rebecca Green in [73]:

> *To specify a relationship, we must be able, first, to designate all the parties bound by the relationship and, second, to specify the nature of the relationship.*

In mathematics, the relational operators like equals (=), less than (<) or greater than (>) express specific and precise meaning that is well understood by those who are familiar with numbers and mathematics. The ideal solution for any deployment of relationship knowledge is to be able to interpret relationships at the same level of preciseness. This is, however, far from the situation when it comes to common

knowledge and information. As argued by Gasser et al. [67], there is no universal set of innate relations hard-wired into biology. The relationship vocabulary is determined by the language and the interpretation of a relationship is depending on the objects involved. Relationship types that are considered fundamental in modeling languages include categories like:

- Classification and instantiations which are used to specify the relationship between an entity and the class that this entity belongs to.
- Generalization/specification which is used to express the relationship between classes where one class is a more general super-type and the other is a more specific subtype.
- Aggregation and composition that are used to express the relationship between the higher level whole and its subparts.
- General associations that are used to model application specific relationships.

The development of relationship vocabularies in other domains may conclude with other fundamental relationship categories, or the interpretation of what constitutes the fundamental categories can be a variation over the same theme. In thesauruses, relationships are used to express the various dependencies and connections between the terms of the thesaurus. Three general classes of fundamental relationships have been established [39]:

- The equivalence relationship that denotes the relationship between a preferred term and the non-preferred term.
- The hierarchical relationship that represents pairs of terms in their superordinate or subordinate status. The superordinate term represents the whole and the subordinate represents the part. This includes both whole/part relationships and categories of generalization hierarchies.
- The associative relationship denotes the relationships between terms that are neither hierarchical nor equivalence.

In domains related to digital libraries there have been quite many attempts to define typologies of application-specific relationships. The ACM/SIGIR[1] workshop "Beyond Word Relations" examined a number of relationship types significant for information retrieval systems and concluded with the following list of relationship types [94]:

- Word-based relationships that denote documents that share the same vocabulary or word.

1. Association of Computing Machinery/Special Interest Group in Information Retrieval.

- Attribute-based relationships that denote relationships based on shared characteristics as same author or same place of origin etc.
- Document-document hierarchical relationships that denotes situations where one document is a sub-set or super-type of the other.
- Document-document topological relationships that are conceptual extensions to the hierarchical relationship and include relationships that denote conceptual equivalence, sequences, etc.
- Document-to-document influence relationships that denote relationships where one document has affected the writing of another.
- Topic-based relationships that exist between documents that are related through less obvious topical resemblances.
- Usage-based relationships that exist between documents that are related through the use of the documents.

The field of relationships between bibliographic entities has been examined by Tillet in [196, 197] and by Leazer in [125]. Tillett identifies the following categories:

- Equivalence relationships which hold between exact copies of the same object.
- Derivate relationships which hold between the original and the modified version.
- Descriptive relationships which hold between a bibliographic entity a description, criticism evaluation, or review.
- Whole-part relationships which hold between a bibliographic entity and a component part of that entity.
- Accompanying relationships which hold between a work and the accompanying material.
- Sequential relationship which holds between bibliographic entities that continue or precede one another.
- Shared characteristic relationships that hold between entities that happen to have the same subject, author, etc.

A rich set of bibliographic relationships partially based on this is in the bibliographic information model proposed by the IFLA Study Group on the Functional Requirements for Bibliographic Records [100].

A different discipline that has a comparable concern in relationship typologies is hypermedia. Several taxonomies for hypermedia link types have been proposed and overviews are given by Kopak in [119] and by Verbyla in [203]. A hypermedia link

is essentially only a connection between hypermedia nodes. A link type is needed to express the semantics of the link and in this way enable the link to become a meaningful relationship. One early attempt to define a link typology for hypermedia can be found in the TEXNET system [199] which included a list with over seventy different link types. This extensive list was motivated by the desire to provide for a sufficiently rich set of link types that users could select among, and to prevent the need for users to create new user-defined types. A more general categorization of link types is identified by Lisa Baron in [131]. She identifies two general types of links:

- Organizational links that are used to describe the "surface" structure of documents, for example the links used to organize the presentation of documents; links to and from the table of contents, and previous and next links to browse sequential structures.
- Content-based links that deals with the relationships between nodes of the text.

  Content-based links can further be categorized as:
- Semantic links that describe the semantic association between words or concepts.
- Rhetorical links that are used by the author to lead the reader through a sequence of information elements.
- Pragmatic links that are concerned with practical results like warnings and errors.

A different concern in the field of relationship semantics is *how* to express the meaning of a relationship. When modeling a relationship it is often labelled with a single term that denotes the intended semantics of the relationship. The mere purpose of this label is to capture and signal the semantics of the relationship to the principals and to the implementers.

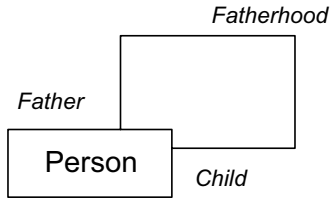


Figure 3.3: Relationship naming.

Certain modeling languages introduce a richer semantic typing of relationships by allowing for roles to be associated with the participants of a relationship. If two persons are related to each other by the *Fatherhood* relationship, it is difficult to determine who is the father and who is the child without any further knowledge of the persons. The role type denotes the role the participant plays in the relationship, e.g. that one participant is the *father* and that the other is the *child* as illustrated in Figure 3.3 The use of more complex typing of links has been explored in the field of hypermedia as well [143]. A different convention found in both modeling languages and hypermedia is the use of direction to indicate the origin and the target of a relationship.

As pointed out by Gasser et al., the understanding of relations is dependent on the specific objects entering into those relations. This implies that a relationship type can be insufficient for expressing the full semantics of a relationship. In addition to rolenames, there can be a need for even more information about the participants if information about the objects of the relationship is unavailable or insufficient for the proper interpretation of a relationship.

4 Relationship Representation

4.1 The Implementation of Relationships

The ways relationships are actually represented in information and software artefacts are highly diverse. Different application areas emphasize different features and different environments offer different primitives that can be used to express relationships, examples are:

- Textual descriptions of relationships in text documents.
- Metadata elements in metadata formats.
- Keys and foreign keys in database relations.
- Embedded references and external link objects in hypermedia.
- Object attributes holding pointers to other objects in object-oriented systems.
- Specific purpose relationship mechanisms in object-oriented systems.
- Predicates in symbol manipulation languages.

This review starts with certain general characteristics of relationship implementations followed by a review of more specific solutions that are in use or have been explored in object-oriented systems, hypermedia and descriptive solutions. The selection, however, is not complete, but it is based on examples that are considered to be relevant as relationship mechanisms in digital libraries.

4.2 Extensional and Intensional Relationships

A first major distinction that can be made between relationship mechanisms is whether relationships are implemented by *extension* or *intension*.

An *extensional implementation* is based on explicit information about existing relationship instances. This can be tuples of entity keys stored in a database relation,

the use of references in object-oriented systems, the use of hypermedia links in documents, or any other information representation that can be used to store relationship information in an explicit form. In such cases, the relationship implementation can be defined as extensional because the set of truth conforming relationships are defined by the extension of the relationship concept.

The contrary to an extensional implementation is an *intensional implementation*. The intension of a relationship may be defined through rules that express when an asserted relationship is true or not. In digital libraries this solution can support the discovery of relationships within or between defined sets by determining the truth of an asserted relationship given two (or more) entities as input. Examples are the evaluation of equivalence between two or more text documents and the evaluation of shared characteristic relationships between documents.

These two distinctions are quite different in nature and serve different purposes. They may, however, coexist within the same system and can be used in combinations or as complementary solutions. The motivation for one or the other may be different in different environments based on both formal and more pragmatic aspects. In general the extensional solution will be used to capture relationship information that otherwise would not be present within the system. Intensional implementation is more likely to be used if relationships are to be discovered dynamically based on some well-defined evaluation criteria. Comparative relations are typical candidates for intensional relationship implementations.

Documents and the various relationships between documents may serve as an example of both of these two implementation techniques. For documents that share the same topic, there is a certain topical equality between them that can be useful for the users of an information retrieval system. If they find one relevant document, they may have an interest in other documents on the same topic. Topical equality may be computed automatically by comparing the terms and term occurrences of documents in the set, such as information retrieval techniques like document clustering [168]. Another kind of relationship detection mechanism is citation extraction, where scientific articles are examined for citations and references to other articles [123].

However, the field of document relationships consist of numerous relationship types that will be hard to express as rules and implement as methods. Relationships like one document being the translation of another or one document being a revised version of another can be difficult to express as they may rely on external knowledge.

The domain of document relationships can also illustrate other pragmatic considerations of extensional versus intensional relationships. The evaluation or extraction of intensional relationships introduces a significant processing overhead

compared to an extensional approach. It is less likely that a system is capable of performing such evaluations continuously at runtime without significantly reducing the overall performance of the system. A more likely solution is to perform these computations once and then store the discovered extension of the relationship. In other situations, the cost of storing the relationship extension may by far exceed the cost of computing the relationship. An example of the latter is the general mathematical relation of $x < y$. Determining whether the *less than* relationship is true for a given set of numbers can efficiently be computed. Solving this by the use of a relationship extension would, on the other hand, require an infinite amount of storage space.

4.3 Property or First-class Object

Another distinction that can be made between relationship implementations is whether a relationship is implemented as in integral part of the participating entities or whether the relationship is implemented as a distinct and first-class information structure separated from the participants. In certain environments this distinction is of minor importance, such as in relational databases. A typical mapping of the relationships expressed in an E-R model to a relational storage structure is to use a separate relation for N:M relationships and otherwise add foreign keys to entity-relations for 1:1 and 1:N relationships [62]. Due to the easy access to data and the flexibility in which data can be queried and retrieved in such storage systems, different relationship implementations do not introduce significantly different application features.

In other systems such as object-oriented databases and hypermedia applications this distinction is of greater importance because it reflects quite different solutions with different strengths and limitations. In object-oriented systems, objects are the major implementation construct for representing real-world entities, and the use of a property to hold a reference to another object is often the only mechanism for implementing a relationship instance. On the one hand this is an efficient mechanism because it is easy to implement. On the other hand this can be a limited solution because relationships only can be accessed and manipulated through the participating objects. There is no inherent mechanism for managing and manipulating the extension of the class-level relationship, and a bi-directional relationship need to be implemented with a corresponding inverse reference on the opposite object. The use of properties and references to implement relationships is to some degree based on the assertion that relationships not are real-world entities, although it can be argued that relationships represent real-world entities that are

intangible and as such they should be considered as candidates for objects equal to other real-world entities [150].

The distinction between relationships as property or first-class object is additionally quite evident in hypermedia solutions where the use of links embedded within the content is considered to be quite different from the use of first-class link objects [45, 48]. The main advantage of embedded links is that the hypermedia document and the links from this document form a self-contained object that easily can be moved or edited. Embedded links do not require any specific support for storing and managing relationships, but the expressed relationship is maintained and used along with the document. The disadvantages are comparable to the use of references in object-oriented systems: links are inherently one-way, only the document that contains the link knows about the link, and access to the link requires accessing and parsing the document. The use of first-class link objects is a different tradition in the field of hypermedia. The advantages of storing links external to the documents they link between is; the inherent support for bidirectional links, the possibility to use third-party applications for creating and accessing links, and the dynamics of this solution that enable applications to retrieve and display only the links that are relevant for a particular user or in a particular context.

Properties can be used to express both the aspects of an entity and the relationship that the entity participates in. As a knowledge primitive, however, the relationship is often perceived as a connection or association between equally important entities that each has a certain identity of their own. A value that merely describes an aspect of an entity on the other hand is subordinate to the entity it describes and should not be considered a proper relationship.

4.4 Relationships in Object-Oriented Systems

4.4.1 Object-Oriented Systems

The introduction of object orientation is one of the major achievements in modern computing. The notion of object and classes, as they are used in the object-oriented paradigm, is comparable to Bunge's individual and class concepts, and object orientation is often promoted as a preferable abstraction when modeling and implementing information systems, because it correspond to how we actually interpret the real world [18, 216].

Although the initiation of object orientation can be related to programming languages [15], the object-oriented paradigm has later been adopted by many other disciplines as well, like object-oriented databases and object-oriented analysis. Object-oriented analysis is based on the assertion that it is intuitive to describe a

given domain of interest as a set of interrelated objects that encapsulates information and behaviour. Furthermore, using the same conceptual model in the analysis, design and implementation ensures an easy and seamless transition between these phases of systems engineering.

In practice, object-oriented development is not as seamless as it appears. Object-oriented programming languages do not provide support that corresponds to the relationship primitive of the analysis and design (except for the inheritance relationship), and rather relies on this to be incorporated manually into the objects using properties and references. This solution is problematic for a number of reasons [21, 118, 144, 150, 175]:

- *Traceability:* At the design level relationships are represented using explicit primitives for this purpose. Since this usually is implemented in an ad hoc fashion in the code, the relationship abstraction is often lost in the transition from design to implementation, which leads to a traceability problem. It may be hidden in implementation details and/or aspects of the relationship can be distributed over different classes.
- *Complexity:* Implementing relationships by the use of object references can introduce a significant complexity in the code. Behaviour and information related to the relationships need to be supported by all class implementations that instantiates objects that potentially will participate in relationships.
- *Duplication of information:* For reference-based relationships each object is the master of the relationships it participates in, and relationship information only exists as part of each object. A unidirectional relationship requires an inverse reference on the opposite object. This inevitably leads to duplication of both information and behaviour, and dispersion of relationship knowledge across objects.
- *Lack of reuse:* Relationships and associated methods embedded and concealed in class implementations disable the reuse of generic mechanisms for relationship behaviour and attributes. Applying a generic operation to a relationship or set of relationships requires that the relationships are uniformly implemented.
- *Management problems:* The access point for reference-based relationships is through the objects of the relationship. The creation and deletion of relationships require access to the objects as well as knowledge about how this is performed for all classes. This disables management and access to the extension of the relationship that otherwise would have enabled the selection of subsets of the extension or the iteration over the entire extension.

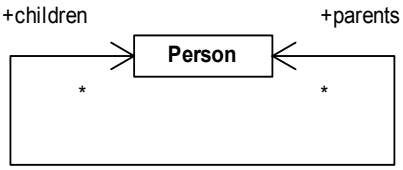
The general solution proposed by many is to enhance the software environment with a specific mechanism for relationships. One of the first contributions on this was

given by Rumbaugh in [175] and later contributions have explored numerous solutions including solutions where the relationship mechanism is transparent to the application and solutions where relationships are explicit and first-class objects that directly can be accessed and manipulated. Relationship mechanisms are often explored as a feature of object-oriented databases, but the support for relationship mechanisms as an implicit part of object-oriented programming languages has additionally been explored.

- *DSM* (Data Structure Manager) [182] is a programming environment that adds a number of extensions to C, including object orientation and the support for inter-object relationships. Relations are used to represent the relationships between objects and are declared in parallel to classes. Rolenames are defined for the participating classes, and each class is extended by generating relation access methods using the rolename as a pseudo-attribute. The relationship construct supports the definition of binary relationships and the participation constraint for the relationship.
- *MOPS* (Metaobject Protocol for Reflective Systems) [117, 118] provides for a full-fledged relationship support implemented in the Common Lisp Object System. Metaclasses are used to endow relationship classes and relationship instances with properties and behaviour. Relationships are defined with minimum and maximum cardinality constraints and the system supports derived relationships.
- OORM (Object-Oriented Relationship Data Model) [98] relationships are represented as separate classes similar to other classes representing entities. The relationship class is specified both by listing the participating classes and by defining constraints, attributes and methods of the relationship.
- *Adam* [57] is an object-oriented database system that provides support for metaclasses that can be used to support application-defined relationships of n-ary degree with constraints, attributes and behaviour.
- *FOOD* (Fully Object-Oriented Database) [24] is an object-oriented database with support for relationships. A relationship is defined as a tuple of roles and attributes, and the role definition consists of a name, a type and a cardinality. A relation is considered to be a set of relationships of the same type and a single relation object exist for each relationship type. Subtypes of relationships can be defined and are contained within the relation object of the parent type. The relation objects have system-defined operations to manage and query the relationships.
- *The object data model of ODMG* (The Object Data Management Groups) [36] is a standardized object model with for object-oriented databases. The object data model defines a specific relationship property for binary relationships. The names and traversal paths for a relationship can be specified by the

use of the object description language (ODL) as shown in the example of Figure 4.1.

- *OSCAR* [179] is an object-oriented database system which uses relations to model relationships. A specific construct is used to specify relationships that include the definition of cardinality constraints and relationship attributes. Each column of the generated relationship relation has a type and the columns of class type are called roles and hold the objects that are related to each other. Other columns are used to store attributes of the relationship.
- *SADES* (Semi Autonomous Database Evolution System) [167] is an approach to relationships in object-oriented databases that supports dynamic relationships between classes. Relationships between classes can be changed at runtime and do not need to be fixed at compile time.




```
class Person (extent persons){
        attribute string name;
        relationship set<Person> parents  inverse Person::children;
        relationship set<Person> children inverse Person::parents;
 }
```

Figure 4.1: Relationships in the object data model of ODMG.

As the above examples show, the field of relationship mechanisms for object-oriented systems is highly diverse with respect to how the relationship mechanism is implemented and what features it support. With the exception of the ODMG standard, these systems are mainly research projects and current practice in commercial applications is still generally based on the use of reference-based mechanisms. This can be due to the dominance of programming languages that do not support a specific relationship construct (like C++ and Java), but may additionally be caused by other factors, like the assumption that a relationship construct will be less efficient in terms of processing speed and capacity. Another reason can be that the gains of using such solutions are less obvious in the implementation of software but more prominent in the context of maintenance and reuse.

4.4.2 The OMG CORBA Relationship Service

Support for relationships can also be found for distributed objects. As a part of the general framework for distributed object applications, OMG has defined various services that implement features that are commonly required for distributed object applications. These are generally referred to as the *Common Object Services* or CORBA Services, and include a service for defining and managing relationships between distributed objects – the *CORBA Relationship Service* [77].

The basic architecture of CORBA defines object references that clients use to issue requests on objects. These object references can be stored persistently and can be used in reference-based relationships. The Relationship Service is a different solution and is specified to support applications that need capabilities like:

- Multidirectional relationships
- Creation, deletion, and manipulation of relationships between third-party objects
- Traversal of graphs of related objects
- Relationships that can be extended with attributes and behaviour
- Relationships with relationship-specific semantics

The Relationship Service of CORBA allows for relationships to be explicitly represented using a set of object instances that together represent the relationship instances in an application. The specification defines three levels of service: base, graph, and specific.

The base level defines the interfaces for the role and relationship objects that implement a relationship. Relationships are navigated by invoking the methods of the role and relationship objects. Role and relationship objects are created when needed by the use of factory objects. A role object represents an entity in a relationship and is created by passing the reference of the entity object to a role factory. A relationship object, on the other hand, aggregates the participating roles of a relationship and is created by passing a set of roles to a relationship factory.

The Relationship Service allows for relationships of arbitrary degree. A relationship can contain two roles as in a binary relationship, or it can contain three or more roles and thus support relationships of higher degree. A role can participate in any number of relationships, only constrained by the minimum and maximum cardinality. Roles and relationships can further be constrained by the type of objects they expect. The constraint mechanism is an integrated functionality of the object implementations and erroneous situations are handled by raising exceptions.

Since a role object only holds a reference to the entity object, it may represent a third-party object and in this way allow for relationships to be created without inferring with the related objects.

The graph level extends the base level by adding the concept of a node object as well as other interfaces for graph traversal. The purpose of the node is to tie the various relationships into a graph by aggregating the roles of an entity object. The node interface may be inherited by entity objects or it may only contain a reference to the entity object it represents.

Specific relationships are defined at the third level by the definition of interfaces for the two important relationships *containment* and *reference*. Additional application specific types of relationship and role objects can be provided by inheriting the basic interfaces, and this also enables the implementation of relationships with application-specific methods and attributes.

The use of the Relationship Service in applications comparable to digital libraries is reported in [183] and [116], and in the initial phase of this project the CORBA Relationship Service was evaluated to determine whether this service was appropriate for supporting relationships in digital libraries. The features of the service related to a potential use in this context can be characterized with the following:

- The service defines a well-considered object model that can support all kinds of relationships with respect to semantic types, structural features like degree, and constraints on cardinality and type. The service allows for relationships between third-party objects to be defined.
- The service assumes a predefined and static information model where the different kinds of relationships have to be specified in advance, prior to the implementation of the application. For each kind of relationship there is a need to define and implement specific interfaces for role and relationship objects as well as their respective factories.

The main motivation for this evaluation[1] was to explore the potential use of the CORBA Relationship Service in a digital library environment. This is one of the few existing open solutions for a relationship mechanism that is based on established middleware. The potential advantages of this service are that the object model it implements is a well considered solution for relationship representation. Non-CORBA entities such as the information objects of digital libraries can be related for instance by the use of an intermediary object with a URI that references the actual information object. This means that the object model of the service

1. The evaluation was based on available implementations of the service and partly by implementing the base level of the service. The CORBA Relationship Service is available as part of the MICO CORBA implementation (http://www.mico.org) and has been available earlier from PrismTechnologies (http://www.prismtechnologies.com/).

potentially is highly usable to represent and navigate relationships between the information objects in digital libraries.

The current design of the service, however, only supports static definitions of relationship types. Additional relationship types other than the specified *containment* and *reference,* needs to be specified and implemented. The service is thus more a toolkit for adding relationship support to applications rather than a service for managing and using arbitrary relationship types. The use of a specific factory object for each object type is an additional disadvantage. With a large number of relationship types this will introduce an undesired level of complexity. A final problem discovered is an ambiguous use of IDL types and named objects. Objects both have an IDL type and additionally need to be assigned names. These names can potentially be used to create a more flexible relationship typing mechanism, but this is contradicted by the lack of support for dynamic definition of cardinality and degree.

4.5 Hypermedia Systems

4.5.1 Hypertext and Hypermedia

The term *hypertext* was initially introduced by Ted Nelson to describe a system for non-linear reading and writing of text [146]:

> *By "hypertext" mean nonsequential writing - text that branches and allows choice to the reader, best read at an interactive screen.*

In computer science, hypertext has become a metaphor for interlinked content where users can view and navigate between nodes of information in an interactive way. Vannevar Bush is often referred to as the originator of hypertext, represented by the associative trails of the future system that he described in 1945 – the memex [29]. Early forerunners of modern hypermedia systems include the HyperText Editing System (HES) developed at the Brown University in 1967 [33] and the oNLine System (NLS) introduced by Douglas Engelbart in 1968 [63].

With the introduction of multimedia support in computers, hypertext technology is naturally expanded to support content other than text, and the broader term *hypermedia* is sometimes preferred, although hypertext and hypermedia are interchangeably used and should be considered as synonyms. The development of hypermedia solutions includes a range of challenging problems, such as the modeling and representation of the nodes and links that make up the hypermedia application (the hyperbase), the storing, addressing and retrieval of multimedia information, authoring and reading hypermedia information, interacting with hypermedia information in user interfaces, etc.

4.5.2 Linking

Linking has always been the heart of hypermedia, and a link is often said to describe or express a relationship [34, 129, 173]. Although the links in hypermedia applications basically are used to represent inter-linked text and multimedia content, the more expanded notion of hypermedia intersects with other applications that utilize relationship information. Isakowitz et al. describe hypermedia as a vehicle for managing relationships among information objects [113]. An even more visionary approach is the field of structural computing [151, 152, 153] where the basic idea is to generalize the concept of hypermedia linking as a structural abstraction, and to support the need for structure in general rather than focusing on links primary as a vehicle for reading information.

Unfortunately, hypermedia infrastructure is not a uniform platform. A major distinction can be made between the use of links implemented as embedded references within the information, such as the use of anchors in HTML documents, and the use of links that are external to the content. Advanced hypermedia solutions often utilize external links and the benefits of this arrangement are [47, 87]:

- Links that can be followed in both directions.
- Links can be created between read-only information.
- Alternative link sets may exist for the same information, e.g. created by different users or for different usages of the same information.
- Third party applications can create, access, use and process links.

Hypermedia infrastructure has evolved significantly from the monolithic applications of the earlier systems that included all the information and functionality within one tightly integrated application [212]. The first major step was the introduction of client/server based solutions that separated the client application that was used to author or read the hypertext from the server process that was used to store and serve documents and links. The later generation of hypermedia infrastructures have explored more open and component-based solutions where the functionality is further decomposed. This often includes a separate process that manages the content, a different process to manage the linking information (the link server), as well as other processes to server other functionality. The Flag taxonomy [159, 215], which often is used to depict the openness and interoperability of hypermedia application, distinguishes between:

- *Open hyperbase systems* that provide integrated content storage and structural services. Examples are Hyperdisco [214], Sepia [188] and HURL [96].
- *Link server systems* that provide only structural services. Examples are Sun's Link Service, Multicard, Microcosm and Chimera, which will be further described in the following.

One of the earliest systems to provide for linking as a separate service is the *Sun Link Service* [163] that includes a protocol specification and a link server program. The link server only stores representation of nodes, rather than the nodes themselves. Links are explicit and bidirectional relationships between nodes, and applications can create and retrieve links by using the Link Service protocol. Applications that integrate with the link service register the class of nodes they are able to handle, and stores in the link database unique keys for their objects (node representations) along with the class the node belongs to. This enables the link server to identify what registered applications can handle requests about each object.

Multicard [171] adopts a similar approach to the Sun Link Service by defining a protocol for an open hypertext system, but differs in terms of the more powerful environment it provides for storing and managing hypermedia objects. Multicard distinguishes between node structures which are handled by Multicard and node content which can be handled by different editors. Nodes can be grouped in a hierarchical way. Anchors represent specific portions of the content of a node and carries links, scripts, and other hypermedia properties. Links in Multicard are viewed as event/message communication channels between two endpoints and the supported messages includes an activation message which typically will open and map the destination object. Link endpoints can be anchors, nodes or groups. The M2000 protocol defines the basic access mechanisms in a storage independent way and the actual storage management is implemented trough a back-end mechanism.

Microcosm [46, 87, 88, 49] distinguishes between the application layer, the link service layer and the hyperbase storage layer. The application layer consists of programs that are used to view and edit information and programs can be integrated with the link service in different ways. All linking functionality of Microcosm is provided by filter processes. Filters can respond to the message it receives or the message can be passed on to other filters. This enables a dynamic system; filters can be installed, removed or the chain of filters can be rearranged. A typical filter is the linkbase, which provides a database of available links. The use of filters as the units of distribution in a distributed version of Microcosm is explored in [97].

Microcosm maintains a database of all files it is aware of, known as the Document Management System (DSM). The DSM stores information about the document which is not known by the operating system and associates a unique identifier with the document. The unique identifier is the key used in the link base, and the DSM is used to resolve from the unique identifier and to the file name when a link is navigated.

Chimera [7, 8] uses the concept of a hyperweb which is defined as a collection of objects, viewers, views, anchors, links, clients and users. An object is a named, persistent entity and its internal structure is unknown and irrelevant to Chimera.

Viewers are entities that display objects, and the association between an object and the viewer is defined as a view. Anchors are defined and managed by viewers in the context of a view and tags some portion of a view as an item of interest. Links are sets of equally related anchors which imply a support for multi-directional relationships of higher degree. A client is program that contains one or more viewers. Chimera implements a server process that clients can communicate with trough RPC requests. The mapping between the concepts of object, view and anchor is a responsibility of the viewer which needs to utilize the Chimera API.

Relationships between entities are major elements when modeling and implementing all information systems, and in a very generic sense hypermedia links are not different from other applications in this matter. Although applications of hypermedia span a variety of uses, a common feature is the presentation of explicit hyperlinks to support non-sequential reading. In the development of such applications the hypermedia tradition has taken an instance-oriented approach to relationships rather than the model-based approach where relationships are identified and determined in the design phase. In hypermedia all documents and parts of documents are uniformly characterized as nodes that can participate in any kind of link. Links and the semantics of the link are specified at run time. The dynamic nature of hypermedia linking is a feature that corresponds well to how relationships need to be supported in digital libraries. A major problem with hypermedia, however, is the use of proprietary protocols and the lack of support for constraints that can be used to ensure consistent and logically correct relationships.

4.5.3 Link Models

The link models of several hypermedia systems are reviewed and compared by E. J. Whitehead in [210] using the Containment Modeling Language, and some of the models are reproduced in Figure 4.2 and Figure 4.3. Containments are relationships used to represent how a container contains a particular entity. The physical inclusion of one item within another is called inclusion and is depicted with a filled arrow. Containments where identifiers are used to reference the members of the contained set are called referential and are depicted with dotted arrows. This way of presenting link models reveals many of the differences that exist. As these figures show, hypermedia solutions are on the one hand highly heterogeneous, but on the other hand there are elements that appear in many of the models, in particular the use of node, anchor and link entities:

- A *node* is the basic information entity of the hypertext. This will typically be a larger information structure such as a document stored in the file system or in a database.

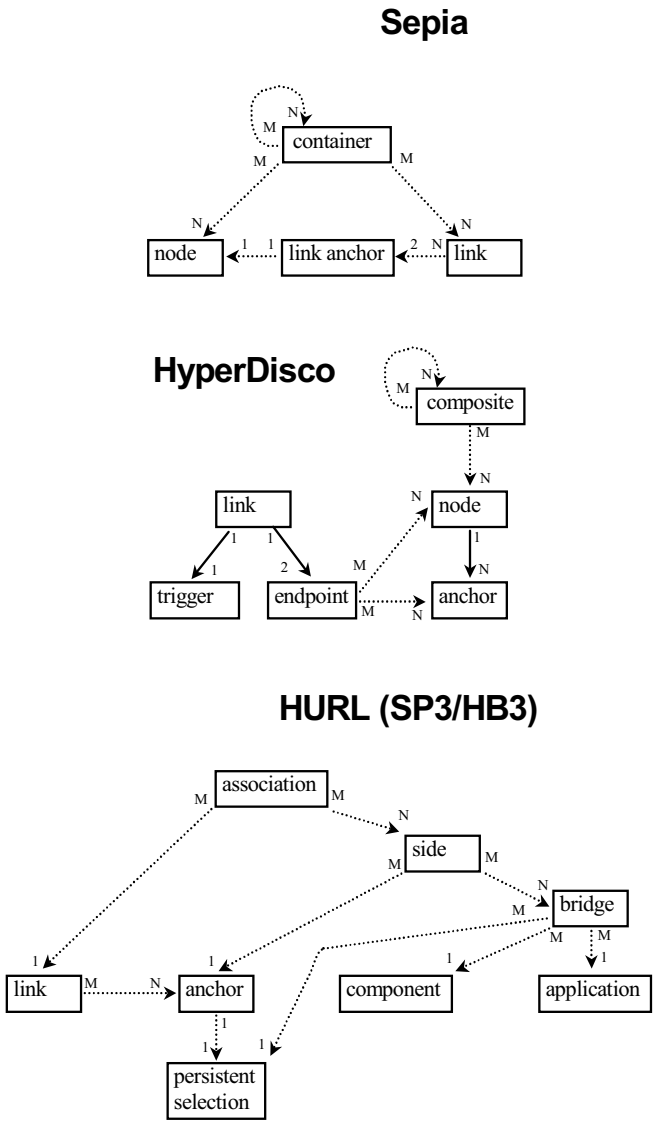


Figure 4.2: Hyperbase systems.

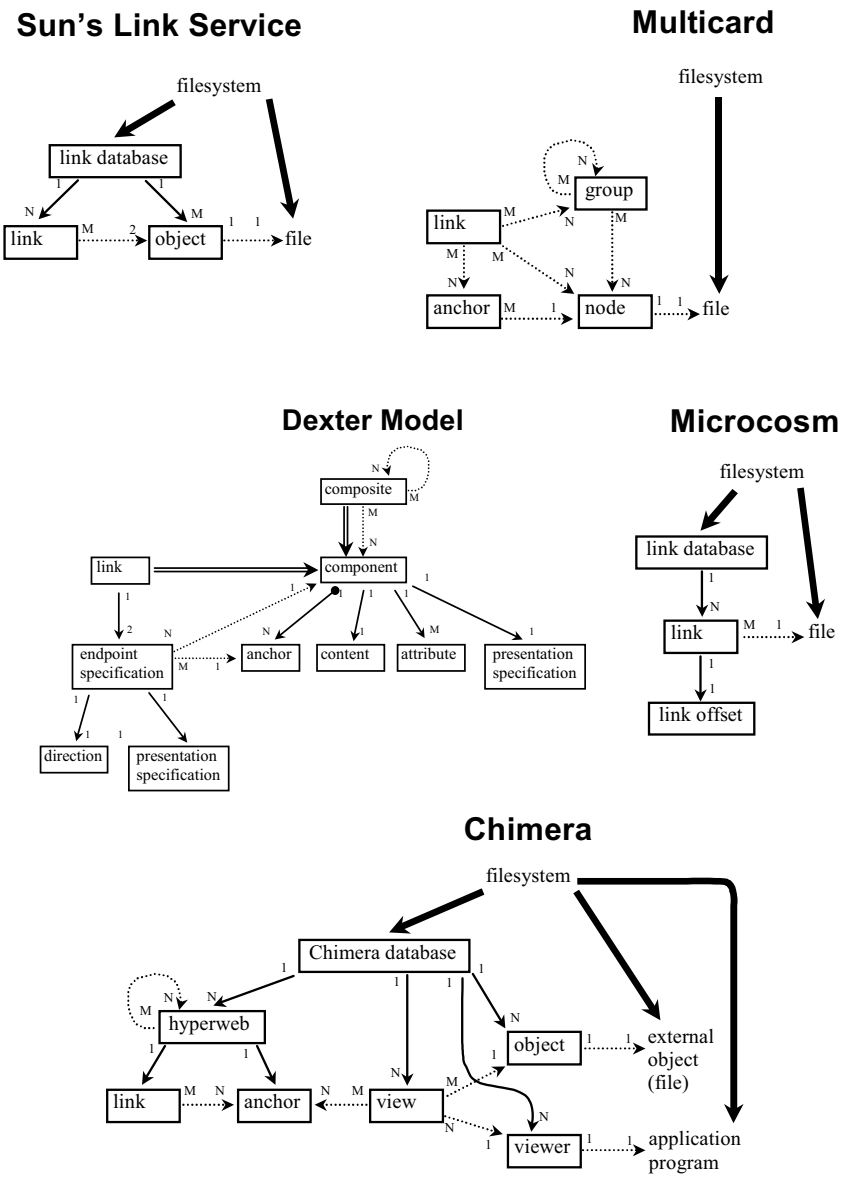


Figure 4.3: Link server systems.

- An *anchor* is a specific selection within the content of the node. The anchor addresses a sub-element of the document such as a particular paragraph or word.
- A *link* is the actual connection between anchors that in most cases are implemented as a tuple of anchor references.

Despite the fact that hypermedia systems at the generic level resemble each other, they are still quite heterogeneous with respect to the details of the link model they implement and the features they support. One aspect is the heterogeneous solutions for mapping between the hypermedia structure and the coarser-grained documents by the use of anchors and endpoints. Another aspect is the integration and communication with applications that in some cases influences the link model. As for many other applications, the use of names and addresses is a fundamental and heterogeneously solved problem in hypermedia [177, 200]. Other elements that influence the link models are the attributes and features that are associated with the links and other elements. The use of link types to express the semantics of the link is described in Chapter 3.6. Additionally explored features include; versioning and time-stamping of links [87, 149], the use of context to determine the behaviour of a link [89, 90], and the use of annotated links to enrich the hypermedia structure [31].

4.5.4 Open Hypermedia

The need for openness, interoperability, and support for distribution of both service and content is widely recognized in hypermedia development and research [71, 139, 173, 174, 202]. The Dexter Hypertext Reference Model [86] was a first attempt to establish a common model for hypermedia systems. The Dexter model distinguishes between the run-time layer, the storage-layer and the within-component layer. The storage-layer models the basic node/link network structure that is the essence of hypertext. A link is a component at the storage layer that aggregates a sequence of two or more endpoint specifications. Unique identifiers are the basic mechanism for addressing any component of the hypertext. Although many systems claim to be Dexter compliant because they implement the same layers, the model never resulted in any significant degree of interoperability.

More recently, there has been other effort in bringing hypermedia over to an open component-based platform that allows for interoperability between hypermedia applications and services [34, 139, 212]. In this context, the linking component is often perceived as a network accessible link service that can be shared by many users and applications. The Open Hypermedia navigational protocol (OHP-Nav) is an attempt to define a standardized solution for link services [50, 138, 169]. The proposed link model for this protocol is illustrated in Figure 4.4, and it defines a link as a collection of endpoints that further can be a collection of data

references. Although the use of this protocol is reported to be successful, e.g. in [208], it may seem that the move towards distributed hypermedia applications based on standardized protocols and models, is a slow process.

The use of the traditional client/server model for link services is the main paradigm explored in distributed hypermedia applications. This, however, is a limited solution. A single link server may only serve a finite number of users and/or be bound to a specific usage, and multiple link servers will naturally occur in a distributed environment. The distribution of filters in Microcosm is one example of a more flexible distributed link service infrastructure, and the transparent access to multiple link servers by query routing mechanism is another example [174]. The more collaborative construction of hyperwebs that is envisioned in e.g. [7], however, remains an issue that requires further research.

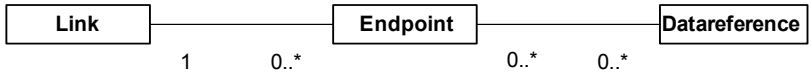


Figure 4.4: The link model of OHP-Nav.

4.6 Descriptive Solutions

4.6.1 Metadata Formats

Typical elements in metadata formats include entries for title, author, date of publication and subject, but additionally metadata can include entries describing the relationships that exist between the described resource and its related resources. One example of a metadata format that supports the description of relationships is the Dublin Core format [60]. Dublin Core defines the *Relation* and *Source* elements which both can be used to describe relationships:

- The *source* element can contain a reference to a resource from which the present resource is derived; which implies a specific kind of relationship.
- The *relation* element can contain a reference to a related resource; which implies a more general purpose element for relationship information.

By using element refinements the semantics of the relation element can be made more explicit than the default interpretation. The DCMI Usage Board maintains a list of element refinements that can be used for refining Dublin Core elements [59], and for the relation element this list includes the following labels:

- Has Format / Is Format Of
- Has Part / Is Part Of
- Has Version / Is Version Of
- References / Is Referenced By
- Replaces / Is Replaced By
- Requires / Is Required By
- Conforms To

The relationship types that Dublin Core supports, however, are not limited to this list but can be extended by defining application- or domain-specific vocabularies that expresses other relationship semantics.

The Dublin Core metadata format serves as a good example of how relationship information can be expressed in metadata. A comparable solution can be found in the IEEE Learning Object Metadata standard [99], whereas other metadata formats, like the various dialects of the MARC format, may have quite different conventions for such information. The way metadata supports relationships will depend on issues like:

- The availability of specific metadata elements for relationship information.
- The ability to specify relationship meaning.
- The ability to address the related resource.
- The level of syntax, such as free text versus highly structured data.

These aspects illustrate some of the characteristics and possible limitations that exist for relationships expressed in metadata. The semantics of a relationship can either be expressed through the use of a general field that contains additional information about the semantics of the relationship (relationship types), or it can be based on the use of different metadata elements where each element expresses a different relationship type. Identification can be made by bibliographic information or by identifiers. The ability to automatically process relationships will depend on the level of syntax. Relationships that merely are expressed as uncontrolled free-text e.g. in note fields will be difficult to process automatically, but the information can still be highly useful for end users when it is displayed.

Relationships expressed as metadata are inherently limited to one-way reference-based relationships if the metadata primarily is intended to describe a single document. If multi-way relationships are required, this must be implemented by adding the inverse reference on the metadata record that describes the opposite resource. Metadata, however, is usually records external to the content they describe, and the inverse relationship can often cost-effectively be determined at runtime if the storage system supports this. A different approach is to implement

metadata that explicitly defines the relationships between documents without specific focus on one or the other of the participants, like the use of metadata to express the structure of digital objects described in [61]. Such solutions are comparable with the explicit link objects of hypermedia.

4.6.2 The Resource Description Framework

The Resource Description Framework (RDF) is a specification developed by the World Wide Web Consortium (W3C) [221, 227]. RDF is initially designed as a foundation for representing and processing metadata about web resources in an interoperable, and it is considered to be the foundation for the Semantic Web. RDF defines a uniform mechanism for describing resources, and makes no assumptions about a particular application domain. It is based on simple data model for representing named properties and property values:

- *Resources.* All things that are described by RDF expressions are called resources. A resource is anything that can be identified by the use of an URI. Examples are a web page, a web site, a person, or an abstract concept.
- *Properties.* A property is a specific aspect or characteristic used to describe a resource. The definition of a property is not a part of the core RDF model but can be defined by the use of the RDF Vocabulary Description Language (RDF Schema), which is a specification for the formal declaration of the resource classes and properties [226].
- *Statements*. A specific resource together with a named property plus the value of that property is an RDF statement. These three individual parts of a statement are called the subject, the predicate, and the object respectively. The object of a statement (the property value) can be another resource identified by a URI or it can be a literal like a simple string or number.

The structure of any expression in RDF can be viewed as a directed graph that consists of nodes and labelled, directed arcs that link pairs of nodes. RDF can be said to express a relationship if the object of a statement is a distinct resource with an identity of its own. On the other hand, if the object is a literal it is more natural to consider the predicate and object as an attribute/value pair. RDF does, however, have some limitations as a relationship description language. The statements of RDF are directed, and a property only captures one side of the relationship semantics – as it is seen from the subject. However, the inverse relationship semantics can be inferred by the use of an ontology.

The main advantage of RDF is the simple but still expressive data model. The data model is independent of any specific serialization syntax, but the main platform for exchanging and storing RDF is XML.

4.6.3 Topic Maps

Another relevant descriptive solution that implements support for relationship knowledge is the International Standard ISO 13250 Topic Maps [112, 165]. Topic Maps is a model and exchange format for semantically rich index-like information. It is based on a data model that consists of three concepts:

- *Topic*. A topic can be anything, such as a web page, a book, a person, an event or any abstract entity. The main purpose of a topic is to reify the concept that the topic is about. It is used to make any kind of abstract or concrete concept into an identifiable entity that can participate in the machine processable knowledge structure of a topic map.
- *Association*. In Topic Maps, an association asserts a relationship between two or more topics. Each topic that participates in an association plays a role in that association called the association role. Associations are inherently multi-directional.
- *Occurrence*. A topic may be linked to one or more information resources that are related to the topic. Such resources are called occurrences of the topic and can include a web page, an image or any other external source of information that is related to the topic in one way or the other. Occurrences are generally external to the topic map document itself, but can also include textual information embedded into the topic map.

In Topic Maps the relationships between things are expressed using associations. Any number of topics can participate in an association, and the participation semantics is defined by the role type. Topic Maps defines a typing mechanism that can be used to specify the type of topics, associations, association roles, and occurrences. This typing mechanism is itself based on the use of topics, which implies that a topic map can be a self-describing multi-level representation of knowledge.

The Topic Map specification was initially based on SGML [105], but is later adapted to the XML platform [198]. A topic map can be implemented as a single file or set of interrelated files that together make up a semantically rich map over any universe of discourse. A simple example that shows a topic map of the persons and relationships of a family is illustrated in Figure 4.5.

4.6.4 HTML

The by far most dominating format for hypertext is the Hypertext Markup Language – the publishing language of the World Wide Web [220]. The basic and most commonly used linking capability of HTML is the use of the anchor element *A* to embedded one-way references in the source document. The *href* attribute of the

```
<topicMap>

<topic id="Ann">
    <instanceOf><topicRef xlink:href="#person"/></instanceOf>
    <baseName><baseNameString>Ann Johnson</baseNameString></baseName>
</topic>
<topic id="Joe">
    <instanceOf><topicRef xlink:href="#person"/></instanceOf>
    <baseName><baseNameString>Joe Johnson</baseNameString> </baseName>
</topic>
<topic id="Pete">
    <instanceOf><topicRef xlink:href="#person"/></instanceOf>
    <baseName><baseNameString>Pete Johnson</baseNameString></baseName>
</topic>
<topic id="Mary">
    <instanceOf><topicRef xlink:href="#person"/></instanceOf>
    <baseName><baseNameString>Mary</baseNameString></baseName>
</topic>

<association id="Johnsons">
    <instanceOf><topicRef xlink:href="#family"/></instanceOf>
    <member>
        <roleSpec><topicRef xlink:href="#mother"/></roleSpec>
        <topicRef xlink:href="#Ann"/>
    </member>
    <member>
        <roleSpec><topicRef xlink:href="#father"/></roleSpec>
        <topicRef xlink:href="#Joe"/>
    </member>
    <member>
        <roleSpec><topicRef xlink:href="#child"/></roleSpec>
        <topicRef xlink:href="#Peter"/>
    </member>
    <member>
        <roleSpec><topicRef xlink:href="#child"/></roleSpec>
        <topicRef xlink:href="#Mary"/>
    </member>
</association>

</topicMap>
```

Figure 4.5: A family expressed as a topic map.

anchor element contains the reference to the target document. The anchor element is intended for links that should be displayed as part of the document content.

A different linking capability is provided by the *LINK* element which can be used to express document relationships. Unlike the *A* element, it may only appear in the *HEAD* section of a document. Links specified by the *LINK* element are not rendered with the document's content, but it can be used to convey relationship information that may be rendered by user agents in other ways.

A more precise meaning of *A* and *LINK* elements can be expressed by using the *rel* and *rev* attributes to specify the roles of the link. For instance, links defined by the *LINK* element may describe the position of a document within a series of documents. In the example of Figure 4.6, links within the head of the document entitled *Chapter 5* point to the previous and next chapters and this way describe the position of this document within a series of documents. The link type of the anchor element specifies that the target document contains a definition of the term that the *A* element encloses. The HTML specification defines a set of recognized link types that includes the *prev* and *next* link types as well as other types that can be used to express different relationships between documents. Additional link types not described in the HTML specification can be defined by authors.

Although the linking elements and the link types of HTML can been used to create a semantic and uniform web structure, the actual structure of interlinked web pages is to a large extent based on ad hoc usage of un-typed anchor elements. The semantics of these links are mainly expressed through the content they enclose and most links are intended for human inspection and not for automatic interpretation. The lack of support for navigation tools that use the *LINK* element can be one reason why few authors use this element to organize information. The additional overhead this introduces in the creation of documents is another. By the introduction of XML [218], the W3C envisions that most markup-based document formats in the future will based on XML and for this reason HTML is reformulated in XML, and all future development of this hypertext format will be based on XHTML [223].

4.6.5 XLink

The Extensible Markup Language (XML) is a generic markup language specified by the World Wide Web Consortium. It is derived from SGML and was originally designed to meet the challenges of large-scale electronic publishing, but XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. W3C has defined a rich set of XML-based formats for a range of applications. A main contribution for hypertext applications is the XML Linking Language (XLink) [224]:

```
<HTML>
    <HEAD>
        <TITLE>Chapter 5</TITLE>
        <LINK rel="prev" href="chapter4.html">
        <LINK rel="next" href="chapter6.html">
    </HEAD>
    <BODY>
        <P>This is a test
        <A rel=”definition” href=”def1.html”>document</A><P>
    </BODY>
</BODY>
```

Figure 4.6: Link elements and link types in HTML documents.

An XLink link is an explicit relationship between resources or portions of resources. It is made explicit by an XLink linking element, which is an XLink-conforming XML element that asserts the existence of a link.

The XLink specification defines six XLink element types; only two of them are considered linking elements. The others provide various pieces of information that describe the characteristics of a link:

- *Simple.* A simple link is comparable to the linking elements of HTML. Simple links offer shorthand syntax for an outbound link with exactly two participating resources, and they have no special internal structure.
- *Extended*. An extended link is a link that associates an arbitrary number of resources. The participating resources may be any combination of remote and local. Their structure can be fairly complex, including elements for pointing to remote resources, elements for containing local resources, elements for specifying arc traversal rules, and elements for specifying human-readable titles.
  The extended-type element may contain a mixture of the following elements in any order, possibly along with other content and markup:
  - *Locator-type* elements that address remote resources participating in the link.
  - *Resource-type elements* that supply local resources that participate in the link.
  - *Arc-type elements* that provide traversal rules.
  - *Title-type elements* that provide human-readable labels for the link.

The XLink architecture does not define specific XML elements for these element types, but rather defines a set of attributes and conventions that can be used to define application-specific xlink elements from the same basis.

Simple xlinks are typically used to embed simple reference links in XML-based content comparable to how HTML links are used in HTML documents. Extended xlinks, on the other hand, are typically stored separately from the resources they associate (for example, in entirely different documents). Thus, extended xlinks are important for situations where the participating resources are read-only, or where it is expensive to modify and update them but inexpensive to modify and update a separate linking element. Additionally, extended xlinks can be used to link between resources with no native support for embedded links.

The semantic attributes *role*, *arcrole*, and *title*, can be used to enhance the various parts of a link with information that enables human and automatic interpretation of the semantics of the link elements and its various participating resources.

The actual relationships between the resources involved in a link are specified using arc-type elements and *label* attributes. Traversal rules can be defined by the *to* and *from* attributes. The behaviour attribute *show* can be used to determine the desired presentation of the ending resource, for example whether the source resource should replace the current content or whether it should be displayed in a new window. Timing of traversal is determined by the value of the *actuate* attribute, for example whether the ending resource should be loaded once the starting resource is found or whether this should be initiated on request, for instance when the user clicks on the link.

A simple example that shows the persons and relationships of a family as an extended xlink is illustrated in Figure 4.7.

4.6.6 HyTime

A more historical attempt to define a standardized hypertext format was the specification of the International Standard ISO 10744 Hypermedia/Time-based structuring language (HyTime) [109]. HyTime provides facilities for representing static and dynamic information that is processed and interchanged by hypertext and multimedia applications. It is defined as an SGML application with different markup-modules like the location address module and the hyperlinks module. HyTime was published in 1997, but the sheer complexity of the specification has not encouraged implementers to support the specification. Many of the modules of HyTime have later been captured in XML-based counterparts like XPath/XPointer [222], XLink [224] and SMIL [219].

```
<!ELEMENT family (person*, relationship*)>
<!ELEMENT person (name)>
<!ELEMENT relationship  EMPTY>
<!ELEMENT name (#PCDATA)>

<!ATTLIST family
       xlink:type     (extended) #FIXED     "extended"
       xlink:title    CDATA      #IMPLIED>

<!ATTLIST person
       xlink:type     (resource) #FIXED     "resource"
       xlink:label    NMTOKEN    #IMPLIED
       xlink:role     CDATA      #IMPLIED>

<!ATTLIST relationship
       xlink:type     (arc)      #FIXED     "arc"
       xlink:arcrole  CDATA      #IMPLIED
       xlink:from     NMTOKEN    #IMPLIED
       xlink:to       NMTOKEN    #IMPLIED>

<family xlink:title="Johnsons">

    <person xlink:label="p" xlink:role="roletypes/mother">
        <name>Ann Johnsen</name>
    </person>

    <person xlink:label="p" xlink:role="roletypes/father">
        <name>Joe Johnsen</name>
    </person>

    <person xlink:label="c" xlink:role="roletypes/child">
        <name>Peter Johnson</name>
    </person>

    <person xlink:label="c" xlink:role="roletypes/child">
        <name>Mary Johnson</name>
    </person>

    <relationship  xlink:from="p" xlink:to="c"
                   xlink:arcrole="arcroles/IsParentOf" />

    <relationship  xlink:from="c" xlink:to="p"
                   xlink:arcrole="arcroles/HasParents" />

</family>
```

Figure 4.7: A family expressed as an extended xlink.

4.7 Overview

The relationship representation technologies reviewed in this chapter covers technologies that are distinctively different, but they all share a common focus on relationships mechanisms and relationship information.

Support for relationships in object-oriented systems has in common the objective to provide implementation support for the relationships captured in the design phase. An additional focus that can be found in many projects is the support for management of relationships at runtime as uniform solutions for adding and deleting relationships, and constraint checking. A typical solution is that the different relationships of the design are fixed to at run-time. The extensions of the various relationship types often exist as different, possibly heterogeneous, implementation units. The implementation of a ternary relationship can be different from the implementation of a binary relationship. The object-oriented paradigm includes the encapsulation of data and functionality which also is applicable to the relationship constructs. Relationship information and relationship behaviour is a coherent solution. The use of object-oriented systems in digital libraries is frequently found, including a few examples on systems that implement explicit relationship objects such as the MARIAN digital library [69, 70] and Intersect_DL [207].

Hypermedia linking solutions – and in particular external link objects – can be considered as comparable constructs to the explicit relationship objects of certain object-oriented systems. They are equal in the sense that the relationship information is externalized from the entities that participate, and that the system provides for a uniform solution to add, delete and use relationships. A major difference, however, is that hypermedia external link objects often are accessible for third-party applications. Another feature is the instance-oriented nature of hypermedia linking. In hypermedia, links do not have to be predefined in terms of what objects they can relate and what meaning they convey. A uniform construct is used to instantiate any kind of link between any documents. This implies that the extension of hypermedia links is a structurally uniform set with individual loosely, typed semantics. Hypermedia link services, however, lack the focus on database-like features that can be found in object-oriented systems, and do not provide for features that can be used to ensure the integrity and consistency of the link-base, such as constraints. The use of hypermedia link services in digital libraries is explored in e.g. [4, 42, 211, 213].

One exception that exists in-between the object-oriented solutions and the hypermedia solutions is the CORBA relationship service which can be used to create relationships between third-party distributed objects. The use of distribute objects to implement relationship instances enables the relationships to be available as sharable resources for other applications. The ability to create a relationship is

detached from the entities that participate in the relationship and the applications that manage entities. The CORBA relationship service additionally implements an object model that is somewhat equal to the link models of certain hypermedia link services. A major problem with the CORBA Relationship Service, however, is that it only supports predefined relationship types. Each application needs to define and implement the relationship types it requires. Another exception that exists in-between the object-oriented solutions and the hypermedia solutions is the SADES system that supports dynamic relationships between the classes and instances of an object-oriented database [167].

The descriptive solutions examined are quite diverse in their purpose. They represent formats for interchangeable information but target different applications. Relationships can be expressed as metadata; exemplified by Dublin Core elements like relation and source. Metadata can be interpreted as relationship statements, as is the case for RDF; and even as links, as explored in [142]. RDF, Topic Maps and XLink are all based on graph-like models and are all intended for describing relationships between entities with identity [141]. The focus of XLink is mainly to support hypertext-like applications, whereas RDF and Topic Maps are formats in the metadata tradition and support semantically rich descriptions of information resources including relationships.

All reviewed solutions can be used to express relationships in digital libraries, in one way or the other. However, certain formats like specific metadata formats and RDF merely support relationships with one-directional semantics, whereas extended XLinks and Topic Maps support multi-directional relationship semantics.

Dublin Core, RDF, Topic Maps, and XLink can provide a flexible instance-oriented typing of relationships, while other features are more related to the specific application domain the various formats are targeting. All formats are envisioned to exist in a distributed environment where applications may access different records from different repositories. A particular problem is the lack of support for consistency and integrity solutions for such information in a distributed context.

The format that most directly focuses on explicit relationships is XLink. Its main influence is standards like HTML, HyTime, and the Text Encoding Initiative Guidelines, but it is also informed by various hypermedia systems. The extended links of XLink provides a well-defined and comprehensive support for expressing relationships, but the target application is text based (XML) encoding of relationship information with a certain focus on hypertext applications. The use of metadata formats is a fundamental element in many digital library systems, and RDF is gaining acceptance as a platform for syntactic and semantic metadata interoperability for all descriptive information, including relationship information. However, when the application focus is on the support for explicit relationships and linking information, it seems like XLink is favoured. Examples and suggestion for the use of XLink for this purpose can for instance be found in [22, 23, 40, 134, 135, 147].

5 A Generic Relationship Model

5.1 A Model for Explicit Relationships

This chapter presents a conceptual reference model for the explicit representation of relationship information. The model is used to formally express the various aspects of explicit relationship information that this work emphasizes. The definition of this model is the first step to establish a generic solution for managing and navigating relationships in digital libraries. It is influenced by the various solutions and models reviewed in Chapter 4, but attempts to generalize the those features that this work considers important.

The model is *generic* in the sense that it defines an abstract model that can be used to capture all kinds of relationships regardless of their meaning and structure. The model reflects the information related to the relationship in focus and represents this information in an explicit form. The model is *abstract i*n the sense that it does not specify any preferred data structure for its implementation.

5.2 Requirements

The definition of the model is based on a set of requirements for what a generic construct for relationship information needs to support. These requirements will be explained in detail in the following sections, but can be summarized with the following list:

- *Participants.* Explicit relationships express information about the relationships that exist between participants, and needs to express the list of participants it relates.
- *Relationship semantics.* Relationships are highly semantic elements that express knowledge about the structure of things. A generic relationship

model needs to be able to express the semantics of different relationships using relationship types.

- *Relationship structure.* Relationships can have different structures, both in terms of degree and cardinality. A generic relationship model must be able to represent any kind of structure within the same extension.
- *Constraints.* The ability to structure knowledge by the use of relationships needs to be complemented with the ability to constrain the relationship instances. Constraints are the rules defined for the relationship type that can be used to ensure logically correct relationships.
- *Identity of a relationship.* Management of relationships requires the relationship to be addressed as a unit. To support this there is a need to provide relationships with identity, either through one identifier for the relationship as a whole or by providing identifiers for the various part of a relationship.

5.2.1 Participants

Any extensional relationship construct needs to store information about the entities the relationship relates. The relationships in class diagrams specifies relationships between classes, but at the instance level each relationship needs to be implemented as a connection. Entities can be represented in relationship instances by any kind of name, identifier or other reference that uniquely identifies the related entities.

Relationship can additionally be generic and relate logical units rather than specific entities, such as the use of generic links in Microcosm where the endpoints can be specific words independent of what documents they occur in [87].

Certain relationship representations allow for the entity to be included within the relationship instance. In XLink an extended xlink can include a resource element that exists within the xlink.

The model presented in this chapter does not presuppose any specific addressing scheme or constrain the interpretation of what a participant is. Participants are simply interpreted as anything that can be referenced.

5.2.2 Relationship Semantics

The semantics of a relationship is the meaning of that relationship. Without meaning, a relationship is merely a connection that in many cases will be of no value, or would require the context to be examined for semantic indications.

In an instance-oriented relationship construct, the relationship itself should be able to express the meaning of the relationship it implements. Support for semantically rich information is a complex issue, but a common initial requirement is to support the naming of things by symbols such as human readable terms. Relationships are

often said to be of a specific type. By supporting the identification of relationship types by a typename, further support for semantics can be handled at the application level: if a specific relationship type is recognized it can be interpreted correctly; if the relationship type is unrecognized, the application at least knows that it is faced with an unknown relationship and may act accordingly. Even though the relationship is unrecognized, the application may still be able to process the relationship in certain ways, if the overall structure of all relationships is uniform.

The symbol required to identify relationship semantics can be any symbol implemented as a unique simple or complex data value. For the relationship of *Fatherhood*, the string "Fatherhood" may be used to identify this particular kind of relationship. Other identifiers are of course also possible. Readable strings may be intuitive and easy to interpret by humans, but other tokens like numbers or strings can be selected for computational reasons. If the requirement is efficient comparison of two symbols an integer number would be better than an extensive string. An additional requirement is that names have to be unique within the context of the application. A local system may solve this in a pragmatic way, whereas a global cooperative environment would require globally unique identifier like URIs or UUIDs.

The use of roles has emerged as an important element when expressing and modelling relationships. This can be found in current modeling languages such as UML, it is used in many object-oriented relationship mechanisms, and occurs in many of the descriptive solutions such as Topic Maps and XLink.

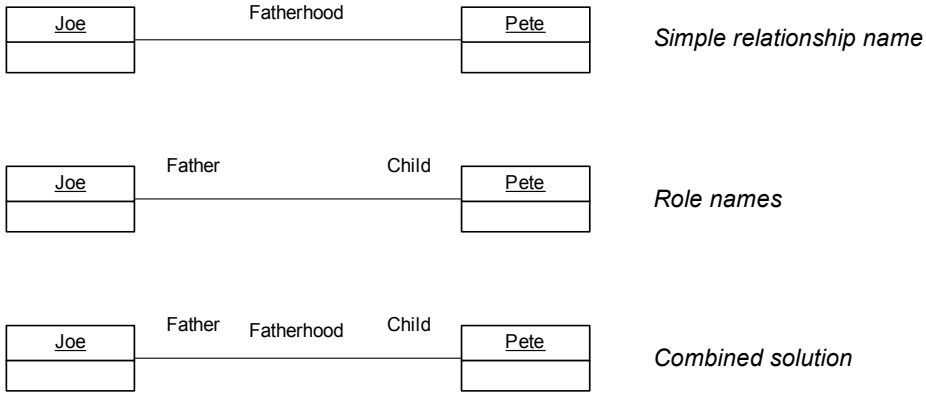


Figure 5.1: Naming relationships.

Roles provide for participation semantics. The use of role names is an efficient guidance when interpreting relationships; e.g. to determine which endpoint is the *Father* and which endpoint is the *Child* in a *Fatherhood* relationship instance. The use of roles in a relationship specification diminishes the need to have knowledge of the entities to understand the relationship. For this reason, the use of role names is an important element. Role names may replace the general relationship name, or both may be used as shown in the object diagram of Figure 5.1. There may even be cases where the full semantics of a relationship (both relationship type and role types) is required to interpret the meaning of the relationship.

5.2.3 Relationship Structure

A generic relationship must be able to support the different structural shapes a relationship may have, and the major variable aspect of relationship structures are *degree* and *cardinality*. In the relationship mechanisms of object-oriented systems, different degrees and cardinality constraints often lead to differences in how the relationship is implemented. The generic constructs of e.g. Topic Maps and XLink, supports the representation of arbitrary relationship structures with the same construct. .

Some relationships are of binary degree; they relate two entities, other relationships are of higher degree; they relate three, four or even more entities as shown in Figure 5.2

A different aspect of relationship structure is cardinality. Cardinality is by definition the size of a set, which in the context of relationship modeling is the set of relationship instances an object participates in. Relationship participation is in general a dynamic aspect of a system, and at the modeling stage cardinality is usually defined as a cardinality ratio constraint. A specific entity may be related to several other entities by different relationship instances of the same type. The entities in Figure 5.3 are only allowed to participate in one relationship instance; which is resulting in a single link between the objects at run time. In Figure 5.4 the model expresses that E1 is allowed to participate in multiple relationship; which is resulting in multiple links at run time. These two examples can be implemented by different solutions. In an object-oriented mechanism, the relationships may be implemented using pointers referencing to the related object(s). If the cardinality ratio is 1, it will be sufficient to use a simple attribute to store a single reference to the related object. If the cardinality ratio allows for multiple participation, it is required that the attribute implements support for a set of references. The set based solution is more general since it also can be used to implement the singleton.

The above example may seem trivial, but the problem of managing multiplicity in a uniform way is a pertinent problem as exemplified by the interoperability problems

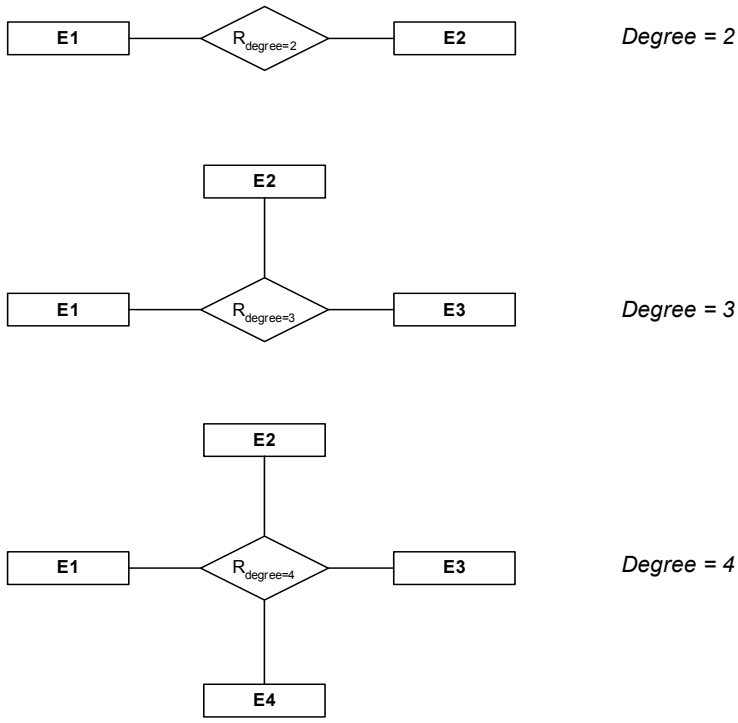


Figure 5.2: Relationships of different degree.

between the different link networks of hypermedia systems [127]. Different solutions that are used include:

- The use of multiple links.
- The use of multi-destination links.
- The use of single link that points to an intermediate node having multiple links to the target.

RDF, Topic Maps, and the ISO HyTime specification are all examples of solutions that implement a data structure where a single instance of respectively association and link may have an endpoint that actually is a set of participants. In HTML the same information needs to be represented as multiple links. In XLink, arcs can be defined between locators in a generic way by indicating to and from. These differences can be quite equal in expressiveness and appearance to end users, but will differ in terms of how the relationship information must be processed.

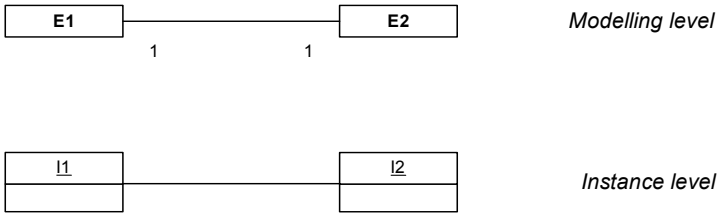


Figure 5.3: Binary 1:1 relationship.

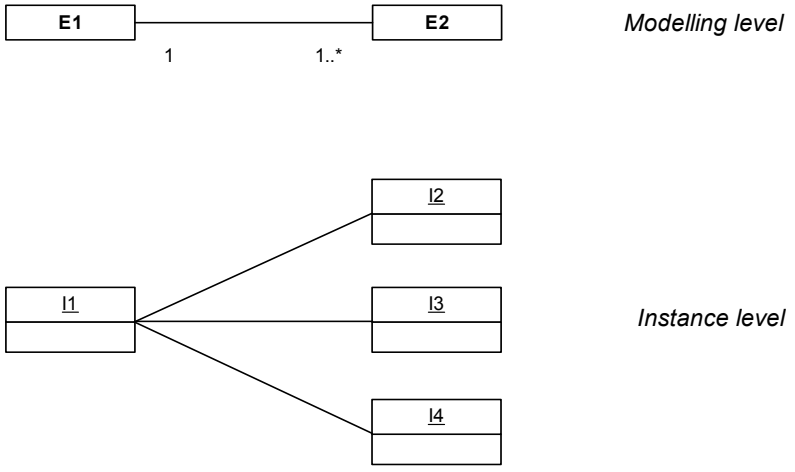


Figure 5.4: Binary 1:N relationship.

Representing multiplicity by the use of multiple relationship links appears to be the most generic solution, and it can for this reason be preferred as the basis of a generic relationship model. In case there is a need to establish a single relationship instance from/to a collection of participants, this can always be solved by creating a collection and otherwise treat this collection equal to other singleton entities.

5.2.4 Constraints

The semantics and the structure of relationship instances are often bound by rules that define what is logically correct or not. When modeling information, constraints

are used to define additional aspects the relationship structure needs to conform to at run-time.

If the structure is generic, but the instances are specific in terms of structural constraints like degree and cardinality ratio, we have to be able to define constraints on each relationship type in order to prevent instances of this particular type from being structured in an erroneous or unwanted way.

In the design of a generic relationship structure, the ability to support constraint definitions is an important issue. This topic has particularly been in focus in the development of relationship mechanisms in object-oriented systems [98, 132, 175, 179], but is rarely found in link models of hypermedia and descriptive solutions.

Although constraints can be defined for any dynamic aspect of a system, the most commonly used constraints when modeling relationships are cardinality and total/partial participation; either as two separate constraint elements or combined with the use of cardinality ratios e.g. where the first number indicates the required number of participation (0 or 1) and the latter indicates the maximum (1, 2 ...,N).

In the development of a generic relationship model, however, there are additional constraints that need to be considered. The first is the ability to express degree as a constraint. If the generic relationship structure allows a relationship to be instantiated with any degree, a degree constraint can be used to specify the correct degree for a type. The second is the ability to specify constraints on the semantic aspects of a relationship type. If a relationship is expressed in terms of the roles of the participating entities, we have to be able to specify e.g. that a role only is allowed to occur in combination with the use of other specific roles on the opposite end of a relationship.

5.2.5 Relationship Identity

The identity of a relationship may be based on either:

- Identity by value
- Identity by name

The first approach assumes that the relationship identity can be established by looking at certain values; like the set of participating entities, relationship type, etc. Identity by name, on the other hand, implements identity by assigning unique identifiers to relationships. The first solution may be difficult to implement and process at runtime, but requires no specific assignment at creation time. The latter approach is easier in e.g. comparison operations, but introduces a certain overhead by the required assignment of relationship identifiers. This overhead may differ in different implementation and runtime environments, e.g. in a distribute environment this would require the use of globally unique identifiers like UUIDs. The use of identifiers and implementation of an identification scheme, however, is more an

implementation issue. At the conceptual level it is sufficient to state that relationship identity is required for the relationship as a whole and/or on subparts of the relationship.

5.3 Developing a Model

5.3.1 An Initial Model

Relationships are often formally explained with the mathematical relation. A generic relationship expression $\Re_{any}$ can be defined as a relation of *any* degree over the domain of discourse *P*, where *P* is the universal set of all possible relationship participants in a domain; like all objects in an object-oriented application, all documents of a digital library, all persons, or any other set of entities. This can be expressed by the generic statement:

$$\Re_{any} \subseteq P \times P \times P \ldots \times P$$

This approach, however, is not transformable into a single, common relational representation for the extension of all possible relationships. Relationships that differ in terms of different degree will inevitably lead to different relations. A concrete mathematical relation is per se of a fixed degree; it is a relation over a fixed number of sets. A generic relationship construct must, however, be capable of representing any kind of relationship within one structural extension. A generic statement can, on the other hand, as a fist step be represented as a set of sets, where each instance of a relationship – the relationship link – is viewed as a set, and the overall relationship extension is viewed as a set of such sets:

$$S_{sets} = \{\{p_1, p_2\}, \{p_1, p_3\}, \{p_1, p_2, p_3\}, \{p_4, p_5, p_6\}\}$$
$$p_n \in P$$

This straight-forward representation is however a simplification of relationships, as it has weak support for relationship semantics. A type can be assigned to each relationship instance, but a major drawback is the problem of semantic ambiguity; how to determine what list position corresponds to what participation role. Using ordered sets does not contribute any meaningful mechanism because each relationship instance may have a different number of elements. The model needs for that reason to be further enhanced by:

- Adding typed participants
- Adding a generic type system

- Defining support for constraint

5.3.2 Embedding Role Semantics

The semantic ambiguity problem can be solved by modeling a relationship as an aggregation of compound role expressions, and adding support for semantic labels – names – to the model. In the following section we will be using a specific terminology when referring to the model. The term relationship will be used when referring to the overall relationship structure, and the terms *link* and *roles* will be used when referring to the specific parts of the model that are named in this way.

A participant together with the participation rolename can be viewed as the role of a relationship. A role structure can be modelled as an ordered pair having the participant as one coordinate and the rolename as the second coordinate:

$$Role = \{(p, n_{role}) \mid p \in P_{participants} \wedge n_{role} \in N_{rolenames}\}$$

The use of such a role construct can be found in certain object-oriented relationship mechanisms like the CORBA Relationship Service [77]. Similar solutions are found in both Topic Maps [112] and XLink as well [224]. Based on this solution, a generic relationship can now be defined as a relation of relationship instances – links – where each link as the first coordinate has a set of *roles* drawn from the set of all possible roles. The name of the relationship instance is introduced as the second coordinate:

$$\Re_{link} \subseteq \{(r, n_{link}) \mid r \subseteq Role \wedge n_{link} \in N_{linknames}\}$$

Some examples based on this model are given below (parentheses are used to delimit ordered tuples and curly braces are used to delimit sets):

$$(\{(Peter, Father), (Paul, Child)\}, Fatherhood) \in \Re_{link}$$

$$(\{(Mary, Wife), (Bill, Husband)\}, Spouse) \in \Re_{link}$$

$$(\{(John, Friend), (George, Friend)\}, Friendship) \in \Re_{link}$$

Additionally, support for relationship attributes and even role attributes can be added without altering the basic principles of the model. Attributes that are common for the whole relationship construct can be added as additional coordinates to $\Re_{link}$, and attributes on the roles can be added to $Role$, following the same pattern as attributes on the link. Finally, both role and link tuples can be identified using the values they contain or additional coordinates can be introduced if there is need to add specific identifiers.

5.3.3 Adding Types and Constraints

The relationships we have considered so far have all been unconstrained. In the examples above, relationships that make sense have been shaped, but there are no rules that govern the constellation of roles and links, the number of roles allowed in a link, etc. To support this, there is a need to add a typing scheme that can be used to specify which relationship instances are valid.

The set of all role and link typenames that are valid for a given domain can be partitioned into respectively a set of role typenames and a set of link typenames as in the following example:

$$N_{role} = Rolenames = \{Child, Father, Wife, Husband, Friend, \ldots\}$$
$$N_{link} = Linknames = \{Fatherhood, Spouse, Friendship, \ldots\}$$

Each rolename can be associated with a minimum and maximum value to express the cardinality constraints. Together this information makes up a role type definition:

$$\Re_{roletypes} \subseteq \{(t_{role}, c_{min}, c_{max}) \mid t_{role} \in N_{role} \wedge c_{min} \in Z \wedge c_{max} \in Z\}$$

A link type can be defined as a tuple that contains the a set of role types as the first coordinate, the link type as the second coordinate and the degree as the third coordinate:

$$\Re_{linktypes} \subseteq \{(r, t_{link}, d) \mid r \subseteq \Re_{roletypes} \wedge t_{link} \in N_{link} \wedge d \in Z\}$$

The use of degree in the definition is essentially not necessary, because the degree can be inferred from the number of role types that are allowed for a link type. The use of a specific degree constraint, however, is a more flexible solution because for instance if the link only accepts one role type but allows two participants of this role type.

The set of all link and role types that are allowed within a domain can be characterized as a relationship typology:

$$Typology = \{\Re_{roletypes}, \Re_{linktypes}\}$$

Together this typology defines the basic typing scheme for relationship instances:

- Role types are defined by:
  - A role typename
  - A value for maximum cardinality
  - A value for minimum cardinality
- Link types are defined by
  - A link typename
  - The degree of the relationship
  - A list of allowed role types

5.3.4 Overview

The model presented in this chapter includes a formal representation of relationship instances and the typing scheme that can be used to specify what relationships are logically correct. The basic relationship model is comparable to the flexible relationship scheme that e.g. is implemented in both Topic Maps and XLink, and reflects an instance-oriented view on relationships. The use of roles to represent participants combined with a linking element that aggregates roles, is additionally found in certain object oriented relationship constructs, such as the CORBA Relationship Service.

The use of the typology to control the creation of relationship instances is a deployment issue. As long as the names used in the relationship instances correspond to the names used in a type definition the application should be able to control that the relationship instances adheres to the type definition. An implementation of this model may further consider adding additional constraints to the typology schema. The schema is generic in the sense that it defines the core structure of a relationship typology. It allows for a range of different solutions to be implemented depending on what the requirements are for typing and constraints.

6 Design and Architecture

6.1 Supporting Relationships in Digital Libraries

A major element of the work presented in this thesis is the design and development of a system for supporting relationships in digital libraries. This system is coined the Digital Library Link Service (DL-LinkService), and as the name of the system indicates, the target application domain is digital libraries. However, digital libraries are a broad category of systems that deal with information in a variety of ways. Rather than focusing on a specific application of relationships applied to a specific class of information, the system addresses relationship support in a generic and flexible way by providing a core solution for creating and maintaining a consistent network of relationship instances.

In digital libraries, the extension of the information object concept is at the general level highly homogenous – all objects are identifiable units of information. On the lower representation level this extension can be heterogeneous, but this is often transparent to the way information is managed and organized. The relationships an information object potentially can participate in are not a predefined and static aspect of the object, but it is rather a dynamic aspect that evolves as information is used and the meta-level knowledge that makes up an information space evolves. This leads to a requirement for a dynamic solution where arbitrary relationship instances can be added at run-time between any set of objects.

The system that further will be explored in this thesis combines features from different disciplines into a dynamic and flexible relationship service for digital libraries. The system implements an object-oriented interpretation of the abstract relationship model specified in Chapter 5. The system is partly influenced by the CORBA Relationship Service, whereas other aspects, such as the dynamic nature of the system, are inspired by hypermedia systems. The design of the system will be described in detail in the following section, but the main characteristics of the system can be summarized by the following:

- *Object-oriented representation of relationships.* The DL-LinkService implements relationships by the use of a compound structure of *node*, *role,* and *link* objects. An object-oriented solution allows for encapsulation; the grouping of data and the operations that affect the data into objects. One example of the advantage of this is the implementation of mechanisms for constraint checking. In an object-oriented solution, the constraint mechanisms can be delegated to the object instances; a solution that facilitates transparent constraint checking on a fine-grained per object level.
- *Relationships are explicit and detached from the information object level.* In a digital library environment, as well as many other applications, the potential relationship participants cannot be assumed to have inherent properties for expressing relationships to other entities. To support the creation of explicit relationships for all kinds of information objects stored in any kind of system, it is necessary to separate the relationship layer and the information object layer. Another motivation for this separation is that it allows relationships to be defined between third-party objects.
- *Support for relationships of any structure.* The DL-LinkService supports relationships of any degree to be defined, and supports the definition and management of any cardinality constraint.
- *Typing.* All relationships created by the DL-LinkService are instantiated using the same core set of object classes (interface definitions). Relationships can, however, be of different types. The type of a relationship is specified by the client when relationship instances are created; by the use of typenames. The motivation for this solution is to enable a reusable system that is independent of compile time knowledge of all relationships types.
- *Support for relationship typologies.* A typology is used to specify typenames and structural constraints for specific categories of relationship instances. This is a flexible system for typing. The same software can be used with different typologies in different domains or for different purposes, and the typology can dynamically evolve by adding new types if needed.
- *Support for distribution.* The DL-LinkService is implemented as CORBA distributed objects and uses Uniform Resource Identifiers to address and identify the information objects participating in a relationship. This solution allows for distribution along many different axes: The interrelated information objects can reside on different systems independent of the location of the service. The relationships and other components of the service can be served as a standalone system on one machine or as a set of transparently cooperating systems distributed across any number of machines.

In the following sections the architecture and design of the DL-LinkService is outlined. Additional application issues are discussed in Chapter 7, and implementation level issues are described in Chapter 8. Although this chapter is devoted to the general patterns and architectural issues of the service, certain implementation level solutions will be touched upon to describe the general ideas of the design that are influenced by the implementation.

6.2 The Core Tasks

The DL-LinkService addresses the general and core tasks needed to manage and use relationships. These tasks are illustrated in the UML use case diagram of Figure 6.1 and include:

- *The management of relationships;* which is defined as the tasks of creating and deleting relationships. Both these tasks inherently depend on the system being able to ensure the consistency of the overall relationship structure.
- *The navigating of relationships;* which is the iterative task of traversing the relationship network. To be able to support navigation, the system must support the discovery of what relationships exist for a specific information object. In addition, the system must support basic methods that can be used to retrieve the opposite entities of the relationship.

The tasks defined in the diagram should be considered a basic set of features. Relationship management and the use of relationships in a specific application can be achieved by relying on the basic tasks defined. Other issues would require extensions to the system, like support for relationship indexing and querying, but this can be achieved without altering the core model.

6.3 The Object Model

The DL-LinkService implements relationship instances by the use of a compound structure of *node*, *role,* and *link* objects. A UML class diagram illustrating the object model is presented in Figure 6.2. Only a subset of the available operations defined in the prototype version of the DL-LinkService is shown for the sake of simplicity. In the following we will be using the term *relationship* when referring to the composite structure of objects that makes up an explicit relationship instance. The various objects will be referred to as *node object*, *role object*, or *link object*. The following list describes the main responsibilities for these objects. Dynamic aspects, for example how relationships are created and deleted and how constraints are handled, is described later in this chapter.

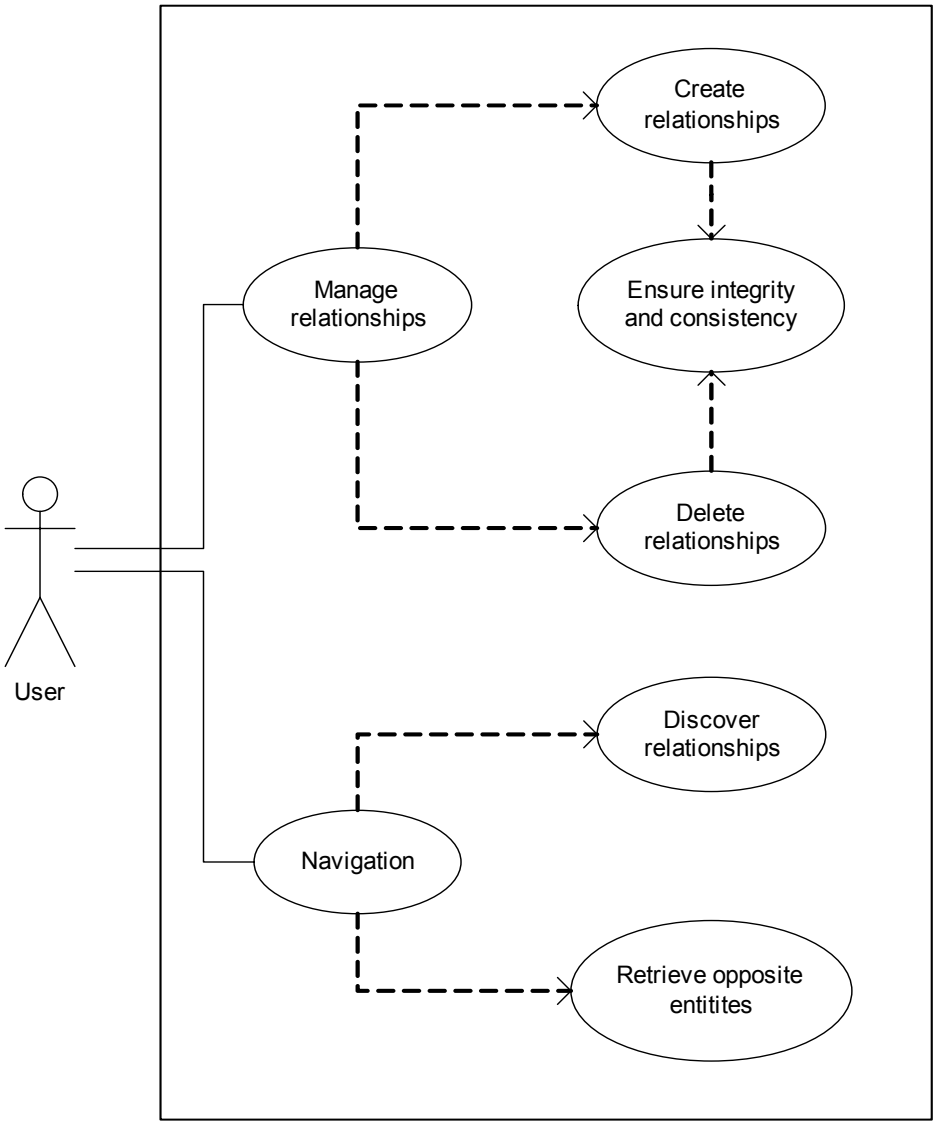


Figure 6.1: Use case diagram of the DL-LinkService.

- The *node object* is an intermediary for an information object. A node object should be considered as a proxy for the actual information object because it implements the functionality that is required for dynamic relationship participation. A node can be of a specific type in order to express the classification of the object it represents. The mapping between the node and the information object is implemented as an attribute on the node object that holds the URI that identifies the actual information object. For each of the information objects there should be only one node. The node is further responsible for maintaining a list of references to its associated role objects, one unique role for each type of relationship it participates in. By aggregating a set of roles for the same information object, the node serves as the connecting element between relationships of different types. This is used to create a network structure of entities and relationships – this will be referred to as *the relationship network*. Additional responsibilities of the node are to support navigation by returning the list of all roles when requested or a single role of a requested type. The title attribute can contain any user assigned value e.g. for display purposes.
- A *role object* reflects the participation of the node in one or more relationship instances of the same type. The semantics of this participation is determined by the value of the type attribute that uniquely identifies a specific kind of role. If a node participates in a relationship with the role of *references,* then there has to be one (and only one) role object connected to the node for this kind of relationship. If the same node also participates in a semantically different relationship there will be an additional role object to represent that participation semantics, for instance a *defines* role. The role object is responsible for maintaining a reference to the node as well as maintaining the list of references to link objects. When new links are added or removed, the role is responsible for checking the participation ratio constraints that are specified as minimum and maximum cardinality. Furthermore, the role is responsible for returning node-references or the list of link references upon request.
- A *link object* is the main connection point between the nodes participating in a relationship instance. For each relationship instance between two participants (or more for higher degree relationships), there will be only one link object. If the relationship is binary, then the link object holds two role object references. For relationships of higher degree, the number of role object references held by the link object is three or more, corresponding to the degree of the relationship. For a node that participates in multiple relationships (cardinality > 1), there will be a corresponding number of link objects. The link object is responsible for returning the set of role references it holds upon request.

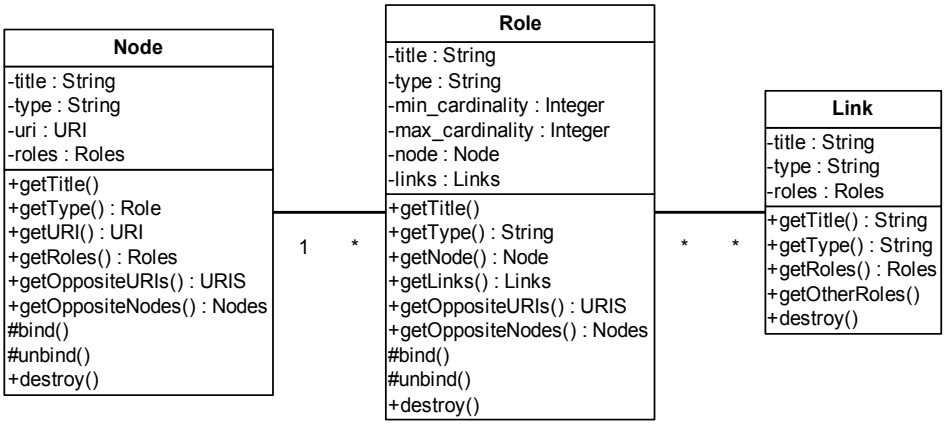


Figure 6.2: The objects used to implement a relationship.

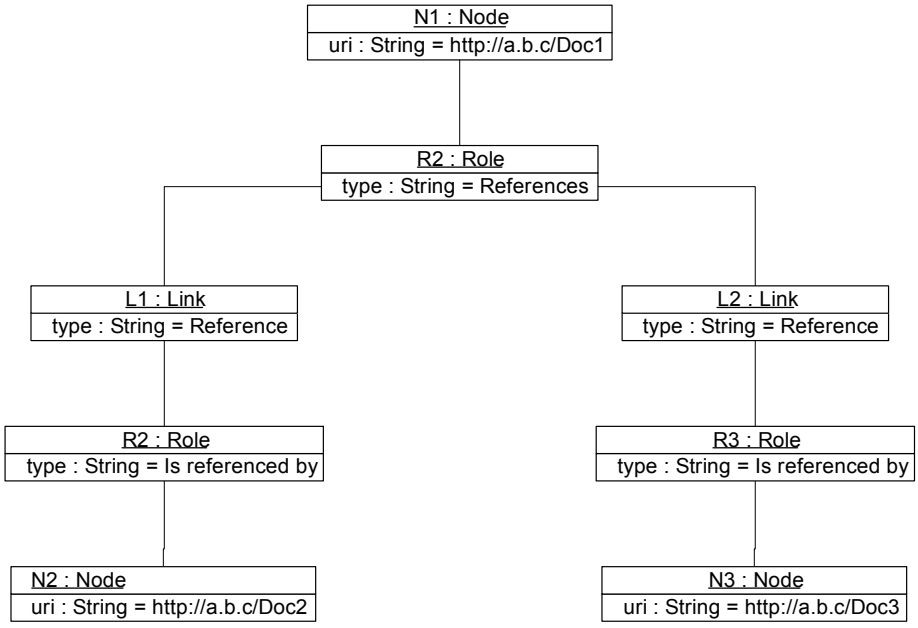


Figure 6.3: The use of node, role and link objects.

An example of the occurring object structure is shown in the UML instance diagram of Figure 6.3. This example shows how a *reference* relationship between the documents Doc1, Doc2, and Doc3 is instantiated as a set of objects at runtime. The actual documents are identified by URIs and represented through the use of the node objects N1, N2 and N3. This instance diagram reflects that Doc1 *references* Doc2 and Doc3. If we view this relationship from the opposite viewpoint it shows that Doc2 and Doc3 *is referenced by* Doc1.

6.4 Navigating Relationships

Navigation is the process of retrieving the opposite target nodes (and the URIs they contain) given a specific start node. The node, role, and link objects are connected to each other by the use of two-way reference-based connections, and the basic get methods on objects are the main enablers for navigation across a relationship.

Direct invocation of these methods by the client application, however, is a complex solution for navigation and a pattern that has to be repeated for each relationship that is navigated. A more intuitive design of the navigation task is to support navigation as a single method invocation on the start node – getOppositeURIs(). When this method is invoked it causes a chained method invocation to occur behind the scene as illustrated in Figure 6.4. Navigation of a relationship, as a task performed by a client, can then be executed in two ways:

- The client can use the basic methods on node, role, and link objects to navigate the compound structure of the relationship, by a series of stepwise and sequential method invocations.
- The client can invoke a single operation on a node object to retrieve the opposite node objects or opposite URIs of that relationship.

6.5 Creating Relationships

Because of the distribution of objects across various address spaces, CORBA does not inherently support creation of objects in contrast to object-oriented languages where this is natively supported by the class construct. In CORBA the mechanism for instantiating objects has to be specifically implemented into the distributed object application. The instantiation of objects in the DL-LinkService follows the frequently used pattern of factory objects [66, 93]; specific objects that implement create methods and return object instances. One factory object is specified for each of the respective node, role, and link objects as illustrated in Figure 6.5. The process of creating a relationship requires the following steps:

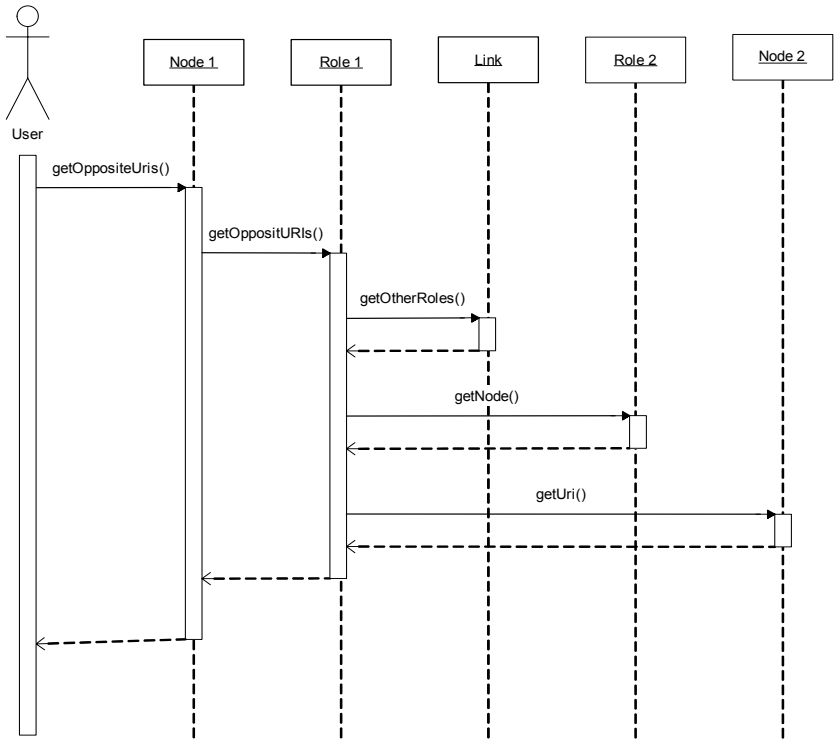


Figure 6.4: Navigation as a chained method invocation.

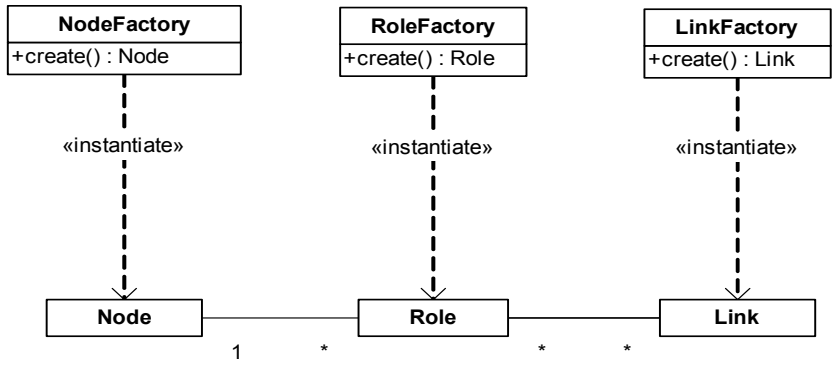


Figure 6.5: Factory objects.

- *Creating nodes.* Nodes are created by invoking the create method on a node factory, passing a typename, title and the URI of the information object as parameters. The create method returns a reference to the created node. If a node for this information object already exists, this step can of course be skipped.
- *Creating roles.* Roles are created by invoking the create method on the role factory, passing the role type, title and a node reference as parameters. The create method returns a reference to the created role. If the node already is associated with a role of the requested type, the role factory forwards the exception that the node throws.
- *Create a link.* A link is created by invoking the create method on the link factory, passing the link type, title and a set of participant roles as parameters. The create method returns a reference to the created link. If the requested link creation will cause link duplication or other errors, the link is not created and an exception is thrown.

Because of the compound structure of objects that are used to represent a relationship, the DL-LinkService implements a "behind the scene" mechanism to establish a synchronized multi-way relationship. The basic pattern for this is illustrated in Figure 6.6. A node is notified whenever a role object is created so that the node can update its list of connected roles. This is implemented by the use of a protected bind method on the node. When the create method is invoked on the role factory, the factory is responsible for invoking the bind operation on the node reference that it receives. The bind method on the node is invoked with the role reference as a parameter. A similar mechanism is designed to establish the two way connection between link and roles. Additional unbind operations are specified for the reverse process of deleting a relationship.

6.6 The Type Service

The DL-LinkService is based on the assertion that the semantic and per instance structural aspects of relationships should be separated from the design and implementation of the server. The system implements a basic structure that initially can be used to create any relationship instance, logically correct or not. As described in 5.2.4, a typing scheme is required to enforce a formally correct relationship structure to emerge according to the rules that may exist.

All objects that the DL-LinkService uses to implement a relationship instance are associated with a specific type. When objects are created, the client specifies the requested type of object in the parameters of the create method. Each created object

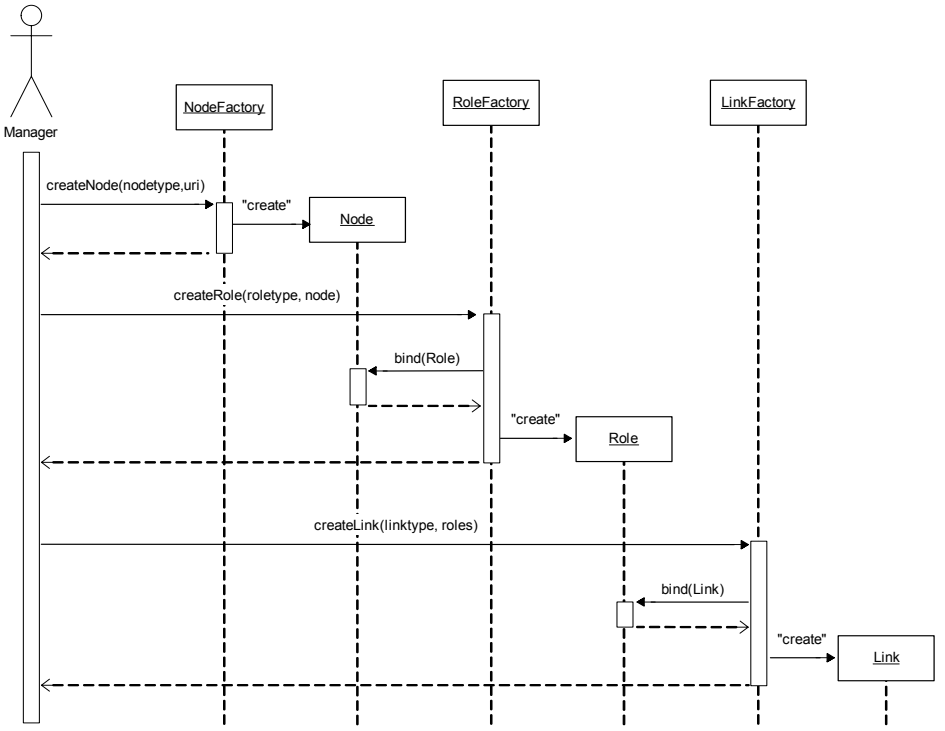


Figure 6.6: The bind pattern.

stores its typename as a string, and clients can later request the typename to determine the semantics of the various objects. Link typenames reflect the general relationship type and role typenames reflect the participation semantics. Additionally, the use of node typenames is introduced to enable a more expressive system than the mere relationship instance is able to express. The proper interpretation of an information space often requires additional knowledge of the node in addition to the relationship semantics. Node types can for example be used to distinguish between information objects that are metadata records and information objects that are documents.

An XML-based typology is used to define the various types and their constraints. The rules and syntax of the format is specified using the XML Schema that is listed in Appendix A.3, and a simple example typology is listed in Figure 6.7:

- A type is identified by its typename. All type definitions have a *TypeName* element that contains the type-name string. All typenames within a typology is bound to a namespace that is identified by the T*ypologyName* element.
- The node type definition consists of a list of *AllowedRole* elements that contains the name of the role types that are valid for this type of node.
- The role type is used to define cardinality constraints and what link types this role can be used in combination with. The *MinCardinality* and *MaxCardinality* elements are used to express the cardinality ratio. The letter N can be used to specify an undefined value, but otherwise the element needs to contain 0 or a positive integer. One or more occurrences of the *AllowedInLink* element can be used to list the link types this role type can be used in combination with.
- The link type definition is used to specify the degree of the link, what role type definitions it can accept as participants, a role uniqueness property and a link uniqueness property. The *Degree* element contains the defined degree for this link type as a number that needs to be greater than 1. The degree can additionally be unspecified which is expressed using the letter *N*. The roles this link type may contain are specified using a set of *AllowedRole* elements. The *RoleUniqueness* element is used to indicate whether the same role type may occur multiple times in a link instance or whether they must be unique within the scope of a link. The *LinkUniqueness* element indicates whether the system needs to verify that instances of the link are unique within the whole relationship network.

The example typology shows the *WholePart* link type and its corresponding *IsPartOf* and *HasPart* role types. The *Document* node type can participate in relationships using the *IsPartOf* and *HasPart* role types. The *Metadata* node type can only participate in *WholePart* links using the *IsPartOf* role type.

The use of an XML Schema enables the definition of specific data types and integrity rules that together enforces the typology to be consistent with respect to the data values of elements, key integrity and referential integrity. The XML Schema defines that all link and role definitions must contain unique names within the scope of the parent typology, and that the definitions maintain referential integrity, which in this context means that the *AllowedRole* and *AllowedLink* elements must refer to existing type definitions. The basic format can be further extended to support additional features like inheritance and reuse of type definitions across typologies.

```
<Typology>

    <TypologyName>DLLS</TypologyName>

    <NodeType>
            <TypeName>Document</TypeName>
            <AllowedRole>HasPart</AllowedRole>
            <AllowedRole>IsPartOf</AllowedRole>
    </NodeType>

    <NodeType>
            <TypeName>Metadata</TypeName>
            <AllowedRole>IsPartOf</AllowedRole>
    </NodeType>

    <RoleType>
        <TypeName>HasPart</TypeName>
        <MinCardinality>0</MinCardinality>
        <MaxCardinality>N</MaxCardinality>
        <AllowedInLink>WholePart</AllowedInLink>
    </RoleType>

    <RoleType>
        <TypeName>IsPartOf</TypeName>
        <MinCardinality>0</MinCardinality>
        <MaxCardinality>N</MaxCardinality>
        <AllowedInLink>WholePart</AllowedInLink>
    </RoleType>

    <LinkType>
        <TypeName>WholePart</TypeName>
        <Degree>2</Degree>
        <AllowedRole>HasPart</AllowedRole>
        <AllowedRole>IsPartOf</AllowedRole>
        <RoleNameUniqueness>true</RolenameUniqueness>
        <LinkUniqueness>true</LinkUniqueness>
    </LinkType>

</Typology>
```

Figure 6.7: XML format for type definitions.

Typenames can be identified independent of the XML file that stores type definitions. Names are represented as URIs that encodes the typology name and the typename. The purpose of this is to support the definition of globally unique typenames and to support possible reuse of type definitions across typologies etc. This work suggests the definition of a specific purpose URI scheme based on the following syntax:

Generic syntax:	*type:<typologyname>:<typename>*
Example:	*type:DLLS:WholePart*

The types defined are actively used in the creation and management of objects at runtime. Type definitions are served by a type service – the TypeLookUp object. The typeservice is used by the factory objects to retrieve type definitions. When e.g. a role is created by invoking the create method on the role factory, the typename passed as a parameter value in the create operation is used by the factory to look up the definition of this role type in the type service. The returned type definition is then used to configure the role with the corresponding predefined values for minimum and maximum cardinality. A corresponding procedure is performed for the creation of link and node objects. Type definitions are retrieved from the type service as CORBA data structures rather than XML fragments in order to avoid parsing of XML whenever a type definition is retrieved. The type service supports the retrieval of a type definition by their URI-encoded typename. Additionally it can be used to dynamically discover available type definitions by first accessing all typology names and then retrieving a list of all typenames that are available within a typology.

6.7 Constraints

The typing system introduced in the previous section defines a mechanism for assigning types to objects. The mechanisms for validating the creation of objects and the runtime structuring of the relationship network, however, is a responsibility that is delegated to the various objects.

- *Node factories* are responsible for the creation of nodes. This includes verification that the requested node type exists.
- *Node objects* are responsible for ensuring that the roles associated with the node during run-time is of the proper type, if the node type defines such constraints. The bind method is responsible for this functionality.

- *Role factories* are responsible for the proper creation of roles. This includes verification that the requested role type exists, and that the role instances are configured with the predefined minimum and maximum cardinality ratio values.
- *Role objects* are responsible for ensuring that the cardinality – the number of link objects it is bound to – is within the range of its minimum and maximum cardinality ratio. The bind and unbind methods are responsible for this.
- *Link factories* are responsible for the degree constraint. The number of roles it is passed in the create method invocation must be equal to the degree of the requested type. In addition, the link factory is responsible for the typename constraints, for example ensuring that role types are allowed for this link type and that all role types are unique.
- *Link objects* are, opposed to role and node objects, static objects that once they have been created do not change their state in any way. They may, however, be deleted, and have to catch exceptions from role objects, e.g. if the minimum cardinality constraint of the role will be violated by the deletion.

A diagram showing the basic design the DL-LinkService, including the typeservice, is given in Figure 6.8. The formal specification of the service in IDL is listed in Appendix A.1 and Appendix A.2.

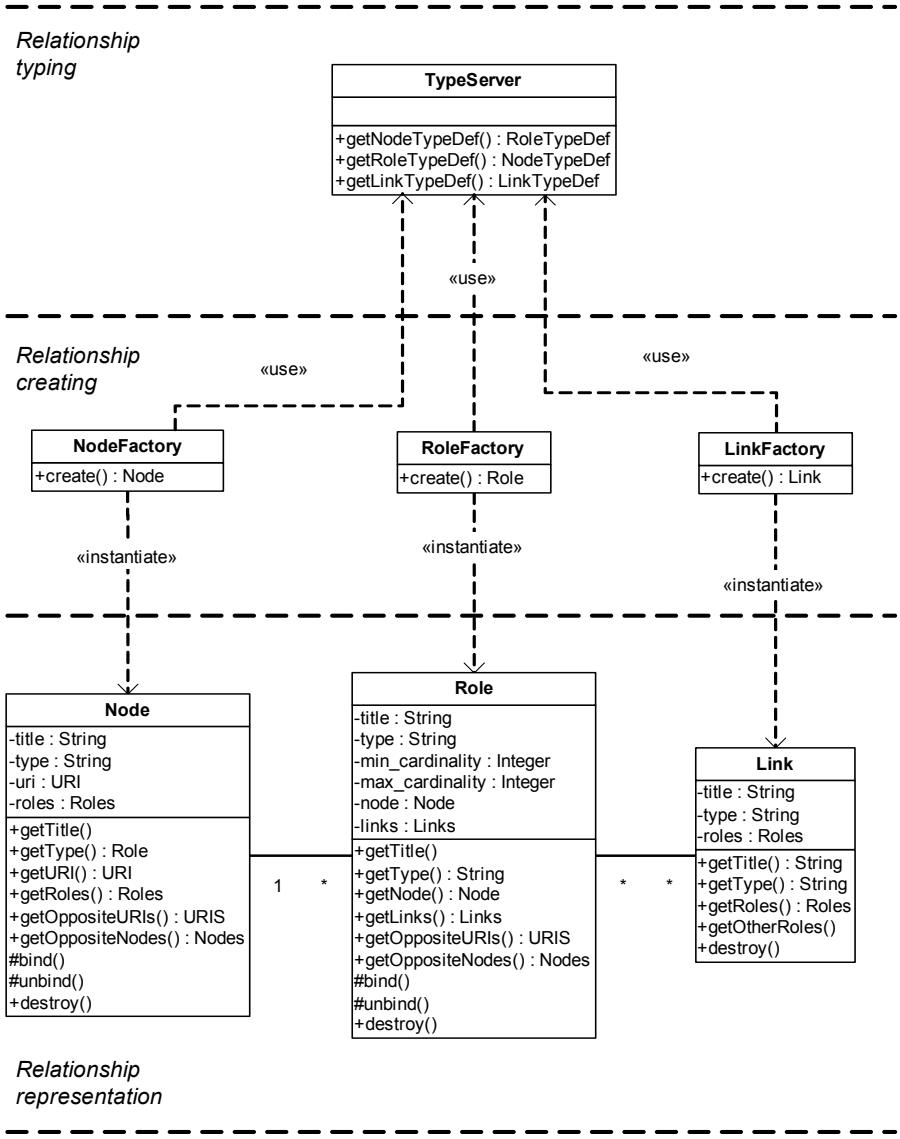


Figure 6.8: The levels of the DL-LinkService.

7 Application Issues

7.1 Distribution

The support for distribution in the DL-LinkService is highly flexible and can be configured in many ways due to the use of distributed objects and the use of URIs to reference information objects. The various axes of distribution are:

- *Distribution of relationships and information.* The service is detached from the information object layer by the use of Uniform Resource Identifiers to identify and address information objects. This inherently enables the interrelated information objects to be remote with respect to the service.
- *Interrelated information objects can be distributed with respect to each other.* URIs identifies networked resources in a uniform way and thus inherently supports interrelated information objects that reside on different systems and machines.
- *The relationship structure can be distributed.* The use of a compound structure of distributed objects to represent relationship instances enables the relationship itself to be distributed across the network. This feature is an important element because it facilitates federated and consistent link networks to be created.
- *Multiple distinct relationship networks.* Rather than using multiple deployment of the DL-LinkService to create a cooperative relationship network, it is also possible to use multiple instances of the service to create multiple separate relationship networks that e.g. can be simultaneously accessed by end-users.

A distributed deployment of the service, along any of the above listed axes, is in the DL-LinkService supported by realizing the object interfaces as a set of components. Due to the transparent object references of CORBA an object needs to be served by the same process as the factory that was used to create the object reference. This

inevitably requires that for instance nodes need to be served by the same process as the node factory. This also guides the possible decomposition of the service into deployable components, as illustrated in the UML component diagram of Figure 7.1. The node factory and node interfaces are realized by the node component, the role factory and role interfaces are realized by the role component, and the link factory and the link interfaces are realized by the link component. The type component realizes only the TypeLookUp interface. The components are coupled as illustrated by the dependency arrows.

These components can be used to create executables that fully or partial support the interfaces of the service, or components can be used if the service is integrated with other digital library software. The runtime realization of a relationship network is independent of the component layer because of the location transparency of distributed objects. The deployment diagram of Figure 7.2 shows an example executable (DLLinkserver.exe) that materializes the node, role and link components which is deployed on three hosts' machines, whereas a single instance of the TypeServer.exe that materializes the type component, is deployed on only a single host. An example of the possible network of relationships that may emerge as relationships are instantiated using the services of these three hosts is given in Figure 7.3.

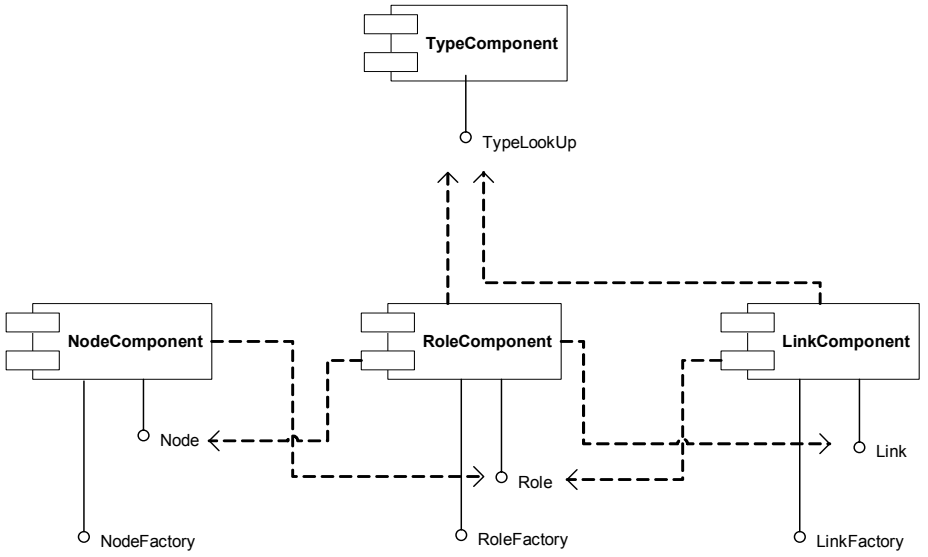


Figure 7.1: DL-LinkService components.

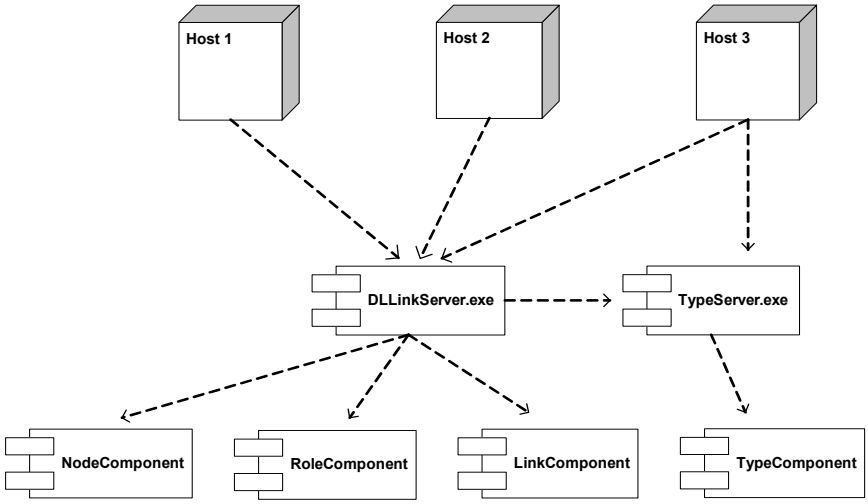


Figure 7.2: Deployment diagram.

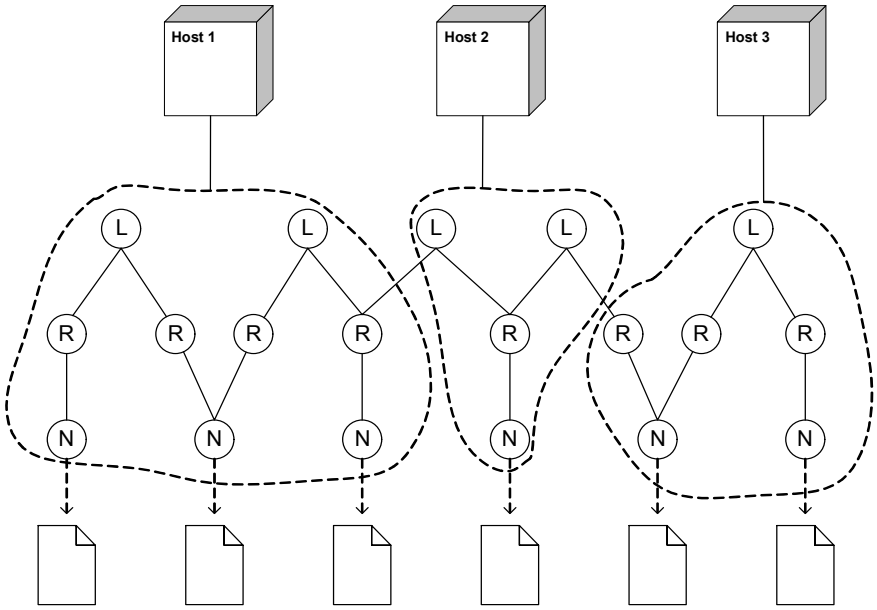


Figure 7.3: Distributed network of relationships.

7.2 Using CORBA

The main enabler for distribution in the DL-LinkService is the use of the CORBA object architecture (Common Object Request Broker Architecture) described in Chapter 2.4.6. The motivation for selecting the CORBA environment for implementing the DL-LinkService can be summarized by the following:

- The use of objects to model and implement explicit relationships is an intuitive approach.
- The encapsulation of data and functionality gives a coherent solution for users to access and navigate relationships.
- The location of an object in the network is transparent. All objects may potentially be distributed with respect to each other, whereas they in a runtime environment appear as local objects.
- An object-oriented solution enables an intuitive distribution of responsibility across the objects and accordingly across the network.
- CORBA is a well established and mature environment for developing and deploying distributed applications with well defined support for commonly needed services.
- CORBA has an adequate and flexible support for issues like runtime management of objects and object persistency.

7.3 Service Integration

The functionality and information7 that the DL-LinkService provides is not intended to be a standalone application. Relationship information is rather useless without access to the information objects the relationships relate. The integration and deployment of the DL-LinkService in specific applications, however, is highly dependent upon the general architecture of the overall digital library system it is a part of.

A main motivation in the design and development of the DL-LinkService is to meet the requirements for relationship support in a general way. Rather than focusing on a specific application or particular digital library system, the service tries to be generic. This requires, however, that it is possible to integrate the service with other digital library service or to make use of the service in client applications. Different integration scenarios will be discussed in the following subsections, and a major distinction can be made between:

- Integration with the information object layer.
- Client-side integration at runtime.

7.3.1 Integration with Information Objects

Many different digital library research projects have explored specific information objects models. A common denominator is that an information object is seen as more than stream of binary information. Information needs to be supported by object models that can represent the structure and complexity of compound information and the behaviour that is associated with the information. The DL-LinkService can be integrated with existing and future information object models in various ways; either based on the loose coupling that the node interface provides or by implementing a tighter coupling between information object and node when this is possible.

Information objects that are based on the CORBA architecture can be tightly integrated with the DL-LinkService by inheriting the Node interface. In this case, the information object itself will provide the methods of the Node interface. The only requirement for this solution is that the uri-attribute needs to be a reference to the object itself to ensure interoperability with nodes that references external information objects. This can be accomplished by e.g. using the *ior* or *corbaname* URI schemes.

A different case is the deployment of the DL-LinkService with non-CORBA information objects, but the service can still be integrated with the information object layer in this scenario. The information object will in this case be a client with respect to the node object, but can still embed a reference to the node in order to support DL-LinkService aware functionality. This, however, requires that the subsystem of the information object implements support for CORBA.

The above two examples are both illustrated in Figure 7.4. System 1 and System 2 implement different information object models, but they use the DL-LinkService in an interoperable way.

The potential advantage of integrating the DL-LinkService with the information object layer is to support the use of the service in a way that is transparent for end-users. Available relationships can easily be accessed by the information object and dynamically be integrated into the content on the server side. If the information object disseminates its content as HTML document, the relationships can be transformed to embedded HTML links.

Role and link objects are not included in this illustration and may transparently exist on both systems, on only one system, or on a third system externally to the system that implements the information objects.

Both solutions assume that the information object is a runtime entity with the ability to integrate relationships into the information it disseminates. Most information objects that exist, however, are still static structures, for instance files that are stored in file systems and retrieved from a web server.

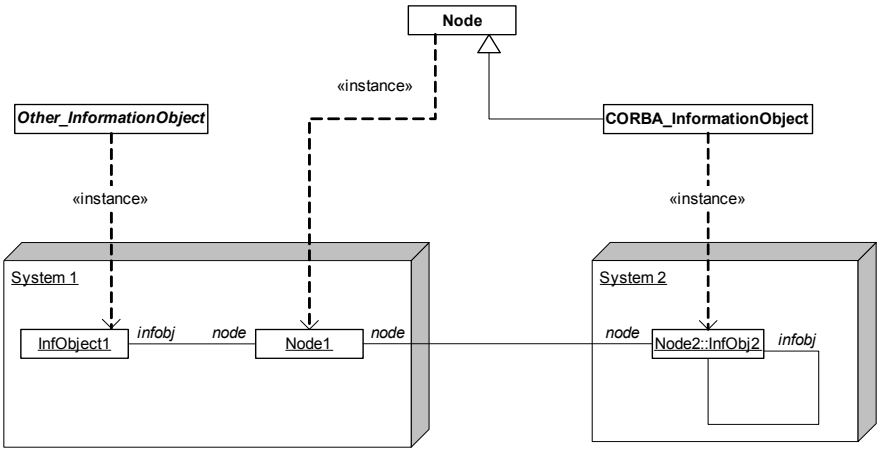


Figure 7.4: Integration with information objects.

7.3.2 Client-side Integration

The DL-LinkService basically supports the integration between information objects and relationships by the use of a uri attribute on the node. This attribute holds a reference from the node and to the information object. This design is chosen in order not to lock the service to specific document formats or information systems, but it also introduced the problem of how to integrate relationships and information objects at run time. The previous section described how the service can be integrated with specific information object models, and a different scenario is the use of the service when the connection from information object and to the node needs to be is discovered at runtime. This is a necessary solution in all deployments where the information repository and the DL-LinkService are separate services.

If an application is processing an information object and wants to know which relationships exist for this information object, it needs somewhere to look in order to find available nodes. This problem must be solved by using some kind of registry that maps from the information object identifier and to the node object that represents the information object in the relationship network.

The CORBA architecture specifies two different ways of discovering objects at runtime. The first is the CORBA Naming Service which is used to associate and resolve human readable names to objects [79]. A name in CORBA can be hierarchical and is somewhat comparable to the use of directories and file names in file systems. A set of simple names can be associated to a named context which again can be associated to another context to form a hierarchical name. Contexts are

implemented as objects, which mean that the hierarchy of contexts may be distributed over different instances of the naming service.

The second service defined by OMG is the Trading Object Service which is used to discover objects based on features [78]. The Trading Object Service is a directory where an object reference can be registered along with properties that describe this object. Object traders can be linked to form a distributed solution for object trading.

Both services can be used to support the runtime discovery of node objects. The URI of a node object can be registered as the CORBA name of this object. This solution enables clients to retrieve the appropriate node for a specific URI. This solution, however, does not include the ability to find object references that for instance are registered under different contexts; and for that reason this is not a very convenient solution. A better solution is to use the Trading Object Service to support the runtime discovery of objects. A node object can be registered with the URI of the information object it represents as a property. Applications that need to resolve a URI into a node object can then query the Trading Object Service and retrieve the node object reference. Both solutions, however, do require that node objects are registered when they are created, either by the node factory or by the client.

Other directory and naming services may be appropriate for this purpose as well, including LDAP [229] and the Handle System developed by CNRI [189, 190]. The use of naming systems allow for direct translation from an identifier and to the object reference, whereas directory services allow for looking up references based on features, where one feature needs to be the URI.

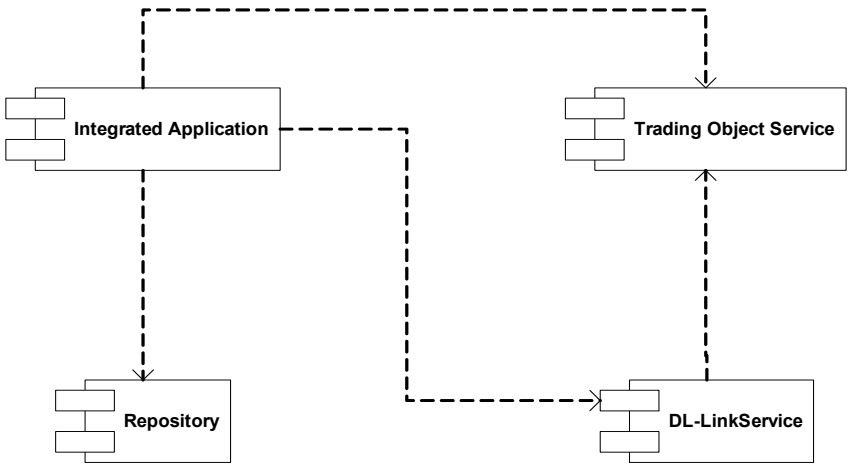


Figure 7.5: Runtime integration.

A UML diagram that shows the dependencies for applications with runtime integration is shown in Figure 7.5. The integrating application is depending on both a digital library repository, the DL-LinkService and the Trading Object Service. There is no dependency, however, between the repository and DL-LinkService in this case.

Accessing the Trading Object Service (or any other directory service) may be an operation that is performed once per session, or it can be an operation that needs to be executed more often. Applications that merely follow the relationships that are available from the DL-LinkService initially need to find a starting point in the relationships network. All other nodes and information objects can be accessed by using the references they discover when they traverse the relationship network. In case the application's retrieval of information objects is following other control paths, the application will have to look up nodes more often; the worst case would be the need to look up all nodes for all documents.

A directory may serve other purposes in applications as well. Hypermedia links are often relationships from one document fragment to another document fragment; like the link between a term and its definition. A node may be used for each fragment, leading to a set of nodes that are associated with the same parent information object. A user that views a hypermedia document often needs to retrieve all the nodes associated with the hypermedia object. This can be achieved by using a richer description of the node in the directory; like what parent document it is associated with. This can enable applications to retrieve a set of nodes for a specific document, rather than a single node for a fragment of the document.

7.4 Import and Export

A main motivation for the DL-LinkService is to serve as a tool for representing and processing relationships. Relationship information is, however, served in many ways in digital library systems and a certain level of interoperability with descriptive solutions is required. The most relevant formats to consider interoperability with are RDF, XLink and Topic Maps.

Interoperability with descriptive solutions implies the capability to import relationship statements from a specific format and the capability to export a relationship expressed in the DL-LinkService to a specific format.

The mapping between a binary 1:1 relationship as it is represented in the DL-LinkService and the more logical levels of respectively RDF, Topic Maps and XLink is exemplified in Figure 7.6.

DL-LinkService	RDF	XLink	Topic Maps
Node	Subject	Locator/resource	Topic
Role	Property	Role of locator/ resource	Topic's role in the association
Link		Arc	Association
Role		Role of locator/ resource	Topic's role in the association
Node	Object	Locator/resource	Topic

Figure 7.6: Mapping to various descriptive formats.

This mapping shows that the objects of the DL-LinkService have corresponding counterparts in both XLink and Topic Maps. This is because both these formats define the possibility to specify multi-way semantics through the use of roles. This is not the case for RDF because it is based on directed labelled graphs where the nodes in a statement fulfil different roles; one is the subject and the other is the object. Interoperability with RDF in this matter, however, can be achieved by relying on an available ontology that can provide the required semantic conversions, such as a DAML/OIL based typology [114] or the DL-LinkService's own typeservice and typology.

Additional aspects of interoperability include:

- Higher degree relationships.
- Support for multiple participation and endpoints that reflect collections.
- Resource identification.
- Type scheme interoperability.
- Format specific features.

The DL-LinkService supports relationships of higher degree, which means that the number of nodes participating in a relationship may be greater than two. This is well supported by XLink and Topic Maps, but RDF is based on the use of triples, and each statement will only contain one subject and one object. Any higher degree relationship may be transformed into a set of triples, and a certain degree of interoperability can be achieved by converting to and from such representations.

RDF, XLink and Topic Maps all support unconstrained participation. Any resource can participate in any number of relationships, and the participation in

various relationships can be distributed across different statements even though the participation semantics are equal. The DL-LinkService aggregates the participation within one distinct occurrence of a role object. This, however, is more a processing issue, and interoperability simply requires the proper transformation.

The support for aggregated endpoints can be found in both RDF and Topic Maps. An RDF subject or object may be an aggregate in the form of a list, bag or set of alternatives. Topic Maps allows for multiple topics to be enclosed within the same role. The DL-LinkService does not allow multiple nodes to be defined as the endpoint of a relationship, but a similar solution can be supported by defining a single node that aggregates other nodes by the use of relationships.

The mapping between the entities of the DL-LinkService and the various entities that are related in RDF, XLink and Topic Maps are rather straightforward. They all use URIs to identify the entities of a relationship, with the exception that XLink can have local resources as the participant of a link by means of special subelements that appear inside the extended link.

All formats support typing conventions that are compatible to the DL-LinkService. RDF and XLink both define the use of URIs for type identifiers, whereas Topic Maps uses topic identifiers. However, in the XTM format, a corresponding solution is used, since topics types are identified by topic references of subject indicators that can be expressed as URIs. The use of the same type of identifiers, however, does not imply actual interoperability for the type definitions. The possibility of dealing with type identifiers should however yield a possible interoperability, but the main problem will be the mapping between type schemes. One specific problem is the support for cardinality constraints that are part of the typing scheme of the DL-LinkService. This, however, can be solved by applying the most unconstrained cardinality constraints for “foreign” types.

A different problem is the more exclusive features that are found in the descriptive solutions. This is particularly a problem for Topic Maps and XLink, as they contain many elements with very specific intentions. The use of *titles* can be found in both of these and is supported by the DL-LinkService as well, but examples like *scope* and the various names defined in Topic Maps is not supported. A comparable problem is found in XLink where applications of XLink may define subelements and additional attributes.

The problems discussed above illustrates that full interoperability between the DL-LinkService and various other relevant formats only to a certain degree is possible. The relationships represented in the DL-LinkService can be transformed and exported to any of these formats, and the import of the basic information stored in these formats is considered to be possible. Full import and representation of all the information in any XLink or Topic Maps is not supported, however, since these formats may contain additional information not included in the DL-LinkService model. Import of RDF additionally requires an ontology that can be used to infer unknown typenames based on the available property name.

7.5 Identifiers and Uniqueness

7.5.1 Information Objects

The information objects of digital libraries are identified using many different identifier schemes and accessed using a range of different access methods. The use of a node object that merely references the information object it represents, is a generic solution that can easily be used to relate information that resides in different systems independent of the conventions that are used to store, access and identify this information. This merely requires that information objects can be identified in a uniform way, and the URI framework is a convenient way to deal with this addressing and identification diversity in a uniform way [13].

The generic URI scheme defines a syntax for identifiers that associates a scheme name to an identifier. This enables different schemes to be dealt with in a uniform way without explicitly stating how. The burden of interpreting the URI is delegated to the clients that retrieve this value who may or may not have knowledge of how to interpret the scheme.

At current the most frequent use of URIs is to encode the information that is necessary to retrieve documents by the use of HTTP and other protocols that mainstream browsers support. Other location based URI schemes have been developed to encode the communication identifiers of other systems. This includes the schemes for encoding stringified CORBA object references (ior) and CORBA names (corbaname).

URIs are defined as identifiers for both physical and abstract resources, and anything that can be identified can essentially be encoded as a URI if there is a scheme name defined for this purpose. As such, the use of URIs is not limited to identify only information objects, it can – at least in theory – be used to identify abstract entities as well. The main intended purpose of the DL-LinkService is to relate information objects, but in certain applications the URI attribute can be used to reference abstract entities as well; as a subject or a person. The use of URIs to identify conceptual units and objects inaccessible over the network is still an open issue [136], but may nevertheless be considered a viable solution based on current practice in different areas. Many recent developments in computer science presupposes the ability to deal with abstract objects using a globally valid identification system, and the use of the URI for this purpose is the most adaptive solution.

7.5.2 CORBA Objects

The DL-LinkService is depending on persistent references to enable referential integrity identity within the relationship network, and additionally requires that each

object has a unique identity within the network. CORBA supports persistent references but these references are basically communication identifiers and cannot be used for instance to compare objects.

Identifying objects by their value is not an option for the DL-LinkService because the state of an object is not sufficient to serve as a reliable and persistent identifier. The solution suggested in this work is to add a specific application defined property that holds a value that is guaranteed to be unique.

The CORBA Relationship Service addresses the issue of object identity by specifying the CosObjectIdentity module [77]. This module defines an attribute for a random numeric identifier and specifies a method that can be used for comparison of two objects. This, however, is a weak solution to identity. The value of this attribute is not guaranteed to be unique; that is, another object can return the same value. If objects return different identifiers, clients can determine that the two identifiable objects are not identical. An additional cost comes from an evenly randomized generation of numbers, which requires a centralized and shared generator of random numbers.

A better solution to this problem is to use Universal Unique Identifiers (UUID) [124, 137, 195]. A UUID is an identifier generated and represented according to a scheme that ensures uniqueness across both space and time. UUIDs are fast to generate and can efficiently be compared by arithmetically comparing the bits of two UUIDs in the order of significance.

In the DL-LinkService, the node, role and link objects are all equipped with a UUID. This is achieved by defining a UUIDObject interfaces which is inherited by the node, role and link interfaces as illustrated in Figure 7.7. The UUIDObject interface simply defines the read-only *uuid* attribute which holds the UUID as a sequence of octets. The UUID is used for different purposes; the most important is that it enables comparison of two objects to determine whether they are equal or not.

7.5.3 Link Uniqueness

Consistency in the DL-LinkService involves preventing that duplicate link objects are created. The link factory may not have knowledge of already existing relationships, and if a new link duplicates an already existing Link, this causes a redundancy in the overall relationship network that may lead to errors.

If the DL-LinkService is running as a standalone application, without any depending external instances, this can be solved by keeping track of the relationships that already have been created. Whenever a link factory is requested to create a new link, identified by a set of roles, it can easily verify whether this link already exists or not. In a distributed environment where multiple link factories may exist, this solution is unfortunately not applicable. A specific instance of the DL-

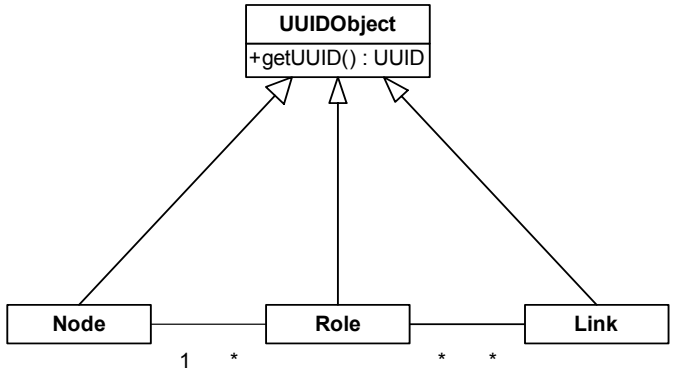


Figure 7.7: The UUID object.

LinkService will only have partial knowledge of all the link objects that exist in the overall relationship network

This must be solved by relying on the role objects that are passed in the create request. A role object will have knowledge of what already relationship instances it participates in. A uniqueness method implemented in the link factory is used to query one of these role objects for the already existing link objects it is associated to. This information can then be used to determine whether the requested link object is unique or not. It is sufficient to query a random role object for link objects, because the criterion for a uniqueness error is that all of the roles in the set are equal to all of the roles in an already existing link object. A simplified presentation of this procedure is presented in Figure 7.8. The actual comparison is based on the UUID that each object is assigned.

This process, however, may severely affect the performance of the DL-LinkService if it is conducted using remote method invocations directly. If a role is already participating in 1000 relationship instances, then the process of testing for uniqueness by this method will cause a very large number of remote method invocations. The problem can fortunately be solved by relying on cached information on role objects. The implementation of this solution is described in Chapter 8.3.

```
Link create(Role[] roles) throws DuplicateLink {
    Link[] links = roles[0].getLinks()
    for (i = 0; i <existingLinks.length; i ++){
        boolean duplicate = compare(roles, links[i].getRoles());
        if duplicate {
            throw new DuplicateLink();
        }
    }
}
```

Figure 7.8: Testing for uniqueness.

7.6 Relationship Attributes

Relationships sometimes involve attributes. Relationship attributes may be used for different reasons. Sometimes they are distinct features of the relationship, as the case is if we define relationships that resemble entities or events. A relationship that is used to express the marriage between John and Mary may include a date for when this marriage started – and maybe even a date for when the marriage ended if they have divorced. A major distinction that can be made between different categories of relationship attributes is whether they are:

- Cardinal attributes
- Non-cardinal attributes

Cardinal attributes are crucial to the identity of a relationship instance and implies that the value of the relationship attribute is an integral part of the relationship. Two relationships that are equal with respect to the participating entities can be considered as distinctively different relationships, because their attribute values differ. Consider again the marriage relationship between Mary and John. If they divorce and marry again – which occasionally occurs – an additional marriage between these two persons expresses a different marriage with respect to the date they marry.

Non-cardinal attributes refers to a different situation. Relationships may hold values that do not contribute to the identity of the relationship. Examples are the use of values to indicate the strength of similarities between two documents, or the use of values to indicate the popularity of relationship instances in terms of how often they have been navigated. In hypermedia applications, the use of link attributes occasionally is used to express various things like the originator of a link, or they may contain link annotations added by users of the hypermedia application. The reification of relationships into manageable concrete objects actually invites

exploration of the use of link annotations and other attributes. If the relationship is an identifiable resource, it is natural to assume that it can be enriched with additional information.

The support for relationship attributes is a typical "per application" requirement with respect to the kind of attributes to support and their purpose. For the DL-LinkService this needs to be solved in an application independent way, in order not to lock the DL-LinkService to a specific application. Additionally, the distributed environment further complicates this. The support for attributes needs to be flexible due to the distributed and cooperative environment. The same relationships may serve different purposes for different users, or different relationships may have different sets of relationship attributes.

This work suggests a solution based on different, but complimentary ways to implement support for relationship attributes. The first is the use of a specifically designed object to hold both *cardinal attributes* and other attributes. The second is the use of the *CORBA Property Service*, and the last is to *use the DL-LinkService* by treating a link object as an information object.

The design of a data structure for *cardinal attributes* and *non-cardinal* attributes must be a flexible and generic solution that provides for any kind of single or composite value. Flexible support for arbitrary values can be solved by many techniques, including the use of XML encoded data or a variable length list of named attribute-value pairs. CORBA defines the universal type *Any* that can hold a value of arbitrary IDL type. The use of the Any data type combined with the *DynamicAny* module of CORBA is a convenient way to deal with data structures in a dynamic and general purpose way.

If the values of a relationship are cardinal, it means that they are significant and should be taken into account when comparing link objects for uniqueness. This can be illustrated by, once more, referring to the example of John and Mary that are married twice. If expressed using the DL-LinkService's construct of role and link objects, this situation implies that two instances of link objects between these entities should be allowed; if and only if their date of marriage is different. This means that the support for attribute values must be implemented by a data structure that contains information about whether an attribute is a cardinal value or not. A possible solution is illustrated in Figure 7.9. This method may solve most problems that may occur; cardinal data types can be extracted and compared both by their type code and their value. The method does not, however, guarantee that data structures that express the same information in different ways are discovered. The proposed data type can be used in the declaration of an additional attribute on the link interface. The use of such an attribute should be restricted to relationship attributes that are known at creation time, and the attribute should be read-only, as

the altering of such data may cause unpredictable behaviour when it comes to uniqueness of relationship instances.

```
struct RelAttributeType{
    boolean cardinal;
    any attribute;
}

sequence<RelAttributeType> Attributes:
```

Figure 7.9: Relationship attributes.

The support for enriching relationships with non cardinal attributes is an easier solution to provide for. The first solution is to use the DL-LinkService itself. By creating a node that represents a link object these objects themselves may be allowed to participate in relationships as entities, e.g. by having a relationship to an annotation that is stored elsewhere. The usability of this solution, however, is questionable if the information is just a set of data values or the information is outside the scope of proper maintenance etc.

A far better solution that can serve a broad range of usages is to implement support for the CORBA Property Service [84]. This service provides the ability to dynamically associate properties with objects. The properties of the Property Service are typed, named values that exist outside the CORBA type system. The server side requirement is that the distributed object must implement the interfaces defined in the Property Service. Clients can then create and manage properties using the interfaces that this service defines. This solution should be sufficient for the many specific purpose uses of link properties that for instance are found in hypermedia applications. This feature may equally well be applied to role and node objects in order to implement an even more expressive solution for user enrichment of relationships.

7.7 Performance and Scalability

The development of distributed object applications requires that the design and implementation considers the infrastructure the application is using. An important characteristic of distributed applications is the existence of network communication that severely constrains the efficiency of method invocations, both in terms of capacity and speed:

- *Latency* is the minimum time cost of sending any message [93], and is a function of both the physical characteristics of the circuit as well as intermediate devices involved in the transfer of messages, such as routers.
- *Bandwidth* is a capacity measure of the amount of data that can be transferred during a period of time; often measured as megabits per second.
- *Marshalling rate* is the additional cost of transmitting and receiving parameters (data) over the network. This is caused by the middleware transformation between the data structures of the application and the message format of the network communication.

The difference between the speed of local calls and network calls is highly significant. A remote call is several orders of magnitude slower than a local call. An application developed in Java or C++ can easily achieve a number of calls per second in the order of millions. The call latency over a network connection is significantly slower, and additionally it will be different depending on the physical distance and routing between the network nodes. Some examples of network latency are illustrated in the table of Figure 7.10. The values are obtained by using the UNIX ping utility to obtain the roundtrip times for datagrams from a computer on the network of the Norwegian University of Science and Technology in Trondheim, Norway to various web hosts around the world. The performance on the local network is measured using the *netperf*[1] utility. The table illustrates that although all hosts are remote, some hosts are more remote than others. The actual call latency, as it is perceived at the application level, is the sum of the network latency, the marshalling rate and the bandwidth, and it will vary among different configurations of network, hardware, operating system and software. The basic performance of remote CORBA invocations in the specific setup that is further described in Chapter 8, shows a baseline performance of 4 milliseconds. This test was performed on computers communicating through a local network connection, and in this environment the middleware layer is a significant bottleneck since the actual network latency is approximately 0.2 milliseconds. In the context of global computing, however, the performance of the ORB is insignificant because the main bottleneck will be the network latency.

With a basic call latency of 2 milliseconds, a distributed application on a local network will be able to perform 500 invocations per second. A worst case scenario for wide-area computing yields only 2 invocations per second, based on the table above. Both numbers are significantly different from a local application that easily can achieve a million or more calls per second.

1. http://www.netperf.org/

Target host name	Institution and geographical location	Ping result
Local 100Mb network		0,2 ms
www.uio.no	University of Oslo, Oslo, Norway	7 ms.
www.su.se	Stockholms Universitet, Stockholm, Sweden	15 ms.
www.uni-frankfurt.de	Johann Wolfgang Goethe-Universität Frankfurt am Main, Germany	43 ms.
www.cam.ac.u	University of Cambridge, Cambridge, UK	54 ms.
www.mit.edu	Massachusetts Institute of Technology, Cambridge, Massachusetts, USA	127 ms.
www.stanford.edu	Stanford University Stanford, California, US	186 ms.
www.unisa.ac.za	University of South Africa, Pretoria, South Africa	459 ms.

Figure 7.10: Ping response time.

Techniques for improving performance and scalability of distributed applications can be categorized as either *design* level or *implementation* level. At the design level, the performance of a CORBA application can be addressed by designing objects and object interactions in such a way that the number of method invocations is reduced [25, 93, 184]. This can be achieved by:

- *Coarse grained object models.* A distinction can be made between coarse-grained and fine-grained object models. Object systems that consist of a large set of interdependent objects where each holds a limited set of attributes, can be considered as fine-grained object models. Interacting with fine-grained object models by, remote method invocations, can result in a slow system. By reducing the granularity of the object model and representing a higher level abstraction of the resource, a coarser object model can be achieved that may involve a simpler and more efficient invocation pattern. A coarser grained model, however, may conflict with the general design if the model cannot be generalized without losing important features.
- *Fat operations.* The use of fat operations is another way to reduce the number of invocations required to perform a task. Rather than accessing the state of an object using a set of methods that each returns a specific property, the use

of fat operations implies that the full state of an object is retrieved using a single method that returns a complex value. This approach will reduce the overall number of operations required, but may introduce overhead by passing redundant data.

- *Objects as values.* Not all objects implement functionality, and what initially is considered an object can sometimes equally well be specified as a complex data structure without embedded functionality. This implies that the client retrieves the data structure and can access and manipulate its data values locally rather than remotely. An extension to this approach is the CORBA support for objects as values – valuetypes – which is a mechanism for sending a copy of the object to the client. This implies that the client can access and manipulate the object locally. A main limitation of both solutions is that changes to the local state are not reflected on the remote object.

Efficiency improving techniques at the implementation level are techniques that can be deployed without changing the general object model and invocation pattern of the design. These include caching mechanisms and other run-time improvements at the ORB level, including the performance of the ORB implementation itself:

- *Caching* can be used to improve the scalability, performance and predictability of distributed systems and serves the same purpose for distributed object applications. By caching, the state of the remote object is transparently stored in a cache local to the client. Subsequent calls to the same remote object are solved by retrieving the value from the local cache rather than from the actual remote object. Main problems in caching are the efficient implementation of a caching mechanism and the issue of maintaining a local state that is synchronized with the actual state of the remote object. Solutions for caching distributed objects in the CORBA environment are reported in several research papers [38, 65, 193, 194, 205], but are not yet available as commercial products.
- *Local invocation on local objects.* CORBA enforces network transparency. At the application level, on both server and clients, the application does not know whether the object is remote or located within the same memory space – in the same process. The request/response processing of a method through the ORB is far more expensive compared to local calls, and the performance of calls between same process objects can significantly be improved if such objects are enabled to communicate more directly. This feature can be found in many ORB implementations and will significantly improve the communication between objects that reside in the same address space. This is, of course, only supported for objects that are local with respect to each other.

Nevertheless, it is a relevant feature, e.g. for applications with chained method invocations on the server side.

- *Iterators to implement predictable response time for variable sized data structures.* As a message increases in size the response time will increase due to the bandwidth limitations and the required marshalling. If the message contains data structures that are of a variable size, like lists of strings or documents passed as octet sequences, this may result in unpredictable behaviour. In such cases the use of the iterator pattern is a commonly used technique. Rather than retrieving a list of unpredictable size, the client of the invocation can retrieve an iterator that is used to sequential retrieve a limited set of list members. This does not improve the overall performance of retrieving and sending large data structures, but it can be used to establish a more predictable behaviour of applications. The client may stop iterating a data structure when the relevant information is found, or it may stop iterating the data for other reasons.

The DL-LinkService implements a fine grained and highly coupled object model, and the problem of performance and scalability is inherent in the initial design as it is laid out in Figure 6.8. This might imply problems in performance, and this is addressed by the use of fat operations and a caching scheme that is described in Chapter 8.2.

7.8 Transactions and Concurrency

Transactions and concurrency are well established fields and are widely supported by many existing systems, including data-oriented systems and process oriented systems. The purpose of transactions is to group a set of operations into one logical execution unit called a *transaction*, to enforce ACID properties – *Atomicity*, *Consistency*, *Isolation* and *Durability* [192]. Concurrency control is the ability to cope with conflicts between operations in different transactions on the same data [41].

The support for transactions and concurrency is equally important in distributed object applications as it is for other applications. If the system allows a set of read/write operations to be executed that may leave the system in an inconsistent state if one operation fails, then the system requires support for transaction. If the system allows for concurrent, possibly conflicting access to its data, then the system requires support for concurrency.

In the CORBA architecture transactional support is through the CORBA Object Transaction Service [85], and concurrency is supported by the CORBA Concurrency Service [75].

Support for transactions and concurrency are relevant features of the DL-LinkService due to the design of relationships as a compound structure. The creation or deletion of a relationship instance is a compound task that inevitably requires a minimum level of support for transactions when the link object is created or deleted:

- *The creation and deletion of node objects* are atomic operations that do not leave the system in an inconsistent state in case of failure.
- *The creation and deletion of role objects* depends on the success or failure of the bind/unbind operations on the corresponding Node. A well implemented system will, however, catch the exceptions that are thrown in case of bind and unbind failure, and the use of further transactional control is not necessary.
- *The creation and deletion of link objects* depends on the successful invocation of the bind/unbind methods on the set of role objects that are connected to the link object, and needs to be executed as a transaction. Transactional support for the creation has to be implemented in the link factory, whereas transactional support for the deletion of a link can be supported in the implementation of link objects.
- *Clients creating relationships.* The client task of creating or deleting all the various parts of a relationship instance may not require transactional execution depending on what is perceived as an inconsistent system. If the system allows for "empty" roles and nodes to exist, and the link factory and link objects implement support for transactions as described above, then there is no further need for transactions because the clients will not be able to introduce inconsistency. If a specific application of the DL-LinkService e.g. requires constraints like existence dependency expressed as the cardinality ratio 1:1, then this needs to be solved through the use of transactions. An example of this is illustrated in the listing of Figure 7.11.

Support for transactions and concurrency is an implementation issue that needs no further considerations in the design of the DL-LinkService. Implementations of the CORBA Transaction and Concurrency services are available from different commercial vendors and as open source freeware and can easily be implemented into the service.

7.9 Security

Designing and implementing any kind of vulnerable information system, distributed or not, inevitably requires security to be considered. Security is in general a complex issue involving both the definition of security policies – the general

```
begin transaction

    nodeA = nodefactory.create(uriA)
    nodeB = nodefactory.create(uriB)
    roles[0] = rolefactory.create(nodeA, roletypeA)
    roles[1] = rolefactory.create(nodeB, roletypeB)
    link = linkfactory.create(roles, linktype)

end transaction
```

Figure 7.11: Transaction.

security requirements – and the implementation of these policies using security mechanisms.

Security is the protection of assets. Assets can be physical components, like personal computers, or they can be intangible, like information and usage of a resource [121]. The OSI security architecture [158, 106] distinguishes between five classes of security services that address different aspects of a distributed system:

- Authentication services
- Access control services
- Data confidentiality
- Data integrity
- Non-repudiation services

A typical scenario in a client/server environment is that a user (principal) identifies himself to the server by a user name and then *authenticates* this identity by the use of a password. A protected network link, for instance encrypted, is then established to ensure *data confidentiality* and *data integrity* of the communication. An *access control service* is used to decide whether the client has the proper rights (credentials) to invoke the requested method or retrieve the requested information. The *non-repudiation* service may be a log that records and stores invocations to document the events of the session.

Security for distributed object systems is just as important for distributed object applications as it is for traditional client/server applications. The nature of distributed object applications makes security a complex issue for different reasons:

- *An application may involve numerous distinct objects.* The number of objects in a distributed object application may range from a single object to an unlimited number of objects.

- *Objects may switch between the role of a server and client.* A specific object may play the role of a server for one request and play the role of a client when it invokes methods on other objects.
- *Method invocations may be chained.* When a client invokes a method on a remote object this may include a chain of remote invocations to be called on other objects in order for the servant objects to process the request from the client.
- *Different granularity of access control.* Some applications can use a peer to peer access control, others may require a per object access control or a per method access control.

For a local systems running on a single computer or a distributed system on a closed network, the surrounding environment may provide for the needed security. Systems that rely on the use of open networks to share resources, like the Internet, do however require security in order not to expose the assets to unrestricted access and other threats.

The security requirements of the DL-LinkService are highly dependent on the environment it will be deployed in. Security is an implementation and runtime issue and should not affect the design of a distributed application. The most relevant and complex security requirements are concerned with access control. Some security scenarios that illustrate alternative security policies to deploy with the DL-LinkService are:

- *Unrestricted access to the service.* The DL-LinkService may not require any implemented security mechanisms if the service is deployed in a closed environment where all clients are trusted and reliable and for that reason do not represent any threat.
- *Access control to the service in general.* This policy implies that a client may be granted access to all methods on all objects – or not.
- *To prevent unauthorized access to specific objects.* This policy requires that the server side of the invocation is able to control access on a per object basis. All users may be granted access to node, role and link objects in order to allow the navigation of an already existing relationship. Additionally, other users may be allowed to create objects by being granted access to the factory objects.
- *To prevent unauthorized access to specific methods*. This policy requires that the server side is able to control access on a per method basis. All users may be granted access to read-only operations on the node, role and link objects in order to allow the navigation of an already existing relationship. A limited set of trusted users may additionally be granted access to those methods that are

used to create or delete a relationship instance. This is, in fact, a more relevant policy than the per object policy described above.

- *To constrain the invocation of chained method invocations.* Method invocations may be chained, and this may require the support for credential delegation. An example of this is the creation of a role object in the DL-LinkService. If a client is invoking the create method on a role factory this causes the factory object to invoke the bind operation on the specified node object. Whether or not the node should grant access to this method can be depending on the factory objects credentials, the client credentials or the credentials of both. The latter two solutions are depending on the support for a credential delegation mechanism.

One feature of the DL-LinkService that needs special attention is the bind and unbind methods on the node and role objects. These methods are used to establish and remove the two-way references that exist between objects, and the proper use of these is particularly important to prevent disruption of the overall relationship network. This inherently imposes the requirement that only the proper clients should be allowed to call the bind and unbind methods. In the class diagram of the service this is, for convenient representation, specified as a method that has limited visibility[1]. CORBA does not, however, have support for restricting access to objects like what can be found in programming languages like Java and C++. This issue needs to be solved by the use of security mechanisms.

Security for CORBA distributed objects is supported by the CORBA Security Service. This service defines the core security facilities and interfaces required to ensure a reasonable level of security of a CORBA-compliant system as a whole [81]. The implementation of access control into an application can be either *security-unaware* or *security-aware* [121]. A security-unaware application deals with access control in the ORB layer. In a security-aware application the credentials are exposed to the application, and the application can enforce its own application-specific access control policies.

An additional CORBA specification that deals with security is the Common Secure Interoperability Specification (CSI) that defines the Security Attribute Service protocol (SAS). The protocol provides client authentication, delegation, and privilege functionality that may be applied to overcome corresponding deficiencies in an underlying transport [82]. Finally, the specification of the Resource Access Decision Facility (RAD) defines a mechanism for obtaining authorization decisions and administrating access decision policies. This facility enables a common way for

1. In the UML terminology a visibility value of *protected* indicates that the given element is visible outside its own namespace only to descendants of the namespace [76].

security-aware applications to request and receive an authorization decision [80]. The facility is intended to be used by security-aware applications.

Support for security in the DL-LinkService is an implementation issue and does not need to be included in the design. Compared to the availability of the transaction and concurrency service, the implementations of the complete security architecture of CORBA is, at present time, sparser. This may be caused by the inherent complexity of this architecture and the following costs in developing implementations. Some implementations, however, are available. The support for core level peer to peer security by the use of IIOP over Secure Socket Layer (SSL) is, on the other hand, more widely supported.

8 Implementation and Testing

8.1 The Prototype Implementation

A prototype of the DL-LinkService is implemented as a part of this work. The prototype is the main tool to verify that the proposed architecture is an applicable solution, and this work attempts to achieve such verification by documenting that the proposed solution:

- Can be implemented.
- Does not have severe performance or scalability problems.
- Can be used to solve a real world problem.

The first two bulleted items are examined in this chapter by describing the implementation of the service and the testing that has been performed. The latter item is the topic of Chapter 9.

8.1.1 Supported Features

The developed prototype reflects the proposed solution by implementing the functionality needed to verify the core features of the DL-LinkService architecture. The prototype implements the following:

- The core object model of node, role and link objects and their corresponding factory objects.
- A type service that serves type definitions.
- An XML based typology format for the definition of types.

The prototype implements only a subset of the features discussed in previous sections. Certain features like support for transactions and security have not been addressed because these are generic services that have been designed to work with any CORBA distributed object application. Other features like the support for relationship attributes by the use of the CORBA Property Service are not

investigated, because they are regarded as extended features. A simple directory lookup mechanism, however, is included in the prototype and is based on an ad hoc solution for convenient reasons. It can be viewed as a simplified front end to other lookup services.

The performance of the DL-LinkService is addressed in the implementation of the service by the use of:

- Fat operations to delimit the number of required remote method invocations to perform specific tasks.
- Role objects that cache information about the roles at the opposite end of a relationship in order to reduce the need for chained method invocations.

8.1.2 Interface Definitions

The formal specification of the DL-LinkService is based on the use of the CORBA Interface Definition Language. Design diagrams of earlier chapters have mainly been used to present the actual objects involved, but the proper specification of the service, as it is implemented in the prototype, is the IDL definitions listed in Appendix A.2 and Appendix A.1.

The sole purpose of the Interface Definition Language is to allow object interfaces to be defined in a way that is independent of particular programming languages. It is a declarative language for specifying interfaces and types and can not be used to define application behaviour. An interface is the object's contract, it defines the object's method signatures; the name of the methods and the parameter types of the methods.

The DL-LinkService is specified as a set of IDL modules that together makes up the formal description of the DL-LinkService. The module construct of IDL is a convenient way to group logically related interfaces within a common namespace. The DL-LinkService is defined using two modules:

- The *Typology* module is used for the interfaces and data structures that are defined for the type system.
- The *LinkService* module defines the node, role and link interfaces and their corresponding factories as well as other related data types.

This does not directly reflect the possible layout of runtime components, but is convenient to avoid an unnecessary use of namespaces and redeclarations that are caused by the coupling between these interfaces. The prototype additionally includes a node lookup module which simply defines an ad hoc directory service. This is kept as a separate module since it is not considered to be a part of the service and is not included in the documentation of the service.

Each module consists of a set of definitions that declares:

- *Type definitions* that define application dependent compound data types that are used across several interfaces. Certain data types are mere redefinitions of basic types to signal a stricter semantics for the data type.
- *Interfaces definitions* that define the object types of the DL-LinkService.
- *Attributes definitions.* Attributes are used to define only readable attributes. Such attributes are converted to get methods by the IDL compiler.
- *Operation definitions* that define the name of the method and its signature in terms of what attributes it accepts and the direction these attributes are passed.
- *Exceptions* are declared for certain operations and are used to exit from erroneous situations, e.g. to signal that an operation is not carried out.

IDL definitions are the initial starting point in the implementation of all CORBA applications. The file or set of files that are containing interface and other type definitions are used as input to an IDL compiler that generates source code in the form of *stub* and *skeleton* code.

- The *stub code* is the client side implementation of a remote object reference – the local object through which the clients make requests. Often this is referred to as a proxy object, since the stub is the client side surrogate for the real (remote) object. Stub objects appear as ordinary objects that the client can invoke methods on – in the same manner as methods on local objects are invoked. The actual use of a remote object is transparent, and there is nothing for the engineer to implement in the stub code.
- The *skeleton code* is the server side equivalent of the stub. The skeleton code is the basis for the implementation of the behaviour of objects. The task of implementing object behaviour is merely a question of implementing the behaviour of the methods defined on the interfaces. An actual implementation instance of a CORBA object is often referred to as a *servant*, and the implementation of applications that serves object servants often has to have considered other issues like the runtime management of servants and object persistency.

8.1.3 Implementation Issues

The implementation of the prototype DL-LinkService included the following tasks:

- Implementing object behaviour.
- Implementing support for persistent objects.
- Implementing run-time management of servants.

The implemented behaviour of objects basically followed the design that is described in Chapter 6. Additional behaviour and responsibilities are introduced by the caching scheme as described in Chapter 8.2. The implemented support for persistent state and run-time management of servants is described in the following sections.

The DL-LinkService prototype is implemented in Java for various pragmatic reasons:

- Java produces platform independent byte code, and the same software can be run on different platforms without having to be recompiled.
- Java natively supports garbage collection that recycles unused objects in order to free the memory they occupy.
- Java has a well documented and broad support for various basic level needs like string processing, dynamic arrays, etc. There are a great number of components available for additionally needed functionality. One such component that is used in the prototype is the Java UUID Generator (JUG)[1].
- Java applications can easily be integrated with database back-ends through the use of the Java Database Connectivity API (JDBC).

Other programming languages may to some degree produce better performing executables than Java. The efficiency of the internal within-servant processing, however, is not an important feature for the DL-LinkService. The major problem in many distributed object applications is the invocation pattern, rather than the internal performance of methods.

When using an object-oriented programming language in the development of distributed objects, implementers can choose between inheriting the generated skeleton classes when implementing the behaviour of objects or to use the tie mechanism of CORBA if inheritance is unwanted. The prototype makes use of inheritance.

The DL-LinkService prototype consists of a set of classes that includes those that are automatically generated by the IDL compiler, those classes that implements the behaviour of the system, and additional classes the prototype is dependent of.

1. http://www.doomdark.org/doomdark/proj/jug/index.html

The decomposition of the system into units for later composition is only delimited by the CORBA constraint that the creation of object references is coupled with the POA that created this reference. Different executables or components can for this reason be constructed by combining one or several of the following indivisible components:

- A type service that supports the type service interface
- A node service that supports the node and node factory interfaces
- A role service that supports the role and role factory interface
- A link server that supports the link and link factory interfaces

Motivation for using finer grained components can e.g. be if a digital library repository would like to support the ability of information objects to participate in relationships, but would like to rely on an externally maintained type service, and does not want to maintain the role and link objects that users of the repository will create.

The prototype developed as part of this work is based on one executable that implements node, role and link interfaces and their factories. A standalone executable is used for the type service. The setup that was used during the testing of the service is shown in Figure 8.1. The figure shows a client and two DL-LinkServers that are running on different machines but uses the same database server. The type service is running on a separate machine.

8.1.4 The Object Request Broker

The CORBA runtime environment is usually referred to as the Object Request Broker (ORB). The ORB is essentially the software component that is needed for transparent communication in distributed object applications. Additionally, the ORB supports the runtime management of object implementations on the server side and other core tasks that are needed in the deployment of distributed applications. Object Request Brokers are developed by a range of different vendors, and are available both as commercial and free software. Java and C++ are major target platforms, and there is a range of options to choose from for these programming languages. Most ORB implementations include an IDL compiler and other tools that are necessary to develop and deploy CORBA applications.

Later versions of CORBA have addressed many of the portability problems that were discovered with earlier versions of CORBA. ORB implementations that support version 2.2. and higher are generally portable. This means that CORBA applications can switch between different ORBs (in the same programming language) without requiring adaptations of the code. ORBs may, however, differ in terms of what CORBA version they support, and as new versions of CORBA

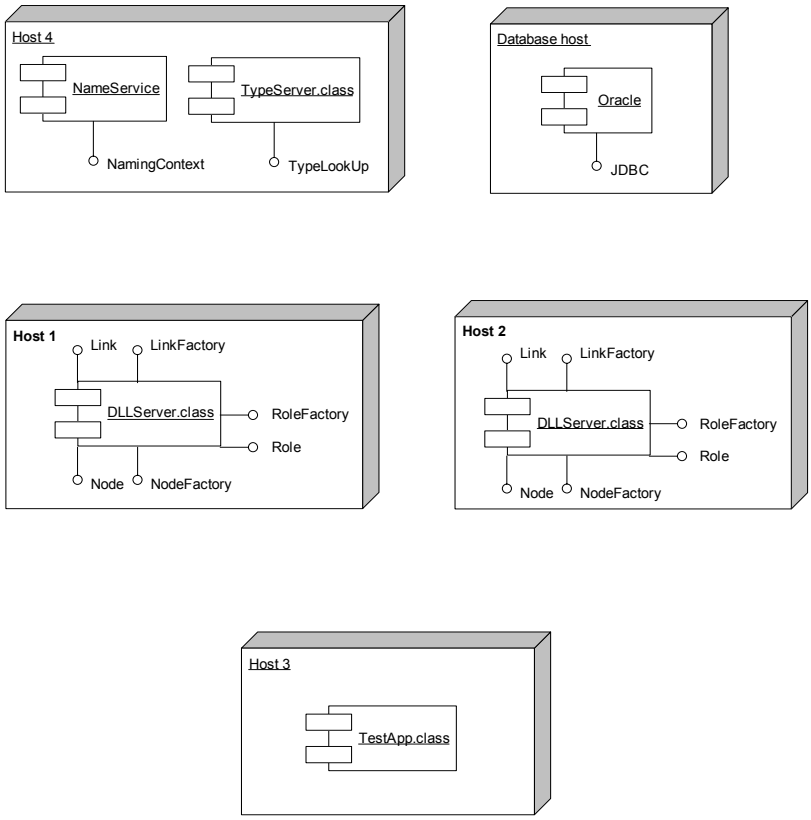


Figure 8.1: Test setup.

introduce new features there may be a corresponding difference in which feature of CORBA different implementations support. Other features that may be different are the performance of the ORB.

This project has been using different ORBs in the implementation of the prototype. The final version of the prototype is using the OpenORB[1] software, which is open-source with a BSD-like license that supports version 2.4.2 of the CORBA specification.

1. http://openorb.sourceforge.net/

8.1.5 Runtime Management of Servants

The object servants are the runtime implementation counterpart to what the client perceives as the distributed objects. The mapping between object references and the runtime servants are in CORBA supported by the Portable Object Adapter. The main responsibility of the Portable Object Adapter is to ensure that an invocation, whether local or remote, reaches the correct servant. The POA is a highly flexible architecture that can be tailored to specific solutions through the use of policies. The main responsibilities for the POA are [93]:

- The POA is responsible for *the creation of object references*. Object references can be created when servants are instantiated, or they can be created independently on the instantiation of servants.
- Objects are identified by *object identifiers* within the scope of the host POA. The generation of object identifiers can be managed by the application, or unique identifiers can be assigned automatically be the POA.
- The *mapping between servants and object reference* can be managed by the POA, or it can be conducted by the application itself by using servant managers. The POA defines interfaces for two patterns that can be used; servant locators and servant activators.

When a DL-LinkService factory object receives a create method request, it fetches the type definition from an available type service and performs the necessary checks to verify that the object it is about to create conforms to the type. The factory then generates a UUID for this object and creates an object reference with the UUID value as the object identifier. Then it updates the database with the initial state of the object it has created a reference for, and returns the object reference as part of the handle data type in the response to the client. The actual instantiation of a servant for this object is deferred until the client invokes a method on the object.

The DL-LinkService prototype uses servant locators to manage the mapping between servants and object references, which enables a high level of control over the management of servants. The servant locator object provides a method that is called whenever the POA receives a request and a method that is called before a response is returned. The servant locators implemented in the DL-LinkService maintain a map of active objects. If a servant locator receives a request to an object that is not “alive”, it is responsible for the incarnation of this object. This includes the instantiation of a servant with the proper state. The servant object is passed back to the POA which then is able to call the correct method on the correct object. If the object is already “alive”, the servant locator finds the correct servant and passes this back to the POA.

Any server has limited capabilities in terms of memory and other resources. If the number of objects that the factory has created is large, then there is a need to

manage the memory usage of the process. This requires that unused servants must be deleted in order to avoid the consumption of memory as new objects are created. In the DL-LinkService prototype this is solved by the use of an *evictor pattern,* which is commonly used for this purpose in CORBA applications [93, 16]. The evictor pattern is a general strategy for limiting memory consumption, and basically means that certain objects are removed from memory. The main difference between various evictor patterns is the policy that is used to determine what objects to delete. Example eviction strategies are LRU (Least Recently Used) and LFU (Least Frequently Used). The DL-LinkService implements the LRU eviction pattern by keeping track of which servants are least recently used. This is implemented by the use of a Java Hashtable that contains the mapping between object identifiers and the servant object. An additional Java Vector is used to track what objects to evict. When objects are invoked they are moved to the last available position of the vector. When the capacity of active servants is reached, in terms of a predefined limit, the servant manager will evict the least recently used servant from the beginning of the vector, to make space for a new servant to be incarnated.

8.1.6 Persistent Objects

CORBA supports both transient and persistent objects. The lifetime of a transient object is limited by the lifetime of the serving process. A persistent object, on the other hand, outlives the operating system process that serves the object. This means that the same object reference can be used independent of the lifetime of the server process. Objects that represent persistent information, or for other reasons need to be persistent, must be implemented as persistent objects, whereas objects that do not have persistent state can be implemented as transient objects.

The notion of a persistent CORBA object involves both the persistent object references and the mechanism that is used to maintain the persistent state of the objects. The first is a feature supported by the basic CORBA environment; the second needs to be implemented using an appropriate solution for storing and retrieving the state of objects.

The feature that enables persistent references in CORBA is the ability of object references to hold information that the server process can use to uniquely identify the state of the object: object identifiers. In the DL-LinkService the UUID of objects are used both as keys in the database backend and as object identifiers.

Persistent objects additionally require some kind of persistent storage solution that outlives the processes that creates and serves object servants – typically by the use of the file system or a database. CORBA defines the Persistent State Service that can be used to add transparent persistency to object implementations [83], but persistency can equally well be implemented ad hoc.

In the DL-LinkService the node, role and link objects are highly persistent. Each of these objects represents information that needs to be persistent for an undefined length of time. Objects may be deleted, but the creation and deletion of objects need to be independent of the activation and deactivation of the process that serves these objects.

The various factory objects and the type service, on the other hand, can be transient objects. They represent functionality rather than information and clients need not be able to reference these objects using the same object reference, but can acquire these references for instance from a naming service.

The DL-LinkService supports persistent objects by the use of an Oracle database backend. The initial state of an object is stored in the database when the object is created, by its corresponding object factory. When a request arrives for this object the servant locator uses the object identifier of the reference to retrieve the state of the object from the database and instantiate a servant object. Some methods include the altering of the object's state (bind/unbind methods), and requires that the database is updated accordingly. The state of the object is removed from the database when it is properly deleted.

8.2 Performance and Scalability

The DL-LinkService implements a fine-grained object model for relationship instances that potentially can cause performance and scalability problems. This is mainly due to the complex invocation pattern that certain tasks require. Early test performed on the service showed that for instance uniqueness testing performed badly as the cardinality of roles increased. Two techniques were applied to solve this problem:

- The use of fat operations.
- The use of a caching mechanism.

In the following, the abbreviation RMI will be used as a general expression for a remote method invocation; which is the call to a distributed object from a client. The client of the invocation can be another distributed object or a client application. The real cost of an RMI is highly dependent on the network latency and can additionally be different between ORB implementations. One way to determine a performance measure of a distributed object system is to calculate the number of RMIs for a given task. An additional performance cost due to bandwidth and marshalling limitations, is the size of the passed data structures. This, however, is a less apparent problem in this prototype due to the small data structures that are

passed in method invocations. An additional technique that needs to be implemented for certain operations is iterators.

8.2.1 Fat Operations

Fat operations can be characterized by having parameters that send rich information structures, and the main purpose is to prevent the use of an excessive number of method invocations to perform a task. Rather than sending single data values to and from operations, methods in the DL-LinkService pass richer data structures that contain information about the essential characteristics of an object. Attributes, like UUID, title, and type, are static throughout the lifetime of an object. When such attribute values are assigned to objects they do not change during the lifetime of the object, and can conveniently be passed to clients for later use without causing consistency problems:

- Static attribute values for the node object are
  UUID, URI, Title, Type
- Static attribute values for role objects are
  UUID, Reference to related node, Title, Type
- Static attribute values for link objects are:
  The list of role references, Title, Type

The DL-LinkService passes the state of node, role and link objects as compound data structures. The IDL types NodeHandle, RoleHandle and LinkHandle is defined for this purpose. Information does not have to be obtained through handles, but can additionally be obtained by accessing each distinct attribute whenever needed.

Other aspects of the objects are more dynamic, such as the dynamic list of references to associated objects that the node and role objects maintain, this data is not included in the handle types to prevent possible inconsistencies.

One example of how the use of fat operations can improve performance is in the navigation task. If the navigation includes iterating the retrieval of the three attributes of title, type and URI from a set of 10 target nodes, the total number of RMIs adds up to 30 RMIs. By using fat operations that rather return the state of each node object as a compound data structure, the same information can be obtained from the opposite nodes using 10 RMIs. In terms of distributed computing, the reduction from 30 to 10 remote method invocations is a considerable improvement. In this case, the total size of the attributes does not introduce any significant increase in the transferred data.

8.2.2 Caching

An important function of the handle data types in the DL-LinkService is that they enable caching. Caching is an important element to establish a solution which is both well performing and predictable in terms of scalability. The DL-LinkService prototype implements a simple but highly efficient caching scheme. Caching only occurs on the role objects, and the main principles of the caching scheme are:

- The role object stores information about the node it is associated to. This information is passed from the node to the role as a return parameter of the bind method on the node.
- The role object stores information about the opposite roles and nodes of the relationship instances it participates in. Link factories receive role handles as a parameter of the create method. Each role handle includes a node handle as a subpart. This information is forwarded to each of the roles in the bind method invocations that the link factory invokes on the roles.
- Whenever a role is deleted it returns the role handles of the opposite roles and can delete the corresponding entries in the cache.

The use of the bind/unbind methods to update the cache ensures that the cache always reflects the right state of the system. Figure 8.2 depicts the flow of information in the caching scheme as dotted arrows. Role objects R_1, R_2, R_3 and R_4 cache the information about the node they represent. Role object R_1 caches information about R_2, R_3 and R_4 including the node information that is already cached in these roles. R_2, R_3, and R_4 cache information about R_1. The effect of the cache is that each role can return for instance the opposite node's URI without having to use a chained sequence of invocations to perform this as shown in Figure 8.3.

Without the cache, retrieving the URI of the opposite node of a relationship basically requires 5 RMIs. For relationships of higher complexity, navigation is depending on both the participation cardinality of the starting node and the degree of the relationship instance expressed as:

$$2 + 3\langle cardinality \times \langle degree - 1\rangle\rangle$$

By using the cached information the same task can be achieved by 2 RMIs. This solution is independent of the number of opposite roles and nodes, because it only requires, first, that a reference to the role is retrieved from the node and, second, that the list of endpoints (roles, nodes or URIs) is retrieved from the role.

The cache can additionally be used to efficiently implement a link uniqueness test. Without this, the only way to determine whether a link is unique is by examine one of the roles to see if this role already participates in such a relationship instance.

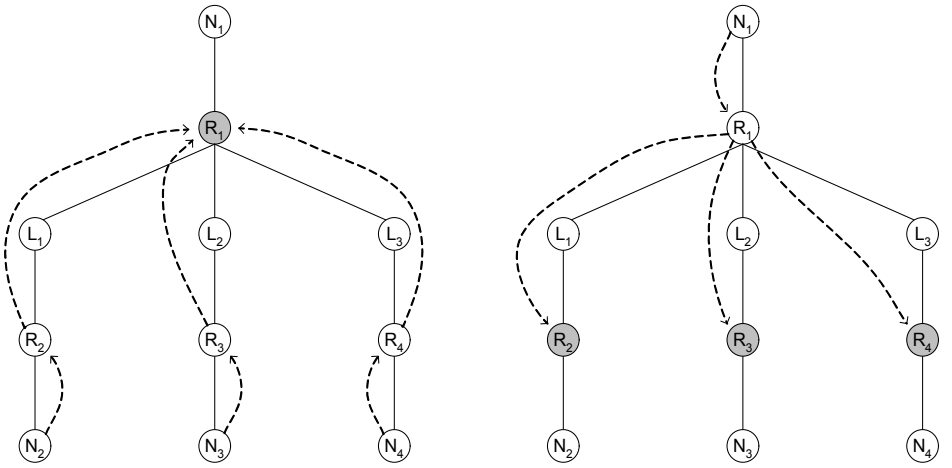


Figure 8.2: The flow of information in the caching mechanism.

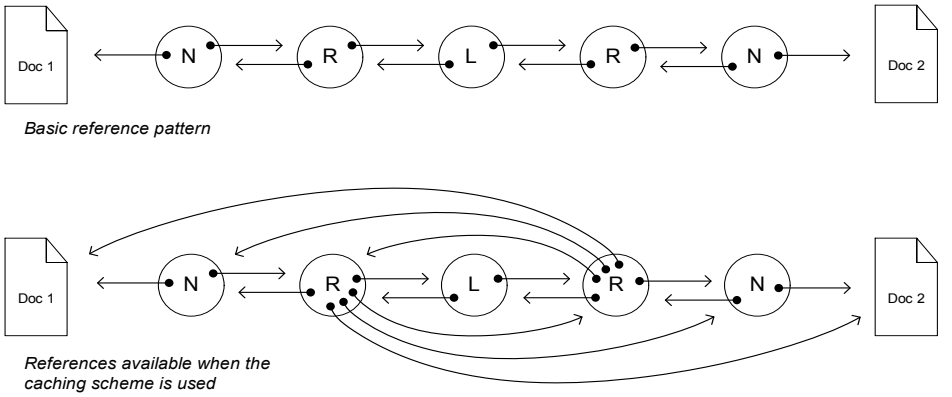


Figure 8.3: The direct references that the cache enables.

This introduces a severe bottleneck if the role participates in many relationships instances, and uniqueness testing has to be verified against all the relationship instances it participates in. If this testing can be performed using the cache, a significant performance gain can be achieved.

There are other positive side effects of using such a cache as well, and the benefits can be summarized by the following:

- It enables efficient uniqueness checking.
- It improves navigation significantly.
- It enables a fallback navigation in case remote objects are unavailable.

The main drawbacks are that each role host now has to store additional data, for each role object and that the solution requires more data to be passed in bind and unbind methods.

8.2.3 Object Locality

The location of objects is transparent when using CORBA. The client of a distributed object does not have to deal with the locality of the object, since methods are invoked on distributed objects in the same way as methods are invoked on local objects. This transparency, however, is sometimes a bottleneck because it forces all communication to pass through the ORB. This is particularly a problem for objects that exist in the same memory space of the same operating system process where the communication overhead introduced by the ORB is quite superfluous. This is addressed in many ORBs by implementing support for transparent local calls when objects are within the same memory space. The deployment of the DL-LinkService can take advantage of this by configuring the system in such a way that collocated objects are promoted. This should indicate that the use of one executable for the node, role and link component is more favourable than running distinct processes for each component.

The question of locality is additionally important with regard to improving the pattern of frequently occurring tasks. The caching mechanism of the previous section is based on the principle that all roles should cache the state of opposite roles and nodes. For a single instance of the DL-LinkService this information is already available, because the state of objects is persistently stored in the database. The database can then be queried to obtain information about opposite ends of relationship, and the database can be used to determine the uniqueness of links at creation time without invoking methods on any object. This, however, is not a flexible solution because the DL-LinkService is based on the idea of multiple cooperating link services with a resulting distributed storage of information. Combined use of the persistent storage of local objects and the cache can, on the

other hand, be combined to reduce redundant storage of information. The same database tables can be used to store cached information and the state of local objects in a transparent way.

8.3 Performance Testing

The main motivation for the testing of the DL-LinkService was to evaluate the design with respect to the scalability and performance of the model. Evaluating the scalability and performance of an actual service is an extensive issue that includes all levels of processing that occur in software and hardware as well as the way different components work together. The testing described in this section is merely concerned with certain aspects of performance and scalability – the ones that are related to the design. The performance test attempts to answer questions like:

- How does the distribution of objects affect performance of navigation?
- How does the service behave when the cardinality of a relationship increases?
- How does the service behave when call latency between objects increase?
- How does the use of bind/unbind operations in the creation/deletion of relationships behave with respect to the distribution of objects?

The absolute measurement values that are made are highly related to the specific test environment and the specific prototype implementation. A better overall performance can be achieved in numerous ways, such as a more efficient database backend, using C++, and running the system on more well performing machinery. Performance is in the following test measured only for the purpose of comparing different configurations.

8.3.1 Test Setup

The test environment consists of two machines dedicated to run the Java based DL-LinkService software. The computers used for this purpose are deliberately on the lower level of performance compared to what is available today[1]. Client applications are run on a better performing machine[2] in order to contribute as little as possible to the measures that have been made. All machines are on the same 100Mb Ethernet local area network. An Oracle database on a dedicated machine is accessed by the use of JDBC in order to store and retrieve the state of objects.

1. 450 Mhz pentium single processor, 256 Mb memory and Windows XP Professional.
2. 1,8 Gb pentium single processor, 256 Mb memory and Windows XP Professional

The basic network latency for a round-trip ping operation is approximately 0,2 milliseconds, and is measured with the netperf[1] utility. The actual performance of CORBA in the test setup is more difficult to determine. A roundtrip operation that sends and receives a sequence of 256 characters is 4 milliseconds, measured as the average of 1000 operations performed in a loop. In general there are many factors that will affect the basic performance:

- *The ORB implementation* from different vendor will behave differently, some ORBs are performing better than others due to e.g. more efficient marshalling.
- *The type and the size of data* transferred. Primitive types like a sequence of octets involve less marshalling than complex structures. Sending a small chunk of data is considerably faster than transferring larger chunks of data.
- *Whether methods receive and send return data*. Methods that do both typically perform slower, because there is marshalling involved when the client submits a message, when the server object receives a message, when the server sends a return value and finally when the client receives this value.
- *The persistency mechanism* that is used to store, update and retrieve the state of objects.
- The *behaviour of the Java Virtual Machine* involves the loading of classes when new object classes are first used, which significantly can slow down first time performance of a method.

The various methods implemented in the DL-LinkService typically have call latencies in the range of 1-20 milliseconds. These numbers are specific to the setup and are highly related to the performance of the underlying software, hardware, the database that is used, etc. Additional latency is added by the use of servant interceptors that halts the passing of request and response messages for a specified amount of time. This is used to simulate 20, 40, 60, and 80 milliseconds latency. The simulated latency is added in such a way that it does not add a delay for local invocations.

Most tests have been performed using loops of variable size to diminish the effect of performance peaks and other sources of errors. The use of loops that perform the same operation 100 or 1000 times is sometimes necessary to achieve stable results from one test to another. This does introduce the problem, however, that the sequential execution of an operation typically performs better – mainly due to performance of the Java runtime environment and the performance of the database backend.

1. http://www.netperf.org/

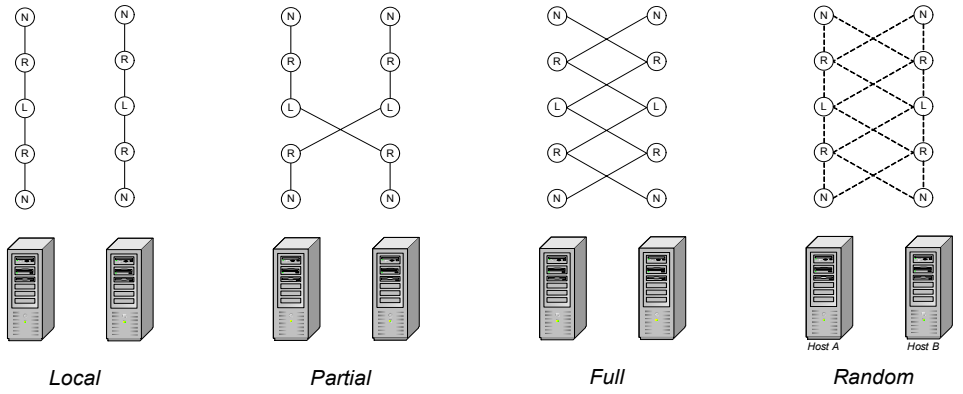


Figure 8.4: Different distribution of the objects in test setup.

The following tests are based on four different patterns of distribution across two host machines:

- *Local:* Two nodes and their interrelating role and link objects are situated on the same machine. The total set is distributed evenly across the machines and there are no object references from the objects on one host to the objects on the other host.
- *Partial:* This configuration uses a partial distribution of the relationship instances across the two machines. A node and its associated role are on the same host, but the opposite node and its associated role are on the other host. The link objects are evenly distributed across the two hosts.
- *Full:* In the full distribution of a relationship all the objects of the relationship instance are remote with respect to each other.
- *Random:* In the random configured test setup, the objects are randomly distributed across the two hosts.

These different configurations may reflect different uses of the service with respect to distribution. The *local* setup represents a typical client/server setup and the relationship network does not include any real distribution of the relationships. There are no paths from the relationships of one network to the other and each host maintains separate relationship networks. The *partial* setup represents a distribution where the endpoints are on different hosts, but each relationship only contains one reference across hosts. This could be a typical use of the DL-LinkService for a collaborative use of the service to create a shared relationship network. The *full* distribution setup is a typically worst case scenario where all objects are remote

with respect to each other. The *random* setup is included for testing purposes. If the testing had included more hosts the random setup would have behaved more like the full distribution, but when including only two machines the chances that an object reference points to a truly remote object is 50%.

8.3.2 Navigation Performance

The design of the DL-LinkService outlines two different implementations of the navigation task:

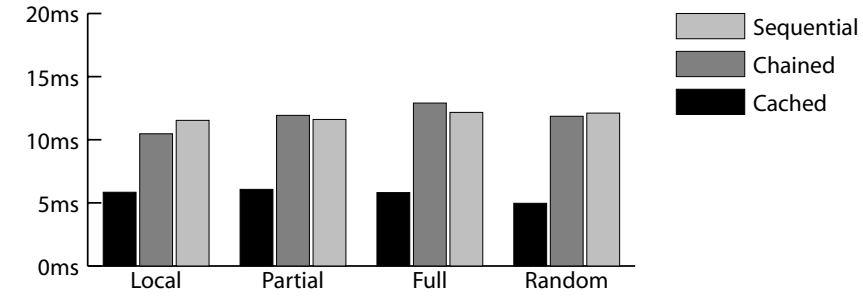
- Sequential
- Indirect

The *sequential* way of navigation simply implies the client follows each relationship instance as a linked list of objects from the start node to the target node. The drawback of using this technique is that it cannot make use of possible "behind the scene" performance enhancements such as caching or object locality. Another disadvantage is that it requires the same set of operations to be performed over and over again, possibly by the use of some client side defined method. The advantage is that clients are in full control over the navigation process.

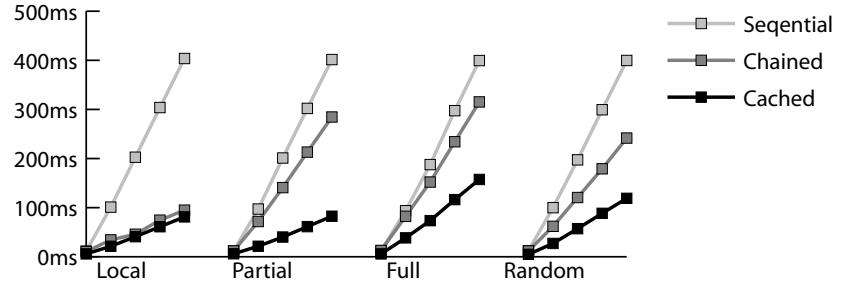
The second technique is the definition of a single-step navigation facility by using *indirect operations* on the objects as outlined in Figure 6.4. The initial purpose of this was to facilitate easy navigation for client applications. Clients issue a single navigation request to a node object and a list of opposite URIs is returned. An additional advantage of this technique is that it delegates the navigation process to the servants and enables the servants to provide for other and more efficient implementations. The tests include results from two different implementations of this pattern that explores respectively the use of the cache and the use of object locality:

- The first implementation is coined *chained* and implies that the node invokes methods on the role which further invokes methods on other objects.
- The second implementation uses the cache on the role and is coined *cached*. This implies that the client invokes a method on the node and that the node invokes a method on the role. The role can then directly return the requested information because it caches information about target roles and nodes.

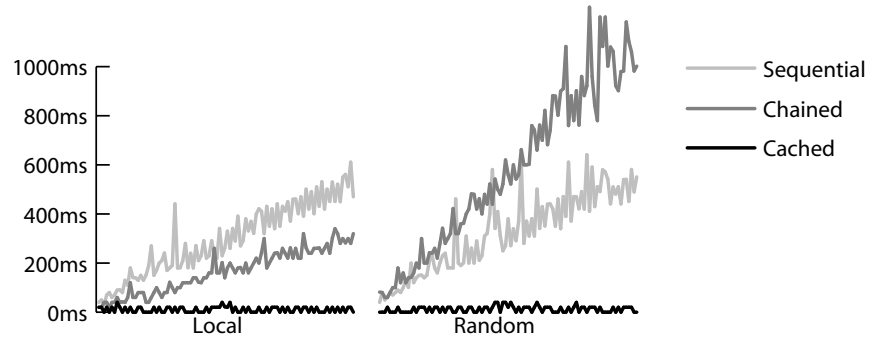
The results of the navigation tests are shown in Figure 8.5. The first test shows the basic performance of navigation over a local area network. The cached navigation technique is the most efficient one, which is quite natural since this implies fewer RMIs than other techniques. The chained and the sequential navigation patterns have equal performance. In this test, the network latency is insignificant compared to the internal processing of methods and the marshalling involved.



Test 1: Average navigation time for the test setup



Test 2: Average navigation with increasing latency: 0,2 - 80ms



Test 3: Navigation performance as the number of opposite nodes increases from 1 - 100.

Figure 8.5: Testing the performance of navigation.

The second test shows the cost of these navigation techniques as the latency between distributed servants increase. The performance of the cached navigation pattern is still by far the most efficient. For local and partial distribution the cost of navigation is equal to the increased latency. An interesting observation is that the use of the chained navigation technique in the local setup is more or less equal in performance to the cached approach, and otherwise performs better than the sequential navigation technique. This is caused by the ORB's efficient implementation of local calls between same process objects. The chained navigation technique can take advantage of this when objects are local with respect to each other. In the full distribution setup the cost of the chained navigation technique will be higher than the increased latency (x2), because of the remote node-to-role call.

The third test shows the scalability of navigation in terms of an increasing number of relationship instances that a single node participates in. This test is performed without added latency because the general performance decrease introduced by latency will follow the same pattern as the second test. The graphs show that the cached method is by far the most scalable one for both setups. The sequential and chained navigation patterns do not scale well due to the increase of RMIs that occurs when the number of relationship instances increases. An interesting observation is that the chained navigation pattern performs better than the sequential in the local setup, but behaves significantly worse in the random setup. The chained navigation pattern can take advantage of the efficient inter-object calls between same process objects that the ORB supports in the local setup. On the other hand, the possible gain of using chained method invocations is contradicted in the random setup mainly due to the increased marshalling that occurs. Data need to be marshalled and demarshalled for each intermediary object it passes through. This is not significant for small chunks of data, but is a cost that increases when the size of the data increases. Only the results from the local and random setup are included in the diagram. The partial and full configurations show the same pattern, but not as apparent as for the random setup.

8.3.3 Creating Relationships

The task of creating a relationship instance includes the creation of nodes, the creation of roles and the creation of a link, and the underlying binding mechanism that creates two-way references. The bind pattern for creating a relationship instance is illustrated in Figure 6.6, and graphs from testing the performance and scalability of the create methods are shown in Figure 8.6.

The basic performance of the create methods is shown in the first test. The difference in performance between the factories is caused by the bind pattern. The node factory does not have to invoke any bind method, the role factory has to invoke a single bind method on a node, whereas the link factory has to invoke the

bind method twice on the two role objects of a binary relationship. An additional is the persistency mechanism. Role and node factories store complex data whereas the node factory only stores a single tuple. The basic test shows that there are no significant differences between the different setups. One should actually assume that the local configuration would perform better due to the ORB being able to call bind methods to same process objects in a more efficient way. The possible gain of this is probably contradicted by the fact that the same process has to serve all operations.

The second test shows the performance of the create methods when latency increases. The main criterion for performance is the bind method; whether it is called on a same process object or is called on a remote object. The effect of increased call latency is most significant for the link factory because of the two bind operation that this factory has to invoke whenever an object is created. Calling a bind method on a same process object is not affected by increased call latency and the local setup is for this reason the most efficient one.

The third test is used to illustrate the scalability of different uniqueness testing methods. The first alternative is to implement a uniqueness testing procedure on the link factory that does a direct search in the available relationship network. This alternative is actually acceptable for the local setup because of the efficient calls between same process objects. In a distributed relationship network, however, this solution does not scale well. The second alternative is to delegate uniqueness testing to the role objects. Each role stores information in the cache about opposite roles, and this can be used to efficiently detect whether a bind request would cause a duplication of an already existing link. The benefit of this method, however, depends on the existence of the cache. Without the cache the role would not be able to perform this more efficiently than the link factory.

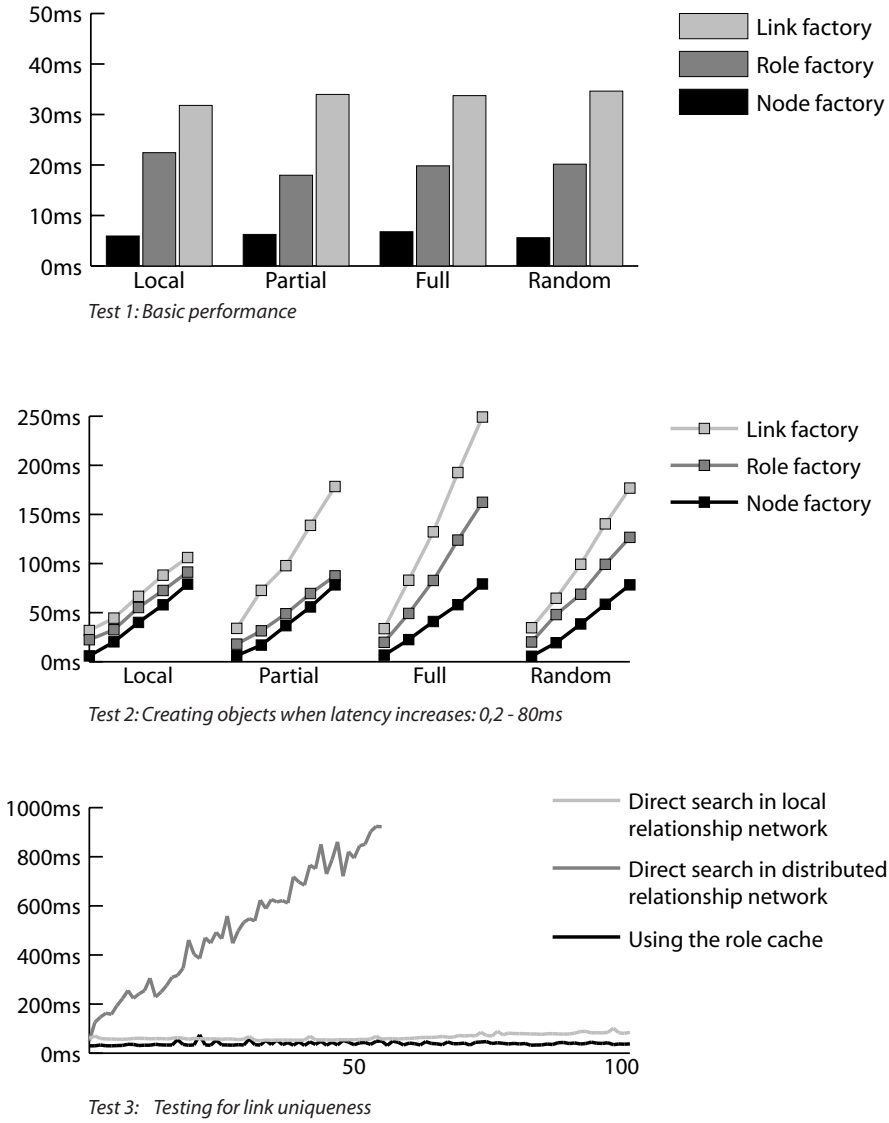


Figure 8.6: Testing the performance of creating relationships.

9 The FRBR Application

9.1 Introduction

Information about books, journals, and other knowledge carrying items, is today contained in numerous bibliographic catalogues that together make an impressive record of the intellectual and artistic endeavours of mankind. Despite the comprehensive bibliographic universe contained in these catalogues, the main focus of the catalogue is the identification and description of “atomic” publications. As such they represent a quite limited view on the bibliographic universe. The need for a more complex model of bibliographic entities and the relationships that exist among these entities is recognized in e.g. [125, 196, 209]. The definition of the entity-relationship based model commonly referred to as the *FRBR model* [100] is a major contribution recently provided by the *IFLA Study group on the Functional Requirements for Bibliographic Records*. The FRBR model identifies the set of entities and relationships of main concern to users of bibliographic information.

The FRBR model, however, is only an initial step towards the "next generation" of bibliographic systems. The next issue is the validity of the model. The model is at present mainly a theoretical solution, and further research and experience is required to validate the model in terms of user benefits, costs, consequences for current cataloguing practice, etc. Another issue is how to apply the model to already existing catalogues that aggregate decades of cataloguing work. Altering the millions of records that already have been created is a risky affair, and sometimes the costs of conversion will be unmanageable for many libraries.

One possible approach to these problems is to implement support for the FRBR model in such a way that it does not alter already existing catalogues. The DL-LinkService is a highly relevant candidate for this by providing support for the FRBR model as a network of nodes and relationships that is detached from the catalogue.

By implementing the FRBR model in a separate level external to the catalogue, support for the FRBR model can be achieved with several potential benefits:

- *Multidirectional relationships.* Relationships in the DL-LinkService are inherently multi-way, which means that if an expression is recognized as the translation of another the inverse relationship will be available too.
- *Consistent relationships.* The DL-LinkService provides for a consistent relationship network, which prevents redundant duplication of information and conflicting participation.
- *Generic support for navigation.* The various relationships of the FRBR model can be accessed in a generic way, which means that it should be easy to implement applications that deal with the complexity of the FRBR model.
- *Support for distribution.* Relationships can be implemented between catalogues. Works and expressions may be described in a national bibliography served by one system, and the manifestation and item entities can be described in the catalogues of specific libraries.

The DL-LinkService would additionally enable experience with the FRBR model to be obtained without intervening with the existing catalogues. Further research and experience with the FRBR model can be obtained by using the DL-LinkService as a test-bed environment to address issues like:

- *Alternative implementations of the FRBR model.* How can the model support user interfaces that provide for information discovery along the paths laid out by the FRBR model? What entities and relationships are important to end users, and what entities and relationships are less frequently used?
- *Abstractions of the model.* Is the model too complex for end users? Are end users able to deal with concepts like expressions and works? If so, what generalizations or abstractions can be made to the model to improve its usability?
- *Alternative models.* Can alternative models or a simplified version of the FRBR model provide for the same functionality with less complexity?

9.2 Current Bibliographic Catalogues

Catalogues have for centuries been the main tool for librarians to create order in the holdings of the library and in the bibliographic universe in general. A catalogue can reflect the holdings of a specific library, or it can be a national bibliography that aggregates information about all publications produced within a country or in a specific language. Other catalogues intend to collect information about all the publications of a single person or publications on a specific subject. While library catalogues reflect the holdings of the library, other biographic catalogues rather reflect what exists; regardless of whether it is available or where it is available from.

The primary intention of a catalogue is to enable persons to find what they are looking for by author, title or subject. The bibliographic record contains the data that is used to create the indexes of search systems, and it contains the information that is presented to the user when he or she uses the catalogue to find relevant books, journals and other items of interest. Catalogues have also provided a surrogate for navigating among the materials in a library's collection, or the entire bibliographic universe, by indicating relationships among the various materials [197]. Major problems, however, are that the relationship information is primarily intended to be readable by humans; it is fragmented across various records and entries within records, and can be both inconsistent and incomplete.

As a "data structure", the bibliographic catalogue can be interpreted as a set of bibliographic records. The bibliographic record aggregates descriptive data elements such as those defined in the International Standard Bibliographic Descriptions (ISBD) [103, 104]. The creation of a bibliographic record is usually referred to as cataloguing, and the cataloguing principles and rules of the library society is a remarkable pioneer in the area of standardized information. This is motivated by an emphasis on reuse and exchange of bibliographic information across system and organizational boundaries. Bibliographic catalogues based on the International Bibliographic Book Description standards and the Anglo American Cataloguing Rules [72] are often captured and exchanged by the use of the MARC format [108, 130]. Although there are numerous "dialects" of the MARC format in use, they are usually crafted on the same basis and are interoperable with each other – to a certain extent. The various data entries of bibliographic catalogues encoded in the MARC format are identified by the use of fields and subfields. Field identifiers are three-numbered values, whereas subfields are identified by a single letter. Some Norwegian example records are found in Figure 9.1.

Today, numerous bibliographic catalogues are maintained by a comprehensive number of institutions all over the world. These catalogues are often made available to the public as Web Based Public Access Catalogues (WEBPACs). A general observation is that these WEBPACs often are similar and usually include a search dialogue, a result set browsing mechanism and a record display facility. WEBPACs are mainly based on the set-based search and retrieve paradigm. Users are required to formulate a query and inspect the result set to discover possibly relevant information. Navigation is sometimes additionally made available, but in most cases this is limited to looking up other records by the same author, subject, etc. The uniform architecture of these systems is mainly caused by the common understanding of catalogues and searching in catalogues that exists within the library community.

```
*001961563842
*008                                    nor
*080c $a839.6
*082uv$a839.82s
*100  $aIbsen, Henrik
*245  $aEt Dukkehjem$bSkuespil i tre Akter$caf Henrik Ibsen$wDukkehjem
*260  $aKøbenhavn$bGyldendalske Boghandels Forlag$c1879
*300  $a180 s.
*096ga$aNBO$cIbsen/si$n80ga25604
*096ga$aNBO$cIbsen/si$n80ga25605

*001000476137
*008                                    eng
*015  $anf0006687
*020  $a0-553-21280-x$bh.
*082ga$d839.822[S]
*100  $aIbsen, Henrik
*245  $aFour great plays$cby Henrik Ibsen ; translated by R. Farquharson Sharp ;
with an introduction and prefaces to each play by John Gassner
*250  $aBantam classic ed.
*260  $aNew York$bBantam Books$c1981
*300  $aXIV, 306 s.
*440  $aBantam classic
*500  $aDenne oversettelsen, 1. utg. 1958
*500  $aInnhold: A doll's house ; Ghosts ; An enemy of the people ; The wild
duck. Originaltitler: Et dukkehjem ; Gengangere ; En folkefiende ; Vildanden
*740  $a4 great plays
*740  $aA doll's house$wdoll's house
*740  $aGhosts
*740  $aAn enemy of the people$wenemy of the people
*740  $aThe wild duck$wwild duck
*740  $aEt dukkehjem$wdukkehjem
*740  $aGengangere
*740  $aEn folkefiende$wfolkefiende
*740  $aVildanden
*096ga$aNBO$cIbsensenteret$n00ga04179

*001990244415
*008                        g           eng
*087uh$aVF Do$bEngland
*245  $aA doll's house$cproduced and directed by Joseph Losey$hvideogram$wA
doll's house
*260  $aU.K., Frankrike$bWorld film services$c1973 (videodistribusjon 1998)
*300  $a1 kassett (VHS) (NTSC) (106 min)$blyd, kol.
*500  $aRolleliste: Jane Fonda, David Warner
*500  $aFrom the play by Henrik Ibsen
*700  $aLosey, Joseph
*700  $aWarner, David
*700  $aFonda, Jane
*700  $aIbsen, Henrik
*096uh$aHIL$cVF Do$n99uh00417
```

Figure 9.1: Example MARC records.

9.3 The FRBR Model

The main objective of the FRBR model as stated in [100] is to provide a:

> *...to provide a clearly defined, structured framework for relating the data that are recorded in bibliographic records to the needs of the users of those records.*

To achieve this, the FRBR captures the entities, attributes and relationships needed to support the generic tasks that are performed when searching and making use of national bibliographies and library catalogues.

The core of the model is a group of entities representing the products of artistic or intellectual endeavour; the *work*, *expression*, *manifestation* and *item* entities. This part of the model is entitled *Group 1 Entities* and is illustrated in Figure 9.2. Another group of entities represent those responsible for the content, and a third group contains entities that serve as the subject of the works. The further focus in this thesis will be on the *Group 1 entities*; *work*, *expression, manifestation* and *item.*

- The *work* is an abstract entity representing a distinct intellectual or artistic endeavour.
- The *expression* entity is the specific intellectual or artistic form a work takes when it is realized.
- A *manifestation* entity is a physical embodiment of an expression.
- The *item* is a single exemplar of the manifestation.

When we refer to the play by Ibsen entitled "A Dolls House", in a generic sense without considering a specific translation, edition or performance, we are dealing with the play as a conceptual *work* entity. This work can be realized in a various intellectual or artistic shapes – as *expression* entities. The original Norwegian text is one realization of this work; the English translation by Henrietta Frances Lord from the 1880s is another realization. The latter English translation was published by different publishers in England and in America, and these publications should be considered different *manifestations* of this particular realization of the work. A specific copy of the American edition, available in the shelves of a library, is an *item* entity.

In addition to the entities, and the attributes that identify and characterize the entities, the FRBR model emphasizes bibliographic relationships [100]:

> *In the context of the model, relationships serve as the vehicle for depicting the link between one entity and another, and thus as the means of assisting the user to "navigate" the universe that is represented in a bibliography, catalogue, or bibliographic database.*

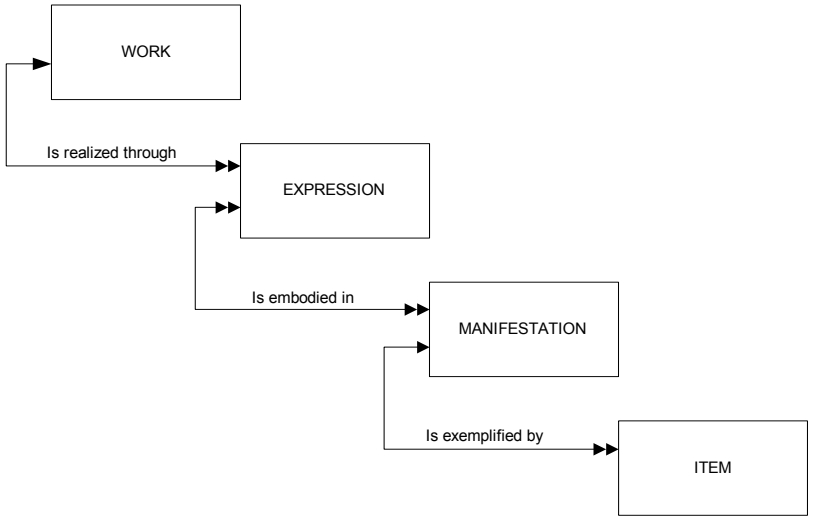


Figure 9.2: The Group 1 entities of the FRBR model.

The work, expression, manifestation and item entities are associated to each other by a set of possible relationships. A work *is realized through* one or more expressions. An expression *is embodied in* one or more manifestations and a manifestation *is exemplified by* one or more items. In addition to these basic relationships, the FRBR-model also defines relationships that may exist between the various entities, orthogonal or parallel to the Group 1 relationships. One expression can be the *translation* of another expression, or the relationship between expressions may be that one is the *adaptation* of the other. Examples of the numerous possible relationships that may exist between works, between expressions, and between a work and an expression, are illustrated in Figure 9.3.

9.4 Enhancing Existing Catalogues

To some extent the current bibliographic record captures aspects of the FRBR model, and mapping between the MARC fields and the FRBR model is provided by Delsey in [55]. Experiences in the extraction of entities and relationships are reported in [91, 95], and these studies show that this is a possible task, although there are considerable problems on certain aspects of this mapping.

When comparing a MARC-based catalogue with the FRBR model, we generally find a 1:1 association between the record and the manifestation entity of the FRBR

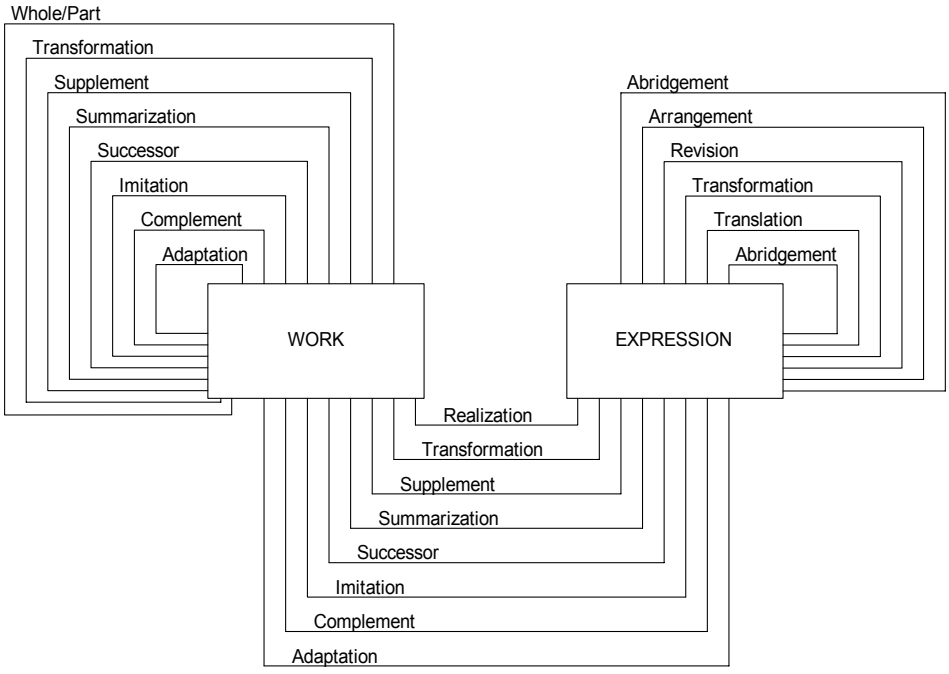


Figure 9.3: Example relationships from the FRBR model.

model. In a specific catalogue there will be one record for each manifestation, and each manifestation will be described by one record. This does not imply that manifestations are the only entities present in the catalogue, but rather that the descriptions of other entities are distributed in a different way. Multiple item entities can be listed in one record, and information related to the expression and work entities can be found in multiple records. One example is the MARC field "uniform title" – which is comparable to the work title of the FRBR model. The same work title can be found in many records if there are many publications containing this work, but the work is not present as a distinct and identifiable entity in the catalogue.

Since the catalogue already contains aspects of the FRBR model, it is theoretically possible to process the data in the catalogue and automatically extract some of the entities and relationships of the model. The resulting information can be characterized as an “index” over the entities contained within the bibliographic catalogue as illustrated in Figure 9.4. This index reflects both the entities of the

FRBR mode – works, expressions, manifestations and items – as well as the extracted relationships that exist among the entities.

The interpretation of a record as a manifestation surrogate enables a convenient alignment between manifestation entities and the catalogue. The manifestation corresponds to one identifiable record in the catalogue, and the record corresponds to one identifiable manifestation entity. A record in the catalogue is thus well suited to serve as a representation of the manifestation entity.

The other entities – work, expression and item – do not, however, have a distinct counterpart within the catalogue structure. Items and the relationship between a manifestation and the item are somewhat easy to deal with because of the 1:N relationship that exists between a relationship and the item. For each item there will be only one source of information, and the item may be represented by defining a limited view on this parent manifestation record. Works and expressions are, on the other hand, conceptual entities at a higher level than the manifestation, and additionally, the information about works and expressions can be fragmented across multiple records. This is further complicated by the fact that a manifestation can embody multiple expressions and correspondingly multiple works. The FRBR model defines the attributes that can be used to describe these entities, but selecting this information from the record can be difficult. This problem is particularly difficult for the expression, as this entity is the least evident entity in existing catalogues.

The actual requirements for the representation of entities will be different depending on the actual use of these entities in the user interfaces of WEBPACs and as processable units in the underlying information system(s).

- *Entities can be represented by distinct records* that describe these entities. Work and expression entities can be maintained in authority files like the ones used for personal and corporate names. The FRBR model defines the attributes that can be used for this purpose, but since the model is not implemented such records are unfortunately not available yet. A similar solution can be used for item entities by maintaining a specific item record for each item in the holdings of the library.
- *Surrogates can be used to represent the entities.* Work and expression entities are abstract entities, but certain manifestations may serve as representations for these. Candidates can be the earliest published manifestation of a work as the work representation. This first edition in the original language of a text is quite close to what we may interpret as the “real” work. Correspondingly the manifestation of the first edition of an expression can serve as the expression representation.

- *Other solutions:* User interfaces that only require records for the display of manifestations can choose other solutions. Work and expressions can be represented in a graphical user interface using generic icons or other unlabelled graphical symbols. Information about items is often available in the catalogue record and may not need further representation.

A different issue is the task of enhancing current catalogues with the FRBR model. In general this can be achieved in two different ways:

- Dynamically enhance the display of records with the FRBR model at run time.
- The creation of a persistent "FRBR index" that is updated on a temporary basis or in "real time".

The FRBR model promotes a view on bibliographic entities that implies a hierarchical view on bibliographic entities (the Group 1 entities). This interpretation of the model can be used to organize the items of a result set into a hierarchical tree-structure. This can be useful when users browse the result sets they are returned from a search [155]. Such a hierarchy can be generated dynamically in a search session by grouping manifestations based on the expression they embody and by grouping expressions by the work they realize. This solution can be further enhanced by the use of runtime routines that detects and presents other relationships between the entities. However, the dynamic grouping and discovery of entities and relationships is an extensive task. To be able to reflect all relevant entities and relationships – not only those that are available in the result set – the system needs to examine the whole catalogue for every search request. The sheer runtime cost caused by the complexity of the FRBR model implies that this solution would be highly inefficient for catalogues other than the ones which have a rather small number of records.

A far more realistic solution is to consider the generation of an updateable "FRBR index" that contains the entities and relationships of the FRBR model and is updated whenever new records are added to the catalogue. Such an index is visualized in Figure 9.4. The index can be implemented as a network of nodes and relationships comparable to the organization of thesauruses and other graph-like structures. The mapping between the index and the catalogue can be solved by aligning the manifestation entities with the records they correspond to, e.g. by implementing a function that resolves between record identifiers and manifestation identifiers. The index can be integrated with the catalogue in many ways at the WEBPAC tier to provide for a FRBR "enabled" view of the catalogue. The FRBR index would additionally be a highly reusable resource that could be used for other purposes as well; such as a tool in the creating of new records.

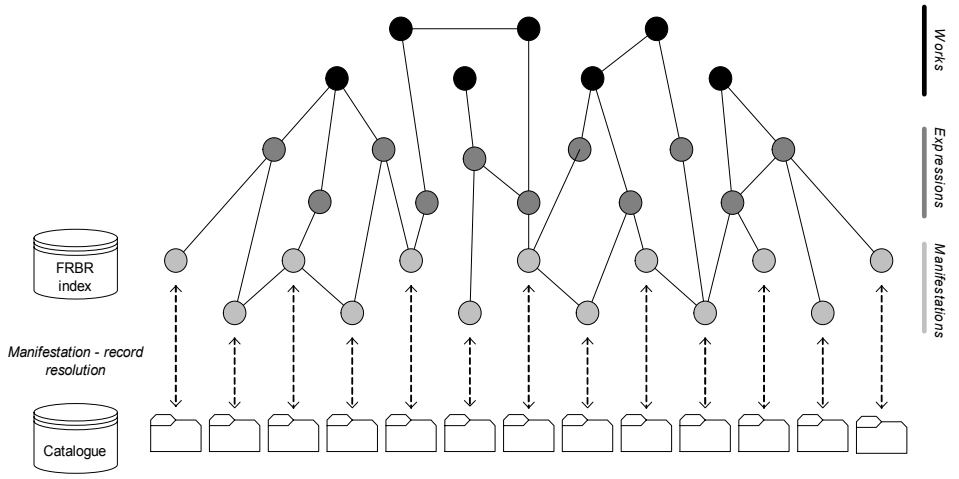


Figure 9.4: The FRBR index.

The DL-LinkService is highly relevant for the implementation of such a "FRBR index". The nodes can be used to represent the entities of the FRBR model, and the explicit relationships of the DL-LinkService can be used to represent the relationships of the FRBR model. The implementation of an application based on this solution is described in the following sections.

9.5 Implementing the Index

The FRBR-index model is implemented in a fully functional prototype system based on a test case that consists of a subset of records from the BIBSYS database. BIBSYS is the Norwegian bibliographic database for university and college libraries and contains approximately 3.5 million records and reflects the holdings of the partaking libraries – approximately 9 million items.

The index was constructed by extracting and processing a set of bibliographic records from this database. A test case consisting of the works by the Norwegian playwright "Henrik Ibsen" was selected because it was likely to contain a rich set of FRBR entities and relationships.

The resulting index is stored and managed by the use of the DL-LinkService. At runtime the index is dynamically integrated with the BIBSYS web interface by using a plugin developed for Internet Explorer. The plugin implements a standard TreeView control in order to visualize the relationships and entities of the map and

enable users to browse the BIBSYS catalogue both by navigating the FRBR relationships as well as using the functionality that already is available through the BIBSYS web pages.

When a user views a record from the BIBSYS system, a complementary window in the Internet Explorer browser shows the existing relationships and the entities participating in these relationships that are relevant for the record in focus. The index can be navigated by selecting any of the available entities, causing the focus to shift to the selected entity. If a manifestation is selected, the corresponding record is retrieved from the BIBSYS system and displayed in the main window of the browser.

The DL-LinkService is fully capable of representing the entities and the relationships of the FRBR model. The relationship structure can easily be navigated by calling methods on the various objects, and the relationships are bidirectional. In essence this means that from a node object all the relationships and opposite nodes are accessible.

Once the general solution was settled, the actual development of the prototype application was a straight forward task, but it included several problems that needed to be solved:

- The definition of a typology that captures the entities and relationships of the FRBR model.
- The identification and extraction of FRBR entities and relationships from the BIBSYS catalogue.
- Choosing a naming convention or resolution mechanism to translate between the record identifiers of the catalogue and the node identifiers of the DL-LinkService.
- Determine how to present entities and relationships in the graphical interface.

9.5.1 Defining a Link Typology for the FRBR-model

For the purpose of this prototype we translated the entities and relationships defined in the FRBR model into the format used by the DL-LinkService. Work, expression and manifestation entities are defined as different node types in order to be able to provide for different client side behaviour for the different entity types. Item entities are not considered in the application, because information about items are already present as a part of the bibliographic record, and the support for entities in the index would not contribute any additional functionality.

The relationship types described in the FRBR model is listed in Figure 9.5. The actual typology includes constraint definitions and is implemented in XML

according to the XMLSchema that is specified for the DL-LinkService. The implemented typology is listed in Appendix A.4.

9.5.2 Extracting Entities and Relationships

The extraction of the FRBR entities and relationships turned out to be the main challenge in the development of this application. Some fields/subfields in the MARC format can be mapped directly to the FRBR model, such as 240$a and 241$a which correspond to the title of the work entity. In such cases the identification of the FRBR entities and relationships is a question of straight forward extraction of data. Other fields in the MARC format only indicate FRBR entities or relationships. In such cases the identification of entities and relationships can be both complicated and unreliable. The solution used in this application is partly guided by [35, 36] and based on inspecting the test case records to uncover alternative ways to solve problems that were caused mainly by records of low quality – like records that were missing the mandatory language code, contained misspelled work titles, etc.

The BIBSYS database uses the BIBSYS-MARC format which is based on the standard Norwegian MARC format – NORMARC – with the exception of some historically related deviations. The database also contains some records converted from other systems, and the records are for that reason of varying quality.

The test case was based on records having "Ibsen, Henrik" as the main entry – personal name. The size of the test case was 2535 records, and the test records were extracted from the BIBSYS catalogue in the MARC format and inserted into a relational database. This enabled a more advanced query facility than what was supported in the native database of the catalogue. The processing of the records turned out to be a rather complex task. The final solution was based on the set of subtasks that are illustrated in the UML usecase diagram of Figure 9.6. A major problem in the creation of the FRBR index was that the information about the entities and relationships needed to be built in a series of sequential steps. In addition to these main tasks, a series of less evident problems needed to be solved in order to deal with problems such as the same kind of information having different fields as the source in different records. Although it is theoretically possible to extract many different works, expressions, and relationships from a MARC record, the real world is somewhat different. The test records contained information rich records well conforming to the MARC format, but many of the records contained sparse information (few fields), and sometimes the information was inconsistent, like the lack of language code or misspelled work titles.

Information regarding the BIBSYS-MARC fields of main concern for the extraction of entities and relationships is listed in Figure 9.7. The column on the right shows in how many test-case records a specific field occurs. Other fields in the

Node	Role	Link	Role	Node
Work	*Has adaptation*	*Adaptation*	*Is an adaptation of*	**Expression**
Work	*Has a complement*	*Complement*	*Complements*	**Expression**
Work	*Has an imitation*	*Imitation*	*Is an imitation of*	**Expression**
Work	*Is realized through*	*Realization*	*Is a realization of*	**Expression**
Work	*Has a successor*	*Successor*	*Is a successor of*	**Expression**
Work	*Has a summary*	*Summarization*	*Is a summary of*	**Expression**
Work	*Has a supplement*	*Supplement*	*Supplements*	**Expression**
Work	*Has a transformation*	*Transformation*	*Is a transformation of*	**Expression**
Work	*Has adaptation*	*Adaptation*	*Is an adaptation of*	**Work**
Work	*Has a complement*	*Complement*	*Complements*	**Work**
Work	*Has an imitation*	*Imitation*	*Is an imitation of*	**Work**
Work	*Has a successor*	*Successor*	*Is a successor of*	**Work**
Work	*Has a summary*	*Summarization*	*Is a summary of*	**Work**
Work	*Has a supplement*	*Supplement*	*Supplements*	**Work**
Work	*Has a transformation*	*Transformation*	*Is a transformation of*	**Work**
Work	*Has part*	*Whole/Part*	*Is part of*	**Work**
Expression	*Has an abridgement*	*Abridgement*	*Is an abridgement of*	**Expression**
Expression	*Has adaptation*	*Adaptation*	*Is an adaptation of*	**Expression**
Expression	*Has an arrangement*	*Arrangement*	*Is an arrangement of*	**Expression**
Expression	*Has a complement*	*Complement*	*Complements*	**Expression**
Expression	*Has an imitation*	*Imitation*	*Is an imitation of*	**Expression**
Expression	*Has a revision*	*Revision*	*Is a revision of*	**Expression**
Expression	*Has a successor*	*Successor*	*Is a successor of*	**Expression**
Expression	*Has a summary*	*Summarization*	*Is a summary of*	**Expression**
Expression	*Has a supplement*	*Supplement*	*Supplements*	**Expression**
Expression	*Has a transformation*	*Transformation*	*Is a transformation of*	**Expression**
Expression	*Has a translation*	*Translation*	*Is a translation of*	**Expression**
Expression	*Has part*	*Whole/part*	*Is part of*	**Expression**
Expression	*Is embodied in*	*Embodiment*	*Embodies*	**Manifestation**
Manifestation	*Is exemplified by*	*Example*	*Is an example of*	**Item**
Manifestation	*Has an alternate*	*Alternate*	*Is an alternate to*	**Manifestation**
Manifestation	*Has a reproduction*	*Reproduction*	*Is a reproduction of*	**Manifestation**
Manifestation	*Has part*	*Whole/Part*	*Is part of*	**Manifestation**
Item	*Has reconfiguration*	*Reconfiguration*	*Is a reconfiguration of*	**Item**
Item	*Has reproduction*	*Reproduction*	*Is a reproduction of*	**Item**
Item	*Has part*	*Whole/Part*	*Is part of*	**Item**
Item	*Has a reproduction*	*Reproduction*	*Is a reproduction of*	**Manifestation**

Figure 9.5: The FRBR typology.

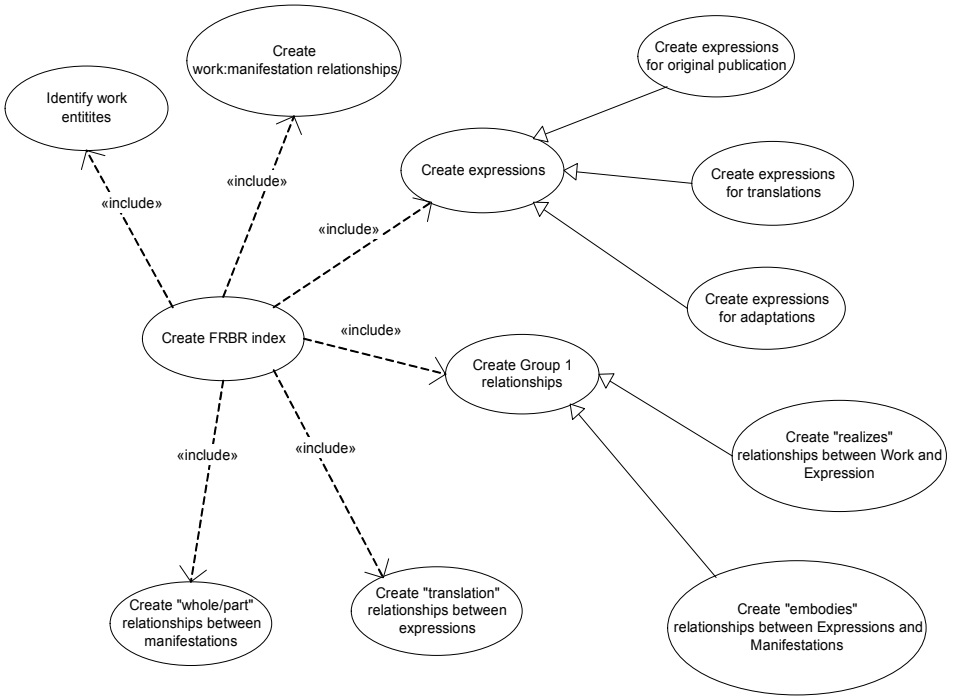


Figure 9.6: Extracting entities and relationships.

MARC format could have been useful as well, but these fields generally had a very low occurrence and for that reason could not be used as a reliable source of information for this application. The following sections summarize the main strategies used to achieve the final index.

9.5.2.1 Identifying work entities

Work entities are in this application mainly identified by the use of the work title. This simplification is made possible by the uniform set of records in the test set – all works are by the author Henrik Ibsen, and it can be assumed that all distinct works by Ibsen have unique names. A more heterogeneous set of records would require a more complex set of work attributes for proper identification; like form of the work, the date of the work, and other distinguishing characteristics.

The most reliable source in a MARC record for the extraction of work titles is he field *240$a – uniform title* and the field *241$a – original title*. These fields occur in approximately half of the records, and other sources of work titles are needed to

Field	Subfield	Data	Count(*)
008	c	General information - language code	2339
020	a	ISBN	448
100	a	Main entry - personal name	2535
240	a	Uniform title	1118
240	l	Uniform title - language	907
241	a	Original title	340
245	a	Title statement - title	2535
245	c	Title statement - responsibility	2295
250	a	Edition statement - edition	367
260	a	Publication - place	2513
260	c	Publication - date	2527
440	a	Series statement	737
500	a	General note	894
700	a	Added entry - personal name	1404
740	a	Added entry - uncontrolled title	498

Figure 9.7: MARC fields and their occurrence in the test set.

come up with a more complete set of work titles for the records. Relevant secondary sources for work titles are the *740$a – uncontrolled title* and *245$a – title*. Due to the occurrence of translated titles and more modern spellings in these fields, only records in the original language of the works can be used for this purpose. 245$a does not contribute any significant new titles other than titles with a more modern spelling and was excluded in the final selection of work titles. The extraction algorithm includes a set of queries that stepwise creates a table of work titles based on the following sequence. The results from this procedure are listed in Figure 9.8:

- Examine all records and select distinct strings from *240$a*
- Examine all records and select distinct strings from *241$a*
- Examine only Norwegian records that do not have values in neither 240$a nor 241$a and select distinct strings titles from *740$a*.

	240$a / 241 $a	740 $a	Total
Records containing this field	1447	296	1743
Unique work titles	41	43	84

Figure 9.8: Extracted works and their origin MARC fields.

9.5.2.2 Identifying expressions

A number of expressions may exist for a specific work, but the identification of a distinct set of expressions can be ambiguous. Expressions can be identified by a range of fields and subfields as described in [55], but a major problem is that the value of these fields only indicates an expression. The proper identification of an expression additionally requires a certain degree of literary analysis, and the required information may be unavailable in the bibliographic record or inconsistently and vaguely described. Rather than attempting to identify the set of expressions directly, the approach chosen for this application is based on the assumption that if there are one or more relationships between a manifestations and a work, then there has to be at least one expression in between them. The set of relevant expressions for a work can be identified by clustering the manifestations according to various information that is available in the records: all translated manifestations in a given language can be grouped into an expression representing the text translated into this language; if the translators name is available this can further be divided into different translations, etc. The expressions that can be identified in the test records are:

- The text in the original language based on the assertion that for each work there is an expression in the original language.
- Adaptations that can be identified by the form of the work either by inspecting the formcode of 008 or by certain keywords in other fields, such as "sound recording" etc.
- Translations of the text in different languages based on the language code in 008. This was by far the most frequently occurring type of expression.

The numbers of expressions found are listed in Figure 9.9 and examples of the expressions that can be identified are shown in Figure 9.10 and Figure 9.11.

Kind of expression	**No. of expressions**
Original expression	122
Translations	452
Adaptations	183

Figure 9.9: Different categories of expressions found.

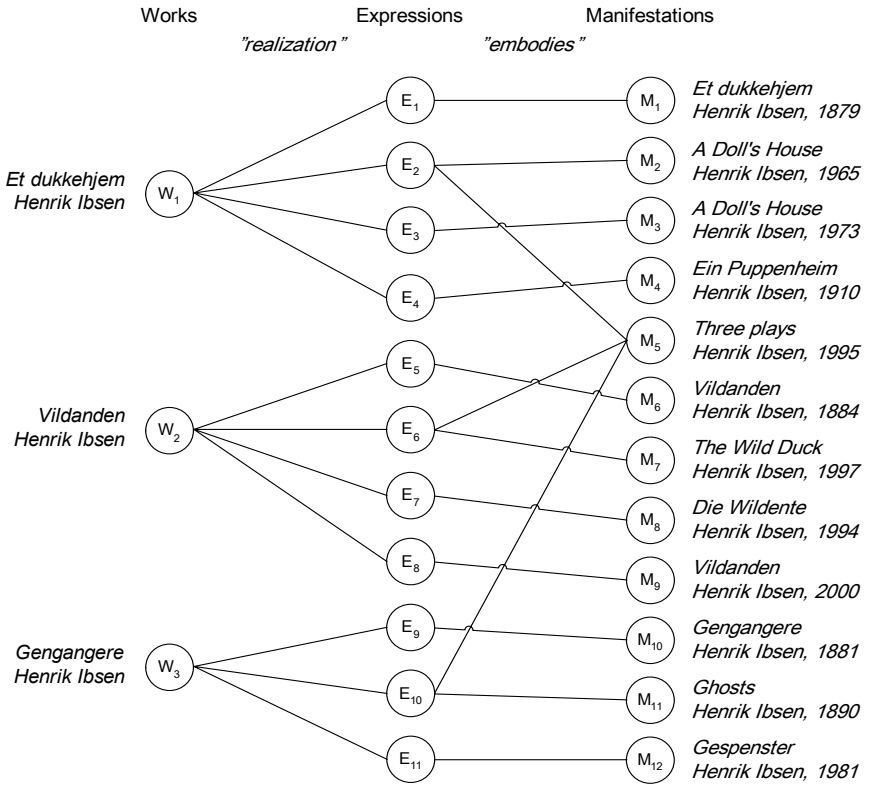


Figure 9.10: Example expressions I.

Expression	Description
E1	Original Norwegian text of "Et dukkehjem"
E2	English translation of "Et dukkehjem"
E3	Video of a performance of "Et dukkehjem"
E4	German translation of "Et dukkehjem"
E5	Original Norwegian text of "Vildanden"
E6	English translation of "Vildanden"
E7	German translation of "Vildanden"
E8	Sound recording from a performance of "Vildanden"
E9	Original Norwegian text of "Gengangere"
E10	English translation of "Gengangere"
E11	German translation of "Gengangere"

Figure 9.11: Example expressions II.

9.5.2.3 Identifying relationships

Once entities are defined it is possible to identify the relationships. Expressions are directly intermediates in the already identified relationships between work and manifestation, and the actual basic Group 1 relationships can directly be derived; the relationship between a work and its expression is of type *realizes*, and the relationship between an expression and the manifestation is of the *embodies* type. Additional expressions identified are either translations or adaptations and are related to the "original text" expression through either translation or adaptation relationships.

An additional kind of relationship that can be identified is the *whole/part* relationship between manifestations. The works of Ibsen are typically published as multi-volume publications, each containing a single play or multiple plays. This is easily captured because the BIBSYS database implements a record structure for this. The "parent" publication is described in a separate record, and each volume is described in separate records that are linked to the parent record.

The various kinds of relationships that are found are listed in Figure 9.12 along with the number of relationship instances that occurred in the test set. Figure 9.13 shows the relationships that exist between the expressions of the previous example. The whole/part relationships are exemplified in Figure 9.14.

From entity	To entity	Relationship type	Number of relationships
Work	Expression	Realization	757
Expression	Manifestation	Embodies	2469
Expression	Expression	Translation	1533
Expression	Expression	Adaptation	183
Manifestation	Manifestation	Whole/Part	453

Figure 9.12: List of identified relationships.

9.5.2.4 Additional Issues

Although the procedure in theory is a sequence of intuitive steps, the actual extraction required some additional problems to be solved. Missing work titles in the records was a major problem for detecting entities and relationships – in particular between translations and their corresponding work entities. This problem

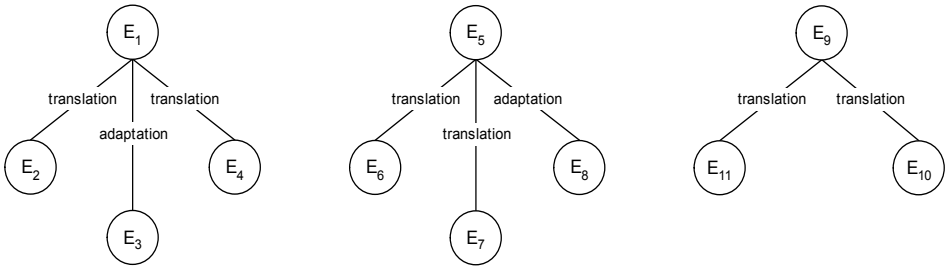


Figure 9.13: Example relationships.

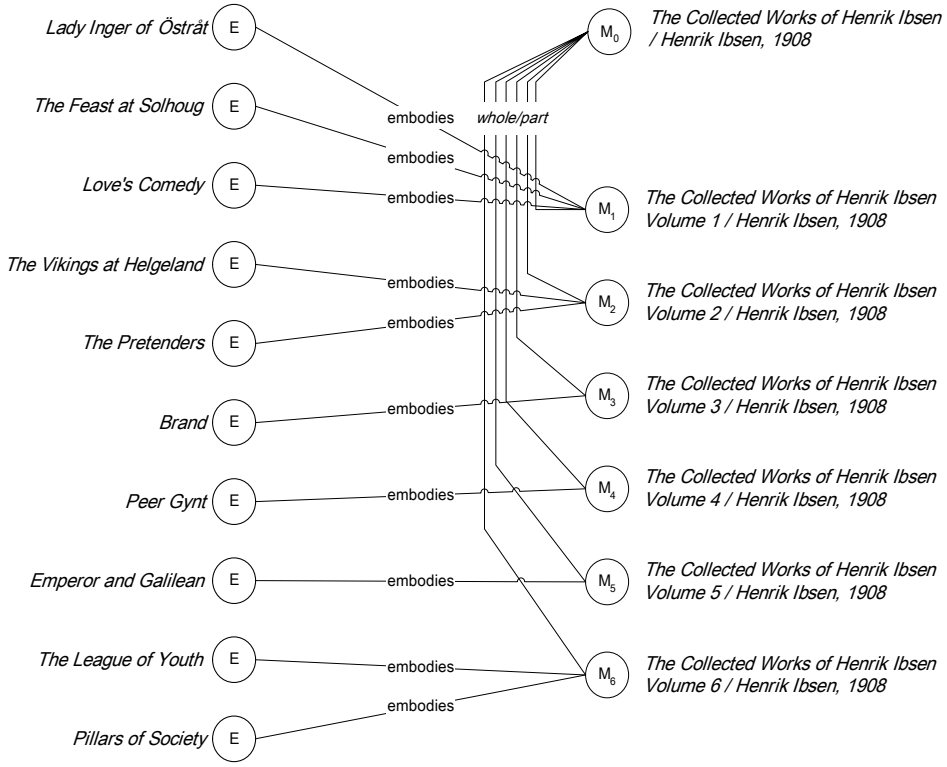


Figure 9.14: Multi-part volumes and the expressions they embody.

was solved by the use of a translation table based on the combinations of known manifestation titles and work titles that existed in other records. If one record contains the English title "A Doll's House" in 245$a and the original title "Et dukkehjem" in 240/241 or 740, this information could be applied to comparable records where the work title was missing in order to associate this manifestation with the correct work.

The identification of the language of a manifestation was another problem caused by a missing language code in the 008 field. The language of a manifestation can, however, be interpreted by looking for language in 240$l or by resorting to the publication place in 260$e which is likely to reflect the language of the manifestation. The number of language code occurrences in Figure 9.7 reflects the total number of language cods after this procedure was performed.

9.5.3 The FRBR Index

The resulting data that was created based on the above procedure contained a rich set of entities and relationships. After identification of the entities and the relationships, a simple batch program was used to create the nodes and relationships in a running instance of the DL-LinkService. Nodes reflecting the work entities were created as nodes of type *Work,* using the title attribute to hold the work title, author name and the term “work” in a compact representation. Unique URI-based identifiers were created for work nodes, but these URIs are merely identifiers and do not point to any specific resource.

Nodes representing the expression were created as nodes of type *Expression*. The node title of the expression was based on the work title combined with the language of the expression in order to indicate a translation, or the adaptation keyword to indicate an adaptation. This simplification was necessary because it was impossible to determine which translated title belonged to which expression. For manifestations that contained multiple works both the translated title and the original title were listed in the same field (e.g. 740), but without any precise mapping from the original title to the translated title. As with work nodes, the URI of expression nodes merely serves to uniquely identify the resource and does not point to any specific resource.

Nodes for manifestations were created for each manifestation in a comparable way, but the manifestation URI points to the actual record for the manifestation in the BIBSYS database.

9.6 A Client for Navigating Relationships

The DL-LinkService can be accessed by any kind of application that is able to communicate with CORBA distributed objects, and the FRBR index can be integrated with catalogues in different ways. As a part of this work, a client application was developed that enables a user to interact with the BIBSYS catalogue by the use of a web browser and the FRBR index. The application is an Internet Explorer plugin that implements a CORBA client for talking to the DL-LinkService. It is tightly integrated with the browser environment as an Explorer bar – using the same window that otherwise is used for the favourite's list, browsing history, etc. The plugin communicates with the main window by sinking the events of the explorer and sending events to the main application to initiate specific actions.

The nodes and relationships of the FRBR index are visualized using a basic TreeView control. Only a subpart of the graph is displayed at one time, using the node in focus as the root and directly connected nodes as leaf nodes. FRBR entities are displayed as squares with a letter inside indicating the kind of node they represent. Relationships are displayed using a diamond shape to indicate that this is a relationship. Relationships are only accompanied by the typename, but *nodes* are labelled with the value of the title attribute. Nodes behave in a slightly different way in the tree view window. In the BIBSYS WEBPAC neither work nor expression entities has a record equivalent, so it is not possible to retrieve records that corresponds to these entities. A double click is used to navigate in the index and retrieve other entities and their relationships, and if the user selects a manifestation node the plugin retrieves the corresponding record from the BIBSYS catalogue.

In addition to interactive retrieval of catalogue records using the TreeView control the client application is notified whenever a new web page is loaded into the browsers main window, for instance if a user has performed a search using the BIBSYS search dialogue and retrieves a specific catalogue record from the result set. The plugin uses the URL of the loaded webpage to look up a directory service and retrieve the node object reference for this particular manifestation. It can then access the DL-LinkService to retrieve the relationships that are available for the manifestation.

A screen shot illustrating the client is shown in Figure 9.15. The record displayed in the main browser window refers to a manifestation that embodies the English translation of "Little Eyolf". The window on the left displays the relationships and the entities available for this expression.

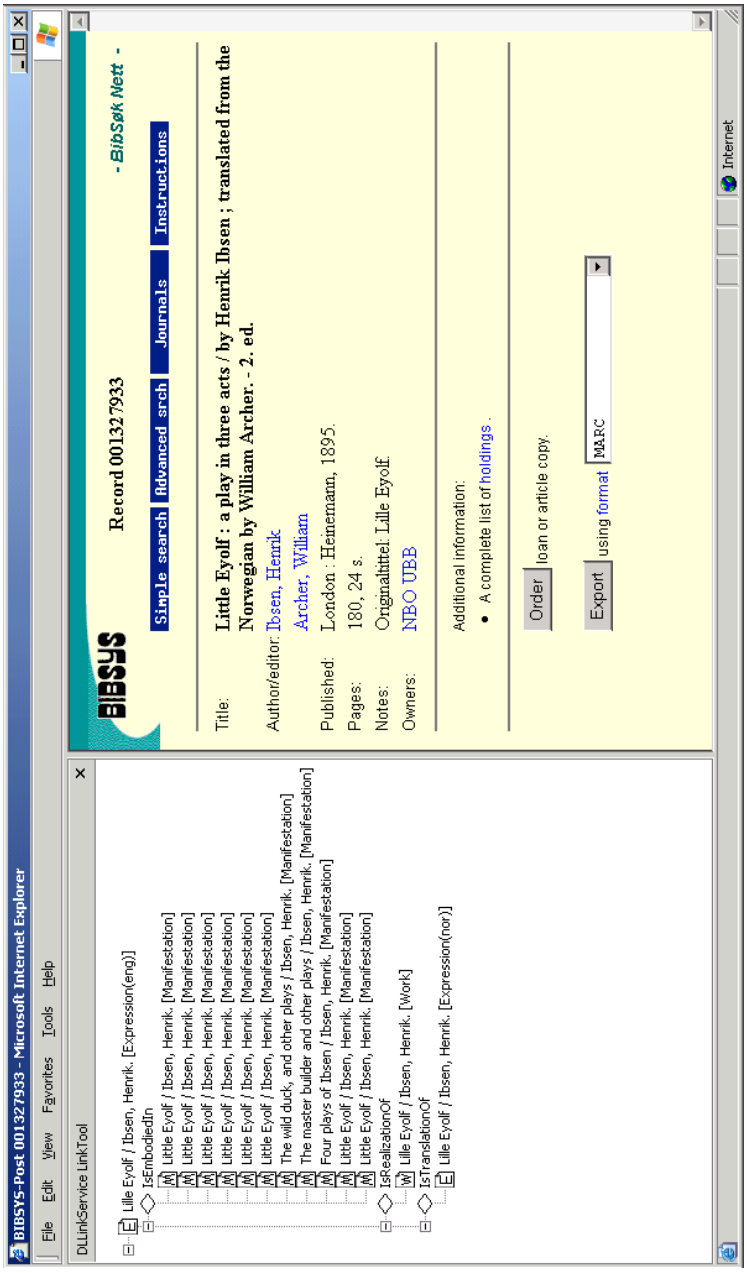


Figure 9.15: Screenshot of the client.

9.7 Evaluating the Application

The main contribution of this application is a framework for adding the FRBR model to current bibliographic catalogues based on the use of an external index. The index is realized by using the DL-LinkService and is an example of the applicability of the service proposed. Other than the definition of the typology and determining how to identify the entities of the FRBR model, the actual use of the DL-LinkService for this purpose was a straight forward implementation issue. The client side implementation related to the use of the DL-LinkService is approximately 40 lines of code, which illustrates the simple and efficient solution for navigation in the DL-LinkService. The behaviour of the TreeView control and the integration with the browser, however, is implemented by a more extensive number of source-code lines.

The main challenge in the development of this application was the extraction of the entities and the relationships according to the FRBR model. This was achieved by using a variety of queries and manipulations on the test case data and demonstrates quite clearly that the task of enhancing bibliographic catalogues with the FRBR model requires an external index, because the dynamic generation of this information at runtime will be too expensive in terms of recurrent processing. A different alternative is to alter the existing records to reflect the same information that the index contains. Such an approach, however, requires that the FRBR model is determined to be the information model for future cataloguing and that the set of expressions and manifestations applicable to existing records are available in authority files.

The focus of this application has been on a specific subset of records describing works by Henrik Ibsen. Parts of the overall procedure that are used to extract the entities and relationships laid out by the FRBR model are general, in the sense that they can be applied to the whole BIBSYS catalogue – and other catalogues as well. Other elements of the procedure are more pragmatic ad hoc solutions that only will be valid for this specific subset of records from this specific catalogue. The identification of expressions for the original text is one example, and was made possible by knowing that Ibsen was Norwegian and asserting that all his works were originally published in Norwegian.

In general we found that the guidelines suggested by Delsey in [55] are valid, but a major problem is the significant difference between the indication of an FRBR entity and the proper identification of an entity. Expressions are, in particular, hard to identify, and as the application shows, one will often have to rely on more general expressions than what the FRBR model suggests. One example is the use of the "English translation" as a common expression for all English translations, despite the existence of different translations by different translators. A more concise and information rich catalogue may, however, yield a better result in terms of identifying

a distinct set of expressions. The implemented application illustrates that the FRBR model may not be as mature as it appears. The complexity of the resulting index is quite apparent and the presentation of this complexity in user interfaces is a challenging issue. One possible simplification that can be applied to the model is to eliminate the expression entity by rather using work and manifestation entities and relationships to capture the information that otherwise would have to be expressed by using expression entities. The use of the DL-LinkService for the purpose of developing and evaluating such variations over the FRBR model is an easy and convenient solution.

The developed client is merely a first attempt to illustrate how the DL-LinkService can be integrated into an application. The networked nodes are presented using the Microsoft tree view control, which is a coarse simplification of the actual structure of the relationship network. The tree only presents a simplified subpart of the underlying graph of the stored index. A better presentation of the index may be achieved by using other graph presentations with correspondingly improved interaction techniques. A different problem is the labelling of nodes and relationships in such a way that users easily can interpret the index. The entities have to be presented using selected metadata elements, and a suitable balance has to be made between the need for a compact, and yet informative presentation.

The test case used in this application does originate from a single bibliographic catalogue, but this is not a limitation. The Digital Library LinkService supports linking of distributed content, and the service itself may also be distributed. An index that implements relationships between distributed records can easily be constructed. The manifestations of the index may include records from different catalogues e.g. with paths to digital facsimiles of the original manuscripts stored elsewhere on the Internet. Work and expression entities may link to authority records e.g. maintained by national libraries.

10 Evaluating the Service

10.1 Application Areas

The solution presented in this work is based on relationships as first-class objects external to the participating entities. A relationship instance is implemented as a compound structure of node, role and link objects that form a relationship network detached from the information object layer. This is implemented as a CORBA distributed object application which supports a high degree of flexibility in terms of distribution. However, requirements for relationships solutions in specific applications may vary, and the proposed solution primarily targets applications where one or more of the following is needed:

- A reusable solution for relationships.
- Explicit relationships that can be accessed by third-party applications.
- Multi-way relationships.
- Dynamic typing of relationships.
- Distributed endpoints.
- Requirements for processing capabilities within the range the service provides.

A typical scenario for using the service is in user-initiated navigation tasks where the user retrieves the complete set (or subset) of relationships for a particular node, inspects the relationships, selects one of the targets and retrieves the content of this target possibly along with the relationships that the target participates in. The usage of the service is, however, not limited to user initiated navigation, but may very well include automized tasks based on the traversal of relationships, such as automatic retrieval of the subparts of a compound information object for integrated presentation. Another scenario is to use the relationship network to propagate operations, such as updating and indexing.

With a subcategory of relationship applications other solutions would be more beneficial. The use of reference based relationship implementation is frequently found in many implementations, mainly because they are simple to implement along with the implementation of the other aspects of an application. A different motivation is that direct addressing mechanism often will be the most efficient solution, because there is no additional overhead when navigating, creating or deleting relationships. Such solutions will typically be used when one or more of the following characteristics are applicable:

- High capacity requirements in terms of processing speed and rate.
- A predefined and delimited number of relationships between well defined classes.
- No requirements for third-party access.
- Sufficient with one-way relationships.
- No requirement for distribution of the relationship information in a cooperative environment.

A typical scenario where the proposed solution will not perform adequately would be a search through the relationship network for all nodes (or the content that nodes reference) that satisfies specific criteria. If the network consists of e.g. a million of interrelated nodes, the mere time cost of searching the entire network will be beyond the time requirements of most applications. The major bottleneck for such applications is the use of remote method invocations to traverse fine-grained information. More efficient solutions for specific tasks can, on the other hand, be implemented by defining interfaces that support the efficient implementation of these tasks on the server side, such as querying a server-side index.

10.2 The Object Model

The object model used in the DL-LinkService is a fined-grained representation of relationships. The basic features of the model can be summarized as:

- *Support for relationships of any degree.* The link object can be used to aggregate any number of role objects and in this way support relationships of arbitrary degree. All examples in the test-case application, however, are binary, which is by far the most common kind of relationship. The extensive use of binary relationships versus the use of relationships of higher degree can simply be due to the lack of support for higher degree relationships or that other solutions often are preferred, such as the use of compound objects or collections. The importance of supporting higher degree relationships can be

discussed, but the DL-LinkService enables high degree of flexibility in the structuring of information spaces within a uniform solution.

- *Support for multi-way relationships.* The model supports multi-way relationship, and does not deal with one-way relationships in a particular way. The use of one-way relationships can, on the one hand, be considered as a specific kind of bi-directional relationship where the inverse direction is invisible from the target. This can be supported by extending the typing system with a direction constraint and extending the implementation with behaviour that leaves the inverse direction inaccessible for clients. The basic model does not directly address this, but the cache implementation of the prototype shows that it is possible to implement reference-based relationships within the same model.
- *Support for relationship typing.* Relationships are typed using a role-based typing scheme that supports the proper interpretation of relationships regardless of what direction the relationship is navigated. Typologies are used to define the allowed typenames, but the typing is detached from the implementation of the service in order to support a flexible use of the system. Deployment of the service only requires the definition of a typology or the reuse of an already defined typology.
- *Support for constraints.* Constraints are defined at the type level, and constraint checking is integrated into the object servants. The constraint mechanism is used to ensure a formally correct relationship network, such as ensuring that associated objects satisfy type constraints and that relationships conform to the cardinality constraints.
- *Support for consistency.* A role aggregates and maintains all semantically equivalent relationship instances that a particular node participates in. This means that the model can be used to prevent link duplication and in this way enable a consistent relationship network. The testing performed on the prototype implementation shows that duplicate detection can be a significant bottleneck, because it involves an unpredictable number of RMIs. However, the caching scheme implemented in the prototype is used to solve this in a way that yields a predictable and scalable solution.

In addition to the basic features described above, the model theoretically supports relationship attributes. The only attributes implemented in the prototype is a predefined set of attributes used to store title and type, but application specific attributes can easily be supported by implementing support for dynamic data types or by the use of the CORBA Property Service. Application specific attributes, in particular if they are based on the use of the CORBA Property Service, will be difficult to cache with the caching scheme that is implemented in the prototype. The

use of application specific attributes may for this reason imply a less efficient access to the relationship network, because it requires that objects are directly accessed rather than through the use of optimized short-cut operations.

The methods defined for the interfaces (and implemented in the prototype) are the basic operations required to navigate relationships either by directly traversing chained objects or by using a server-side operation that allows for optimization techniques like the caching scheme that is implemented in the prototype. The first strategy gives clients full control over the navigation process. On the other hand, due to the possibly large number of RMIs that this may involve, this can be a performance bottleneck as the number or relationship instances grow. The latter strategy, based on the use of a role that caches information about opposite ends, is a more reliable access method that scales well.

The model can further be extended to support other commonly needed methods. The model does not directly support application specific relationship methods defined as part of the interface signature without requiring implementation efforts. On the other hand, this can be achieved by extending the service, such as defining application specific interfaces that inherit the interfaces of the basic object model. Inherited interfaces can still be interoperable with other components, but client-side access to application specific behaviour requires support for these methods either by supporting this subtyped interface or by using the dynamic method invocation facility of CORBA.

10.3 Distribution

The fine-grained object model provided by the DL-LinkService enables a flexible solution for distribution that can be tailored to a range of different needs. The service can be used as a standalone service in a traditional client-server setting, in which case the relationship network can be “owned” by a single enterprise and will not contain any cross-organizational dependencies. On the other hand, the service can be used in a cooperative environment where multiple services are used to build a network where the relationship information is distributed over any number of hosts and enterprises. The degree of distribution is a deployment concern and needs to consider issues such as trust, performance requirements, quality of service, etc. The main contribution of the service is that it can be tailored to most configurations in terms of distribution.

The testing performed on the prototype shows that the use of the fine-grained object model can cause problems in terms of performance and scalability, but the tests conducted also prove that this can be solved if certain techniques are used to improve on the performance. The use of an ORB that supports efficient calls

between objects that reside in the same memory space, combined with the use of server side navigation, is a sufficient solution if the relationship network is served by a single instance of the service. The proposed caching scheme is a significant contribution to improve the performance and scalability of the service if it is deployed as multiple cooperating instances in a distributed environment. This is particularly evident for the task of navigation where this technique reduces the number of RMIs from an unpredictable and possibly high number to a constant number of 2 RMIs. In a multi-service setup, the best solution is to keep endpoints (nodes and their associated roles) collocated with respect to each other.

The creation of objects is on the other hand not very efficiently solved in the current design of the system. Creating relationship instances when there is a high latency in between the objects, is a particular problem. Creating relationship is on the other hand a task that can be solved in many other ways than the basic factory pattern that is defined for the service. If clients need to create a large number of relationship instances efficiently, a more batch-like solution would be appropriate. The task of creating relationships can also benefit from the cache as this can be used to implement a scalable uniqueness testing mechanism.

10.4 Reuse

The proposed service supports reuse with respect to:

- Reuse of software assets.
- Reuse of resources.

The main purpose of reusing software assets is to decrease the development efforts and cost of building applications. A main problem with component deployment is that components need to be integrated into the application and/or with other components in some way. The DL-LinkService can be integrated in digital library systems as a detached service that merely references the information object layer, which is the kind of deployment explored in this work. The main motivation for this solution is to overcome the problem that current digital library systems are based on heterogeneous software architectures, and as such there is no uniform solution for component deployment yet. URI-based referencing is a mechanism that can be used to support integration with digital library systems in many ways. Unfortunately, the use of URIs is not a global solution for all kinds of information addressing. Many digital library systems only implement internal identification schemes for information objects, such as internal record identifiers and file names.

Integration of the service into a digital library system can additionally be based on a tighter coupling between the information object layer and the relationship

layer. This can be achieved by implementing support for the interfaces of the DL-LinkService directly into the digital library system. However, this solution requires a more system-specific implementation of the service or adaptation of an existing code in order to create the coupling between the information object layer and the relationship layer.

Reuse of resources is a different kind of reuse that allows third-party applications to access the functionality and information a specific service provides. The DL-LinkService is based on CORBA distributed objects which implies that all information inherently is accessible for external parties. The degree of accessibility is determined by the implemented security policies. Support for the service at the client side is merely a question of automatically generating the required code by the use of ordinary CORBA tools, and further usage of the service is a client-side issue. Digital library clients can even integrate dynamic support for the service if they support the dynamic interface invocation facilities of CORBA. Whether this actually is a feasible solution remains a question. The use of a complex service that includes several interrelated interfaces without prior knowledge of the overall behaviour is highly difficult to support.

A different aspect of reuse for digital library services is the capability they have to integrate with common services such as security, naming and directory services, transactions, etc. The service proposed in this work uses CORBA and can easily be integrate available CORBA services.

10.5 Referential Integrity

The DL-LinkService is based on the use of persistent objects and relies on the use of object references to accommodate the integrity of the system, which is not without certain problems. Applications that are based on the use of distributed objects require referential integrity – a requirement that they share with many other applications. A system of CORBA objects and their object references has referential integrity if there are no dangling references (references without objects) and there are no orphaned objects (objects that cannot be contacted via a reference) [92]. Dangling object references are for example analogous to the dangling links that (too) frequently occur on the Web – links that point to documents that do not exist any more.

Referential integrity in CORBA is a potential problem, because object references are permitted to propagate by uncontrollable means. Object references can be freely passed around as stringified references, and they can be copied in a number of ways. The object reference is merely a communication identifier which contains information about where the object can be found in the network. The reference itself

is completely detached from the object it points to, which means that removal or migration of the object will not be reflected in the reference.

Digital libraries store information with a long-term persistence in mind, and it is natural to assume that the information/knowledge that extends or builds on the primary content is created with the same long-term persistence in mind. If a relationship is created between two long-term persistent documents by the use of the DL-LinkService, it is natural to assume that this relationship potentially should be available for the same period of time. Whereas information tends to be persistent, the hardware and software environment that is used to store and disseminate this information is more likely to change over time. In the long term this will unavoidably result in random failures and unpredictable changes that can compromise referential integrity. This is further complicated if the relationship network spans multiple independently managed hosts.

Unfortunately there is no universal solution that can be applied to prevent lack of referential integrity. One way to deal with the lack of referential integrity is to live without it and to have fallback behaviours to recover from the problem, such as reacquiring an object reference when it fails as described in [93]. CORBA supports forwarding of object references, and the use of the CORBA naming service targets the use of logical names that can be resolved to actual references at runtime. Other strategies include the garbage collection of references that refer to permanently unavailable (deleted) objects. A similar problem with referential integrity is often found in hypermedia applications as well [47], and the proposed solutions are similar [101].

The current DL-LinkService prototype does not explicitly implement any fallback behaviour to recover from referential integrity problems. However, the system can easily be extended to support such behaviour in different ways:

- Objects are uniquely identified by their UUIDs. The actual references to objects are based on ordinary CORBA object references, but clients may additionally store the UUID of the target object. In case of dangling references, the UUID can be used to reacquire a reference from a naming or directory service.
- A different problem is caused by permanently unavailable objects. The DL-LinkService supports consistent deletion of relationships, but errors may occur in a distributed environment if objects are removed without properly deleting the relationships first. In order to recover from this it is required to perform garbage collection of object references. In the DL-LinkService prototype the caching mechanism is used to duplicate opposite endpoints, and this information can be used to perform proper garbage collection of relationship endpoints and in this way re-establish referential integrity.

The problem of maintaining referential integrity may seem as an apparent problem with the DL-LinkService, because it relies on a fine grained object model with an extensive use of object references. This problem, however, is mostly appearing when relationship instances are distributed over multiple independently deployed instances of the DL-LinkService. Systems that are maintained by a single enterprise usually have means to ensure the integrity of the system. The degree of this problem will increase as the number of component instances deployed by independent actors increase, and additionally increase with deployments of unpredictable reliability. A service deployed by a responsible organization/company is naturally more reliable than a service deployed by a private user on a private machine. Correspondingly, the problem can be decreased by carefully considering the deployment of cooperating instances. The DL-LinkService provides distribution in a highly flexible way, but how the service is deployed to provide the needed balance between flexibility and ensured level of referential integrity is a security policy issue.

The problem of maintaining long-term referential integrity is a problem that is apparent in many other distributed digital library applications as well. The referential integrity problem of URL-based identification is generally acknowledged, and has led to persistent identifier solutions like the Handle System, PURL [181], and the URN initiative. Many current technologies include the use of URIs to identify resources at a level of granularity comparable to the DL-LinkService, such as the use of URIs to identify the subject and object of RDF statements, the use of URIs to identify types, ontology entries, etc. Although the use of persistent identifiers for such purposes appears to be a promising solution, it is merely a partial solution that provides for convenient maintenance of references, global uniqueness, etc. These solutions, however, are not any more reliable than the directory that contains the mapping between the persistent identifier and the actual address. In this context, the possible referential integrity problem of the DL-LinkService should be considered a general problem that will occur as federated digital libraries with fine grained interdependency at the information level emerge.

10.6 Interoperability

The proposed service and the prototype that is explored in this thesis encompass both the data storage level and the processing level of the service. The prototype implementation stores information in a database using an ad-hoc database schema. Although interoperability with specific formats that support relationship information is described in Chapter 7.4, the implemented prototype does not attempt to implement the use of a specific format at the bottom layer of the model depicted in Figure 2.4.

The ideal solution for a digital library relationship service is a service that can be used as a front end to relationship information that is locally stored in different formats, such as Dublin Core metadata, XLink, Topic Maps, RDF, etc. Such formats are, unfortunately, highly diverse, and a direct mapping between the relationship model of the DL-LinkService and other formats has not been further explored in this work. The implementation of such interoperability is a task that needs to be explored for each of the relevant formats. In most cases this needs to be solved on a per application basis. Furthermore, this is complicated by the possible distribution of relationships information that the DL-LinkService allows for.

11 Conclusions and Future Work

11.1 Conclusions

The motivation for this work is based on two generally recognized research issues for digital libraries. One is the need for interlinked and semantically rich information spaces in digital libraries. Relationship information is considered to be an important element when developing such information spaces. The other is the service-oriented architecture envisioned for future digital library systems, which implies that the overall functionality of digital libraries will consist of smaller independent services that can be reused in a federated and cooperative digital library environment. These two orthogonal research issues lead up to the specific problem statement explored in this research which is:

> *How to provide a service for using and managing relationships in digital libraries.*

This question is answered by exploring digital libraries with respect to the content they store and disseminate, which at the general level can be interpreted as information objects. Relationships among information objects are in current digital library solutions based on a highly heterogeneous set of solutions. The approach that is explored in this work is the use of an explicit and generic data structure for relationship information and a service that implements the support for managing and using such information.

This approach is explored by first examining relationship knowledge in general and then examining specific conventions and solutions that are used for explicit relationship information in technologies related to digital libraries. Based on this understanding, an abstract but formal model of explicit relationships is developed that captures the variable structural and semantic aspects of relationships within one uniform construct.

A specific solution – the Digital Library Link Service – is proposed which is designed as a CORBA distributed object application. The service extends and combines features from relevant existing solutions. It combines the detached solution and instance-oriented approach to linking that is found in hypermedia link services with the support for consistency and constraints that is inspired by object-oriented solutions for explicit relationship objects. The object model of the CORBA Relationship Service is used as the initial model, because it captures well the generic model of relationship information that this thesis proposes. The result is a coherent solution for managing consistent, multi-way relationships in a distributed digital library environment. The use of a generic relationship construct in combination with a flexible implementation-independent typing mechanism provides a generic and reusable solution that can be applied in a range of different digital library applications.

A prototype implementation that uses caching and fat operations is implemented, and test results show that the proposed service is a feasible solution that yields predictable performance in terms of response time and scalability.

The proposed service is a generic solution for using and managing relationships that easily can be adapted to various applications by defining application-specific typologies. Validation that it can be used to solve specific problems is provided by the developed test-case application that implements the relationships of the FRBR model as navigational paths on top of an existing bibliographic catalogue based on the use of the DL-LinkService.

11.2 Summary of Contributions

- The thesis contributes to the understanding of relationships in digital libraries by reviewing how relationships generally are understood and supported in information technology. A general conclusion is that existing solutions for representing and processing relationship information are highly diverse.
- An abstract model of explicit relationships is specified that provides a flexible solution for representing relationship information in a uniform way. This shows that a generic data model can be used to capture the full range of relationship types that are relevant for digital libraries. This solution, however, requires a relationship typing scheme to ensure logically correct relationships.
- The thesis specifies and describes the Digital Library Link Service – an instance-oriented solution for managing and using relationships in digital libraries. The use of an object-oriented solution enables a coherent and flexible solution for the data, behaviour and constraints of explicit relationships.

By separating the typing of relationships from the implementation, a reusable and dynamic service for relationship support in digital libraries is achieved. The DL-LinkService can be used to create complex relationship networks.

- This work shows that the use of a fine-grained relationship model implemented as distributed objects enables distribution of the relationship network while still being able to support constraints and maintain consistency. The cost of this, however, is a certain complexity that can reduce performance and scalability due to the call latency of network communication. A prototype is developed that utilizes caching in order to solve this. Tests conducted show that this technique significantly contributes to the scalability and efficiency. This is particularly important when the relationship information is distributed across different processes with high call-latency in between.
- The fine-grained relationship object model that is deployed enables distribution along many axes. The service can be used in a client/server fashion or it can be used for peer-to-peer collaborative construction of a consistent, coherent and highly distributed relationship networks. A general conclusion is that relationship information can be distributed if this is needed. Implementing long-term persistent information as distributed objects, however, is a challenging issue due to the possible lack of referential integrity that may occur. This work suggests the use of globally unique identifiers as the main enabler for implementing solutions to solve and prevent such problems.
- The proposed solution supports interactive traversal of the relationship network. However, the main access paradigm explored in this work is user-initiated navigation with a relaxed requirement for processing capacity. The need for automatic and efficient processing of a large relationship network, e.g. for the purpose of indexing, can be supported by extending the system with additional functionality.
- Furthermore, a prototype application for enhancing bibliographic catalogues with a rich set of relationship types is implemented. This demonstrates the applicability of the service as a flexible tool for associative organization of information spaces and illustrates the potentially rich information structures that relationships can enable in digital libraries.

11.3 Related Work

The solution proposed in this thesis is in many ways related to the works of others. The various aspects of the proposed service are inspired by different solutions and

can be interpreted as a novel combination of features that otherwise are fragmented across various technologies.

Current relationship support in digital libraries is to a certain degree focused on formats for structured information, such as particular metadata formats and RDF, Topic Maps, XLink, and various document formats that support links. The formats that are most comparable to the model proposed in this work are the XLink specification and the Topic Maps specification. The major difference between this work and the various descriptive solutions is a focus on both the processing and representation of relationship information in a coherent solution. The use of object-orientation enables the embedded support for features like consistency mechanisms and constraint checking. The development of various processing environments for Topic Maps, RDF, and XLink, is an emerging solution, but this is mainly focused on the development of APIs for processing single repositories of such information.

The intention of the proposed typing scheme used in the DL-LinkService is comparable to the use of types and ontologies in Semantic Web applications. The implemented typology scheme is based on a specific XML Schema, but a comparable scheme can for example be based on the DAML+OIL language [114] to achieve interoperability with the typing schemes of other digital library services. The advantages of the proposed typing scheme are that it is easily readable and easy to generate manually.

Different solutions for relationships in object-oriented solutions are described in [21, 24, 57, 36, 182, 179]. Comparable solutions for digital library systems can be found in [70, 207]. This work proposes a solution that is comparable to solutions that implement relationships as first-class objects. The main difference is that this work does not consider relationships as a static feature at the class level, but rather as a dynamic feature of instances. One object-oriented database solution with a comparable dynamic approach is described in [167]. The ability to allow any object to participate in any relationship without having this feature fixed at compile time is considered to be important for digital libraries due to the heterogeneous nature of digital library information objects and the unpredictable requirements for relationship participation.

An additional difference is that the DL-LinkService directly exposes the relationship information to third-party applications as distributed objects, which seldom is the case for relationship constructs in object-oriented languages and databases.

The DL-LinkService is based on the same object model as the CORBA Relationship Service, and some of the features of the service are for that reason comparable. The main difference is the use of the same interfaces and factories for all types of relationships, which in the CORBA Relationship Service needs to be specified as a distinct interface for each relationship type. The dynamic run-time

typing of the DL-LinkService yields a quite different solution that can be deployed without programming level adaptation. The use of fat operations extends a feature that is used in the CORBA Relationship Service, whereas the caching scheme is specific for the implementation of the DL-LinkService prototype and is an important solution that improves the performance and scalability of the service.

Hypermedia link services have been a major source of inspiration in the development of the solution proposed in this work. The links of hypermedia are two-ways, they can be added or removed dynamically without interfering with the information objects and certain hypermedia link services have a comparable approach to dynamic typing. However, several aspects of the DL-LinkService are different from hypermedia link services:

- The DL-LinkService supports relationships of arbitrary degree, whereas many hypermedia link services are mainly concerned with binary links, such as [87, 214].
- The support for constraints (and consistency) mechanisms are less apparent in hypermedia.
- Hypermedia systems usually use simple labels when typing links, although other more complex semantic structures have been explored, such as in [143]. The use of node-, role- and link typenames in the DL-LinkService is a more powerful mechanism for expressing the semantics of a relationship structure.
- Hypermedia link servers are in most cases based on the client-server model. Clients may retrieve links from multiple link repositories, but there has been little research on integration of link repositories. This is supported significantly differently in the DL-LinkService, because it can be used to build an integrated relationship network that is distributed across multiple instances of the service.
- The DL-LinkService addresses the need for relationships in many kinds of applications. A hypermedia application can be based on the use of the DL-LinkService, but this is merely one potential use of the service among others. This feature is more comparable to the structure services of structural computing [152, 153].

11.4 Limitations and Further Work

The following issues have been identified as problems that should be addressed in future work and extensions that can be considered in future development of the service:

- Further development of the service to include support for features like transactions, security and properties, as described in Chapter 7.
- A more advanced typology system e.g. based on the ontology languages that are used for the Semantic Web. Features that can be included are the use of type inheritance, the ability to import and reuse type definitions across typologies, more advanced constraints such as node-node constraints, etc.
- Supporting one-way references and multi-way relationships within the same framework is another possible extension that can be added to the proposed solution. The service emphasizes support for typed and constrained multi-way relationships which can be interpreted as the most complex kind of relationships. However, certain applications simply do not need such an advanced solution. The use of simple references can be adequate for many purposes and can even be an advantage because it does not require any kind of synchronization. Support for one-way and reference-based relationships can be considered as a special case of the proposed model, and the system can be extended to support the abstraction of a role as a one-way reference.
- The object model of the proposed service is based on strong coupling between the various objects, which can be a possible problem if the service is deployed as a highly distributed application, e.g. if numerous service instances are used to create a cooperative relationship network. However, the dependencies between various instances of the service can be reduced by using an event-based mechanism rather than method invocations in the implementation of the bind/unbind pattern.
- A different field that needs to be explored is the support of fallback behaviour to referential integrity problems, such as reacquiring references through the use of directories or naming services, or the use of garbage collection of references to permanently unavailable objects etc. This is not only related to the service proposed in this thesis but applies to the use of CORBA as the infrastructure for distributed and interdependent long-term persistent information in general. The use of location independent names as the primary addressing mechanism, combined with a high performance resolution technique, is an important research issue related to many digital library applications.
- The main application areas that the service addresses are tasks with a relaxed requirement for efficiency in terms of processing capacity. Retrieval of all

relationships for a particular node is quite efficient and only affected by the network latency of the client/server connection when a node and its associated roles are served by the same host/process. This means that the service should perform equally as well as e.g. hypermedia link services. Automatic processing of larger network structures, on the other hand, may exceed a reasonable response time, because this implies multiple consecutive invocations, such as when processing all relationships for a large set of nodes. However, a solution that supports high capacity processing of relationships can be supported by extending the service with interfaces and server side implementations for specific tasks, such as querying an index for nodes that satisfy particular criteria.

- Finally, further validation through implementation and experimentation is required in order to properly eliminate unforeseen problems and improve on the model. The test-case application presented in this work is only one out of a range of applications that the service should be able to support. The main design criterion of the service is that it should be adaptable to a variety of applications by defining application specific typologies. In addition to the test-case explored in this work, applications relevant for testing the service could be:
  - A traditional hypertext application where the nodes and relationships are used to represent fragments of interlinked text or other hypermedia objects and the dynamic discovery and integration of such links into the content at runtime.
  - An annotation service where users can create and store annotations and personal relationship structures, and possibly annotate existing relationships by the use of the CORBA Property Service. The challenge in such applications would be to support relationship networks that are associated to a specific person either by associating a context with the relationship objects or by using multiple node sets for the same information where the nodes are associated to particular users.
  - Citation linking is another important application area in digital libraries. The challenges of such applications would be to interoperate with other initiatives that address the same problem like the repository independent addressing scheme of OpenURL [53].
  - Thesaurus browsing integrated with the document repository where the thesaurus entries and the relationships between these are represented as nodes and relationships, as well as the representation of the document to thesaurus entries by the use of the DL-LinkService.

– Explore the use of the service to represent compound information objects and collections. Implementing a sorted collection can be achieved e.g. by defining a relationship type of undefined degree and the use of relationship properties to contain the data value that is used to sort the collection.

Appendix

A.1 The LinkService Module

```
module LinkService {

    typedef octet UUID[16];
    typedef string URI;
    typedef sequence<URI> URIS;

    interface UUIDObject {
        readonly attribute UUID uuid;
    };

    // Forward declarations
    interface Node;
    interface Role;
    interface Link;

    typedef sequence<Node> Nodes;
    typedef sequence<Role> Roles;
    typedef sequence<Link> Links;

    struct NodeHandle {
        Node the_node;
        Typology::Type the_node_type;
        string the_node_uri;
        string the_node_title;
        UUID the_node_uuid;
    };
```

```
struct RoleHandle {
    Role the_role;
    Typology:Type the_role_type;
    string the_role_title;
    UUID the_role_uuid;
    NodeHandle the_node_handle;
};

struct LinkHandle {
    Link the_link;
    Typology::Type the_link_type;
    string the_link_title;
    UUID the_link_uuid;
};

typedef sequence<LinkHandle> LinkHandles;
typedef sequence<RoleHandle> RoleHandles;
typedef sequence<NodeHandle> NodeHandles;

interface NodeFactory {
    exception InvalidURI{string uri;};
    exception NodeNotCreated{string reason;};
    Node create (in string uri, in Typology::Type node_type,
                 in string title)
                 raises (InvalidURI, NodeNotCreated);
};

interface Node : UUIDObject{
    exception UnKnownRole {};
    exception UnKnownRoleType {};
    exception DuplicateRoleType {};
    exception RoleSetNotEmpty {};
    readonly attribute RoleHandles role_handles;
    readonly attribute string title;
    readonly attribute Type node_type;
    readonly attribute string uri;
    RoleHandle get_role_handle (in Typology::Type role_type);
    Nodes opposite_nodes(in Typology::Type role_type)
                 raises (UnKnownRoleType);
    URIS opposite_uris(in Typology::Type role_type)
                 raises (UnKnownRoleType);
    void destroy()raises (RoleSetNotEmpty);
    NodeHandle bind (in RoleHandle a_role_handle)
              raises (DuplicateRoleType);
    void unbind (in RoleHandle a_role_handle);
};
```

```
    interface RoleFactory {
        exception NilRelatedNode {};
        exception UnknownRoleType {};
        exception NodeError {string reason;};
        exception RoleNotCreated{string reason;};
        RoleHandle create (in Node entity_node,
                           in Typology::Type role_type,
                           in string title)
                           raises (NilRelatedNode, NodeError,
                           UnknownRoleType, RoleNotCreated);
    };

    interface Role : UUIDObject{
        exception RoleError {string explanation;};
        exception UniquenessError {string explanation;};
        readonly attribute Node entity_node;
        readonly attribute string title;
        readonly attribute Typology::Type role_type;
        readonly attribute unsigned long current_cardinality;
        readonly attribute LinkHandles link_handles;
        Nodes opposite_nodes();
        URIS opposite_uris();
        void destroy() raises(RoleError);
        void bind (in LinkHandle link, in RoleHandles other_roles)
                   raises(RoleError, UniquenessError);
        void unbind (in LinkHandle link)
                     raises (RoleError);
    };

    interface LinkFactory {
        Link create (in RoleHandles role_handles,
                     in Typology::Type link_type,
                     in string title)
                     raises (TypeError, Role::UniquenessError);
    };

    interface Link : UUIDObject{
        exception DestroyError {string explanation;};
        readonly attribute string title;
        readonly attribute Typology::Type link_type;
        readonly attribute RoleHandles role_handles;
        RoleHandles other_role_handles(in RoleHandle role_handle);
        void destroy () raises(DestroyError);
    };

};
```

A.2 The Typology Module

```
module Typology {

    // Typologies are identified by simple strings
    typedef string TypologyName;

    // A data structure for compound typenames
    struct Type{
        string typology_name;
        string type_name;
    };

    // The declaration of various types
    typedef sequence<TypologyName> TypologyNames;
    typedef sequence<Type> LinkTypes;
    typedef sequence<Type> RoleTypes;
    typedef sequence<Type> NodeTypes;

    // A data structure for link type definitions
    struct LinkTypeDefinition{
        Type link_type;
        string degree;
        RoleTypes allowed_roles;
        boolean rolename_uniqueness;
        boolean link_uniqueness;
    };

    // A data structure for role type definitions
    struct RoleTypeDefinition{
        Type role_type;
        string min_cardinality;
        string max_cardinality;
        LinkTypes allowed_links;
    };

    // A data structure for node type definitions
    struct NodeTypeDefinition{
        Type node_type;
        RoleTypes allowed_roles;
    };
```

```
    interface TypeLookUp{
        exception UnknownType {};
        exception UnknownTypology {};
        LinkTypeDefinition get_link_typedefinition
            (in Type link_type) raises (UnknownType);
        RoleTypeDefinition get_role_typedefinition
            (in Type role_type) raises (UnknownType);
        TypologyNames get_typology_names()
            raises (UnknownTypology);
        LinkTypes get_link_types (in string typology_name)
            raises (UnknownTypology);
        RoleTypes get_role_types (in string typology_name)
            raises (UnknownTypology);
    };
};
```

A.3 The Typology Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schemaxmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">
   <xs:element name="Typologies">
      <xs:complexType>
         <xs:sequence maxOccurs="unbounded">
            <xs:element ref="Typology"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="Typology">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TypologyName" type="xs:string"/>
            <xs:element name="NodeType" type="NodeTypeDefinition"
                        maxOccurs="unbounded"/>
            <xs:element    name="RoleType" type="RoleTypeDefinition"
                        maxOccurs="unbounded"/>
            <xs:element name="LinkType" type="LinkTypeDefinition"
                        maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>

      <xs:unique name="NodeUniqueness">
         <xs:selector xpath="NodeType"/>
         <xs:field xpath="TypeName"/>
      </xs:unique>

      <xs:unique name="RoleUniqueness">
         <xs:selector xpath="RoleType"/>
         <xs:field xpath="TypeName"/>
      </xs:unique>

      <xs:unique name="LinkUniqueness">
         <xs:selector xpath="LinkType"/>
         <xs:field xpath="TypeName"/>
      </xs:unique>

      <xs:key name="LinkKey">
         <xs:selector xpath="LinkType"/>
         <xs:field xpath="TypeName"/>
      </xs:key>
```

```
    <xs:key name="RoleKey">
      <xs:selector xpath="RoleType"/>
      <xs:field xpath="TypeName"/>
    </xs:key>

    <xs:keyref name="AllowedRolesInNodekeyRef" refer="RoleKey">
      <xs:selector xpath="NodeType/AllowedRole"/>
      <xs:field xpath="."/>
    </xs:keyref>

    <xs:keyref name="AllowedRolesInLinkKeyRef" refer="RoleKey">
      <xs:selector xpath="LinkType/AllowedRole"/>
      <xs:field xpath="."/>
    </xs:keyref>

    <xs:keyref name="AllowedLinkInRolesKeyRef" refer="LinkKey">
      <xs:selector xpath="RoleType/AllowedLink"/>
      <xs:field xpath="."/>
    </xs:keyref>
  </xs:element>

  <xs:complexType name="NodeTypeDefinition">
    <xs:sequence>
      <xs:element name="TypeName" type="xs:string"/>
      <xs:element name="AllowedRole" type="xs:string"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="LinkTypeDefinition">
    <xs:sequence>
      <xs:element name="TypeName" type="xs:string"/>
      <xs:element name="Degree" type="xs:byte"/>
      <xs:element name="AllowedRole" type="xs:string"
                  maxOccurs="unbounded"/>
      <xs:element name="RoleNameUniqueness" type="xs:boolean"/>
      <xs:element name="LinkUniqueness" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
```

```
   <xs:complexType name="RoleTypeDefinition">
      <xs:sequence>
         <xs:element name="TypeName" type="xs:string"/>
         <xs:element name="MinCardinality" type="CardinalityType"/>
         <xs:element name="MaxCardinality" type="CardinalityType"/>
         <xs:element name="AllowedLink" type="xs:string"
                     maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>

   <xs:simpleType name="CardinalityType">
      <xs:restriction base="xs:string">
         <xs:pattern value="[0-9]*|N"/>
      </xs:restriction>
   </xs:simpleType>

</xs:schema>
```

A.4 The FRBR Typology

```
<?xml version="1.0" encoding="UTF-8"?>
<Typologies xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Typologies.xsd">

    <Typology>

        <TypologyName>FRBR</TypologyName>

        <NodeType>
            <TypeName>Work</TypeName>
            <AllowedRole>HasAbridgement</AllowedRole>
            <AllowedRole>Complements</AllowedRole>
            <AllowedRole>HasComplement</AllowedRole>
            <AllowedRole>HasSuccessor</AllowedRole>
            <AllowedRole>HasSummary</AllowedRole>
            <AllowedRole>HasSupplement</AllowedRole>
            <AllowedRole>HasTransformation</AllowedRole>
            <AllowedRole>HasAdaptation</AllowedRole>
            <AllowedRole>HasImitation</AllowedRole>
            <AllowedRole>HasPart</AllowedRole>
            <AllowedRole>IsSuccessorOf</AllowedRole>
            <AllowedRole>IsSummaryOf</AllowedRole>
            <AllowedRole>IsTransformationOf</AllowedRole>
            <AllowedRole>IsAdaptationOf</AllowedRole>
            <AllowedRole>IsImitationOf</AllowedRole>
            <AllowedRole>IsPartOf</AllowedRole>
            <AllowedRole>IsRealizedThrough</AllowedRole>
            <AllowedRole>Supplements</AllowedRole>
        </NodeType>

        <NodeType>
            <TypeName>Expression</TypeName>
            <AllowedRole>Complements</AllowedRole>
            <AllowedRole>HasComplement</AllowedRole>
            <AllowedRole>HasRevision</AllowedRole>
            <AllowedRole>HasSuccessor</AllowedRole>
            <AllowedRole>HasSummary</AllowedRole>
            <AllowedRole>HasSupplement</AllowedRole>
            <AllowedRole>HasTransformation</AllowedRole>
            <AllowedRole>HasTranslation</AllowedRole>
            <AllowedRole>HasAdaptation</AllowedRole>
            <AllowedRole>HasAbridgement</AllowedRole>
            <AllowedRole>HasArrangement</AllowedRole>
            <AllowedRole>HasImitation</AllowedRole>
```

```
    <AllowedRole>HasPart</AllowedRole>
    <AllowedRole>IsRealizationOf</AllowedRole>
    <AllowedRole>IsRevisionOf</AllowedRole>
    <AllowedRole>IsSuccessorOf</AllowedRole>
    <AllowedRole>IsSummaryOf</AllowedRole>
    <AllowedRole>IsTransformationOf</AllowedRole>
    <AllowedRole>IsTranslationOf</AllowedRole>
    <AllowedRole>IsAbridgementOf</AllowedRole>
    <AllowedRole>IsAdaptationOf</AllowedRole>
    <AllowedRole>IsArrangementOf</AllowedRole>
    <AllowedRole>IsImitationOf</AllowedRole>
    <AllowedRole>IsEmbodiedIn</AllowedRole>
    <AllowedRole>IsPartOf</AllowedRole>
    <AllowedRole>Supplements</AllowedRole>
</NodeType>

<NodeType>
    <TypeName>Manifestation</TypeName>
    <AllowedRole>HasReproduction</AllowedRole>
    <AllowedRole>HasAlternate</AllowedRole>
    <AllowedRole>HasPart</AllowedRole>
    <AllowedRole>IsReproductionOf</AllowedRole>
    <AllowedRole>IsAlternateTo</AllowedRole>
    <AllowedRole>Embodies</AllowedRole>
    <AllowedRole>IsExemplifiedBy</AllowedRole>
    <AllowedRole>IsPartOf</AllowedRole>
</NodeType>

<NodeType>
    <TypeName>Item</TypeName>
    <AllowedRole>HasReproduction</AllowedRole>
    <AllowedRole>HasPart</AllowedRole>
    <AllowedRole>HasReconfiguration</AllowedRole>
    <AllowedRole>HasReproduction</AllowedRole>
    <AllowedRole>IsReconfigurationOf</AllowedRole>
    <AllowedRole>IsReproductionOf</AllowedRole>
    <AllowedRole>IsExampleOf</AllowedRole>
    <AllowedRole>IsPartOf</AllowedRole>
</NodeType>

<RoleType>
    <TypeName>HasAbridgement</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Abridgement</AllowedLink>
</RoleType>
```

```
<RoleType>
    <TypeName>IsAbridgementOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Abridgement</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasAdaptation</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Adaptation</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsAdaptationOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Adaptation</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasAlternate</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Alternate</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsAlternateTo</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Alternate</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasArrangement</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Arrangement</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsArrangementOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Arrangement</AllowedLink>
</RoleType>
```

```
<RoleType>
    <TypeName>HasComplement</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Complement</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>Complements</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Complement</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasImitation</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Imitation</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsImitationOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Imitation</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasPart</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>WholePart</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsPartOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>WholePart</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasReconfiguration</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Reconfiguration</AllowedLink>
</RoleType>
```

```
<RoleType>
    <TypeName>IsReconfigurationOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Reconfiguration</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasReproduction</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Reproduction</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsReproductionOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Reproduction</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasRevision</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Revision</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsRevisionOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Revision</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasSuccessor</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Successor</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsSuccessorOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Successor</AllowedLink>
</RoleType>
```

```
<RoleType>
    <TypeName>HasSummary</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Summarization</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsSummaryOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Summarization</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasSupplement</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Supplement</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>Supplements</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Supplement</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasTransformation</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Transformation</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsTransformationOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Transformation</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>HasTranslation</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Translation</AllowedLink>
</RoleType>
```

```
<RoleType>
    <TypeName>IsTranslationOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Translation</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsEmbodiedIn</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Embodiment</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>Embodies</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Embodiment</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsExemplifiedBy</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Example</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsExampleOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>1</MaxCardinality>
    <AllowedLink>Example</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsRealizedThrough</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>N</MaxCardinality>
    <AllowedLink>Realization</AllowedLink>
</RoleType>

<RoleType>
    <TypeName>IsRealizationOf</TypeName>
    <MinCardinality>0</MinCardinality>
    <MaxCardinality>1</MaxCardinality>
    <AllowedLink>Realization</AllowedLink>
</RoleType>
```

```
<LinkType>
    <TypeName>Abridgement</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasAbridgement</AllowedRole>
    <AllowedRole>IsAbridgementOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>Adaptation</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasAdaptation</AllowedRole>
    <AllowedRole>IsAdaptationOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>Alternate</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasAlternate</AllowedRole>
    <AllowedRole>IsAlternateTo</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>Arrangement</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasArrangement</AllowedRole>
    <AllowedRole>IsArrangementOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>Complement</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasComplement</AllowedRole>
    <AllowedRole>Complements</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>
```

```
<LinkType>
    <TypeName>Imitation</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasImitation</AllowedRole>
    <AllowedRole>IsImitationOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>WholePart</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasPart</AllowedRole>
    <AllowedRole>IsPartOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>Reconfiguration</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasReconfiguration</AllowedRole>
    <AllowedRole>IsReconfigurationOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>Reproduction</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasReproduction</AllowedRole>
    <AllowedRole>IsReproductionOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>Revision</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasRevision</AllowedRole>
    <AllowedRole>IsRevisionOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>
```

```
<LinkType>
    <TypeName>Successor</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasSuccessor</AllowedRole>
    <AllowedRole>IsSuccessorOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>Summarization</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasSummary</AllowedRole>
    <AllowedRole>IsSummaryOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>Supplement</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasSupplement</AllowedRole>
    <AllowedRole>Supplements</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>Transformation</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasTransformation</AllowedRole>
    <AllowedRole>IsTransformationOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>

<LinkType>
    <TypeName>Translation</TypeName>
    <Degree>2</Degree>
    <AllowedRole>HasTranslation</AllowedRole>
    <AllowedRole>IsTranslationOf</AllowedRole>
    <RoleNameUniqueness>true</RoleNameUniqueness>
    <LinkUniqueness>true</LinkUniqueness>
</LinkType>
```

```
        <LinkType>
            <TypeName>Embodiment</TypeName>
            <Degree>2</Degree>
            <AllowedRole>IsEmbodiedIn</AllowedRole>
            <AllowedRole>Embodies</AllowedRole>
            <RoleNameUniqueness>true</RoleNameUniqueness>
            <LinkUniqueness>true</LinkUniqueness>
        </LinkType>

        <LinkType>
            <TypeName>Example</TypeName>
            <Degree>2</Degree>
            <AllowedRole>IsExemplifiedBy</AllowedRole>
            <AllowedRole>IsExampleOf</AllowedRole>
            <RoleNameUniqueness>true</RoleNameUniqueness>
            <LinkUniqueness>true</LinkUniqueness>
        </LinkType>

        <LinkType>
            <TypeName>Realization</TypeName>
            <Degree>2</Degree>
            <AllowedRole>IsRealizedThrough</AllowedRole>
            <AllowedRole>IsRealizationOf</AllowedRole>
            <RoleNameUniqueness>true</RoleNameUniqueness>
            <LinkUniqueness>true</LinkUniqueness>
        </LinkType>

    </Typology>

</Typologies>
```

References

[1] Trond Aalberg. Linking Information with Distributed Objects. In Panos Constantopoulos and Ingeborg T. Sølvberg, editors, *Proceedings of the 5th European Conference on Research and Advanced Technology for Digital Libraries, ECDL 2001*, number 2163 in Lecture Notes in Computer Science, pages 149–160. Springer-Verlag, 2001.

[2] Trond Aalberg. Navigating in Bibliographic Catalogues. In Maristella Agosti and Constantino Thanos, editors, *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries, ECDL 2002*, number 2458 in Lecture Notes in Computer Science, pages 238–250. Springer-Verlag, 2002.

[3] Nabil R. Adam and Yelena Yesha. Introduction. *International Journal on Digital Libraries*, 1(1), 1997.

[4] Robert M. Akscyn and Donald L. McCracken. PLEXUS : A Hypermedia Architecture for Large-Scale Digital Libraries. In *Proceedings of the 11th Annual International Conference on Systems Documentation*, pages 11–20. ACM Press, 1993.

[5] American National Standards Institute. Serial Item and Contribution Identifier (SICI). ANSI/NISO standard Z39.56-1996 (Version 2), August 1996.

[6] American National Standards Institute. Syntax for the Digital Object Identifier. ANSI/NISO standard Z39.84-2000, May 2000.

[7] Kenneth M. Anderson, Richard N. Taylor, and Jr. E. James Whitehead. Chimera : Hypertext for Heterogeneous Software Environments. In *Proceedings of the 1994 ACM European Conference on Hypermedia Technology*, pages 94–107. ACM Press, 1994.

[8] Kenneth M. Anderson, Richard N. Taylor, and Jr. E. James Whitehead. Chimera : Hypermedia for Heterogeneous Software Development Enviroments. *ACM Transactions on Information Systems (TOIS)*, 18(3):211–245, 2000.

[9] William Y. Arms. Key Concepts in the Architecture of the Digital Library. *D-Lib Magazine*, July 1995.

[10] William Y. Arms, Christophe Blanchi, and Edward A. Overly. An Architecture for Information in Digital Libraries. *D-Lib Magazine*, 3(2), February 1997.

URN:NBN:no-3403

[11] Association of Research Libraries. Appendix II : Definition and Purposes of a Digital Library. In *Realizing Digital Libraries : Proceedings of the 126th Annual Meeting*. Association of Research Libraries, 1995.

[12] Donna Bergmark. Collection Synthesis. In *Proceedings of the Second ACM/ IEEE-CS Joint Conference on Digital Libraries*, pages 253–262. ACM Press, 2002.

[13] T. Berners-Lee, R. Fielding, U.C. Irvine, and L. Masinter. Uniform Resource Identifiers (URI) : Generic Syntax. RFC 2396, The Internet Engineering Task Force, August 1998.

[14] Philip A. Bernstein. Middleware : A Model for Distributed Services. *Communications of the ACM*, 39(2):86–97, February 1996.

[15] G. Birtwistle, O. Dahl, B. Myrthaug, and K. Nygaard. *Simula Begin*. Auerbach Press, 1973.

[16] Fintan Bolton. *Pure CORBA : A Code-Intensive Premium Reference*. Sams Publishing, 2002.

[17] Magnus Boman, Janis A. Bubenko Jr., Paul Johanneson, and Benkt Wangler. *Conceptual Modelling*. Prentice Hall Series in Computer Science. Prentice Hall, 1997.

[18] Grady Booch, Ivar Jacobsen, and James Rumbaugh. *The Unified Modeling Language : User Guide*. Addison-Wesley, 1999.

[19] Christine L. Borgman. *From Gutenberg to the Global Information Infrastructure : Access to Information in the Networked World*. Digital Libraries and Electronic Publishing. The MIT Press, 2000.

[20] Christine L. Borgman, M.J. Bates, M.V. Cloonan, E.N. Efthimiadis, A. Gilliland-Swetland, Y. Kafai, G.L. Leazer, and A. Maddox. Social Aspects of Digital Libraries. Final Report to the National Science Foundation, 1996.

[21] Jan Bosch. Relations as Object Model Components. *Journal of Programming Languages*, 4(1):39–61, 1996.

[22] Niels Olof Bouvin. Experiences with OHP and Issues for the Future. In S. Reich and K.M. Anderson, editors, *Proceedings of the 6th International Workshop on Open Hypermedia Systems, OHS-6*, number 1903 in Lecture Notes in Computer Science, pages 13–22. Springer-Verlag, 2000.

[23] Niels Olof Bouvin, Polle T. Zellweger, Kaj Grønbæk, and Jock D. Mackinlay. Fluid Annotations through Open Hypermedia : Using and Extending Emerging Web Standards. In *Proceedings of the Eleventh International Conference on World Wide Web*, pages 160–171. ACM Press, 2002.

[24] Svein Erik Bratsberg. FOOD : Supporting Explicit Relations in a Fully Object-Oriented Database. In Robert A. Meersman, William Kent, and Asmit Khosla, editors, *Proceedings of the IFIP TC2/WG 2.6 Working Conference on Object-Oriented Databases : Analysis, Design & Construction (DS-4)*, pages 123–139. Elsevier Science Pubslisher, 1990.

[25] Gerald Brose, Andreas Vogel, and Keith Duddy. *Java Programming with CORBA : Advanced Techniques for Building Distributed Applications*. John Wiley & Sons, 2001.

[26] Michael Buckland. What is a ”Document”? *Journal of the American Society for Information Science*, 48(9):804–809, 1997.

[27] Mario Augusto Bunge. *Philosophy of Science : From Explanation to Justification*, volume 2 of *Philosophy of Science*. Transaction Publisher, 1998.

[28] Mario Augusto Bunge. *Philosophy of Science : From Problem to Theory*, volume 1 of *Philosophy of Science*. Transaction Publisher, 1998.

[29] Vannevar Bush. As We May Think. *Atlantic Monthly*, 1945.

[30] Robert Cailliau. A Little History of the World Wide Web from 1945 to 1995, 1995. Available online at http://www.w3.org/History.html

[31] Licia Calvi and Paul De Bra. Improving the Usability of Hypertext Courseware Through Adaptive Linking. In *Proceedings of the Eight ACM Conference on Hypertext*, pages 224–225. ACM Press, 1997.

[32] Priscilla Caplan and William Y. Arms. Reference Linking for Journal Articles. *D-Lib Magazine*, 5(7/8), 1999.

[33] S. Carmody, T. Gross, T. Nelson, D. Rice, and A. van Dam. A Hypertext Editing System for the 360. In *Proceedings Conference in Computer Graphics*. University of Illinois, 1969.

[34] Leslie Carr, Wendy Hall, and David De Roure. The Evolution of Hypertext Link Services. *ACM Computing Surveys*, 31(4es), 1999.

[35] Donatella Catelli and Paquale Pagano. OpenDLib : A Digital Library Service System. In Maristella Agosti and Constantino Thanos, editors, *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries, ECDL 2002*, number 2458 in Lecture Notes in Computer Science, pages 292–308. Springer-Verlag, 2002.

[36] R.G.G. Cattell and Douglas K. Barry, editors. *The Object Data Standard : ODMG 3.0*. Morgan Kaufmann Publishers, 2000.

[37] Peter Pin-Shan Chen. The Entity Relationship Model : Towards a Unified View of Data. *ACM Transactions on Database Systems*, 1(1), 1976.

[38] Gregory Chockler, Roy Friedman, and Roman Vitenberg. Consistency Conditions for a CORBA Caching Service. In Maurice Herlihy, editor, *Proceedings of the 4th International Conference on Distributed Computing*, number 1914 in Lecture Notes in Computer Science, pages 374–388. The International Symposium on Distributed Computing, Springer-Verlag, 2000.

[39] C. G. Chowdury. *Introduction to Modern Information Retrieval*. Library Association Publishing, 1999.

[40] Paolo Ciancarini, Federico Folli, Davide Rossi, and Fabio Vitali. Linking Documents : XLinkProxy : External Linkbases with XLink . In *Proceedings of the 2002 ACM Symposium on Document Engineering*, pages 57–65. ACM Press, 2002.

[41] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems : Concepts and Design*. Addison-Wesley, 1998.

[42] Gregory Crane, David A. Smith, and Clifford E. Wulfman. Building a Hypertextual Digital Library in the Humanities : A Case Study on London. In *Proceedings of the First ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 426–434. ACM Press, 2001.

[43] Charles A. Cutter. *Rules for a Dictionary Catalogue : Special Report on Public Libraries*. Government Printing Office, 1904.

[44] L. Daigle, D. van Gulik, and R. Ianella. URN Namespace Definition Mechanisms. RFC 2611, The Internet Engineering Task Force, June 1999.

[45] Hugh Davis. To Embed or Not to Embed. *Communications of the ACM*, 38(6):108–109, 1995.

[46] Hugh Davis, Wendy Hall, Ian Heath, Gary Hill, and Rob Wilkins. Towards an Integrated Information Environment with Open Hypermedia Systems. In *Proceedings of the ACM Conference on Hypertext*, pages 181–190. ACM Press, 1992.

[47] Hugh C. Davis. Referential Integrity of Links in Open Hypermedia Systems. In *Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia : Links, Objects, Time and Space : Structure in Hypermedia Systems*, pages 207–216. ACM Press, 1998.

[48] Hugh C. Davis. Hypertext Link Integrity. *ACM Computing Surveys*, 31(4es), 1999.

[49] Hugh C. Davis, Simon Knight, and Wendy Hall. Light Hypermedia Link Services : A Study of Third Party Application Integration. In *Proceedings of the 1994 ACM European Conference on Hypermedia Technology*, pages 41–50. ACM Press, 1994.

[50] Hugh Charles Davis, Siegfried Reich, and David Millard. A Proposal for a Common Navigational Hypertext Protocol, 1997. Available online at http://www.ecs.soton.ac.uk/ hcd/ohp/ohp35.htm

[51] Jim Davis, David Fielding, Carl Lagoze, and Richard Marisa. Dienst : Overview and Introduction, 2000. Available online at http://www.cs.cornell.edu/cdlrg/ dienst/
DienstOverview.htm

[52] Randall Davis, Howard Shrobe, and Peter Szolovits. What is a Knowledge Representation? *AI Magazine*, 14(1):17–33, 1993.

[53] Herbert Van de Sompel and Oren Beit-Arie. Open Linking in the Scholarly Information Environment Using the OpenURL Framework. *D-Lib Magazine*, 7(3), March 2001.

[54] Digital Libraries : Future Research Directions for a European Research Programme. Workshop report 02/W02, ERCIM, 2002.

[55] Tom Delsey. Functional Analysis of the MARC 21 Bibliographic and Holdings Format, 2002. Available online at
http://www.loc.gov/marc/marc-functional-analysis/home.html

[56] Lorcan Dempsey and Rachel Heery. A Review of Metadata : A Survey of Current Resource Description Formats, 1997. Available online at
http://www.ukoln.ac.uk/metadata/DESIRE/overview/

[57] Oscar Diaz and Norman W. Paton. Extending ODBMSs Using Metaclasses. *IEEE Software*, 11(3), May 1994.

[58] Karen M. Drabenstott. Analytical Review of the Library of the Future, 1994. Washington, DC, Council Library Resources.

[59] Dublin Core Metadata Initiative. DCMI Elements and Element Refinements : A Current List, 2002. Available online at http://dublincore.org/documents/2002/ 10/06/
current-elements/

[60] Dublin Core Metadata Initiative. Dublin Core Metadata Element Set, Version 1.1 : Reference Description. DCMI recommendation, 2003. Available online at
http://dublincore.org/documents/2003/02/04/dces/

[61] Naomi Dushay. Localizing Experience of Digital Content Via Structural Metadata. In *Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 244–252. ACM Press, 2002.

[62] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings publishing Company, 1994.

[63] Douglas C. Engelbart and William K. English. A Research Center for Augmenting Human Intellect. In *AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference*, December 1968.

[64] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions : (MIME) Part Two : Media Types. RFC 2046, The Internet Engineering Task Force, November 1996.

[65] Roy Friedman and Erez Hadad. Client-Side Enhancements Using Portable Interceptors. In *Proceedings of the Sixth International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 153–160. IEEE, 2001.

[66] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1994.

[67] Michael Gasser and Eliana Colunga. Developing Relations. In Emile van der Zee and Urpo Nikanne, editors, *Cognitive Interfaces : Constraints on Linking Cognitive Information*, pages 185–214. Oxford University Press, 2001.

[68] Gary Geisler, Sarah Giersch, David McArthur, and Marty McClelland. Creating Virtual Collections in Digital Libraries : Benefits and Implementation Issues. In *Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 210–218. ACM Press, 2002.

[69] Marcos Andre Goncalves, Robert K. France, and Edward A. Fox. MARIAN : Flexible Interoperability for Federated Digital Libraries. In Panos Constantopoulos and Ingeborg T. Sølvberg, editors, *Proceedings of the 5th European Conference on Research and Advanced Technology for Digital Libraries, ECDL 2001*, number 2163 in Lecture Notes in Computer Science, pages 173–186. Springer-Verlag, 2001.

[70] Marcos Andre Goncalves, Paul Mather, Jun Wang, Ye Zhou, Ming Luo, Ryan Richardson, Rao Shen, Liang Xu, , and Edward A. Fox. Java MARIAN : From an OPAC to a Modern Digital Library System. In A.L. Oliveira A.H.F. Laender, editor, *Proceedings of the 9th International Symposium on String Processing and Information Retrieval, SPIRE 2002*, number 2476 in Lecture Notes in Computer Science, pages 194–209. Springer-Verlag, 2002.

[71] S. Goose, A Lewis, and H. Davis. OHRA : Towards an Open Hypermedia Reference Architecture and a Migration Path for Existing Systems. In *Proceedings of the 3rd Workshop on Open Hypermedia Systems, OHS-3*, Scientific Report 97-01. The Danish National Centre for IT Research, 1997.

[72] Michael Gorman and Paul W. Winkler. *Anglo-American Cataloguing Rules*. Library Association Publ., 1988.

[73] Rebecca Green. Relationships in the Organization of Knowledge : An Overview. In Carol A. Bean and Rebecca Green, editors, *Relationships in the Organization of Knowledge*, number 2 in Information Science and Knowledge Management, chapter 1, pages 3–18. Kluwer Academic Publishers, 2001.

[74] Stephen M. Griffin. Taking the Initiative for Digital Libraries. *The Electronic Library*, 16(1):24–27, February 1998. Interview.

[75] Object Management Group. Concurrency Service Specification. OMG specification, 2000.

[76] Object Management Group. OMG Unified Modeling Language : Specification. OMG specification, 2000.

[77] Object Management Group. Relationship Service Specification. OMG specification, 2000.

[78] Object Management Group. Trading Object Service Specification. OMG specification, 2000.

[79] Object Management Group. Naming Service Specification. OMG specification, 2001.

[80] Object Management Group. Resource Access Decision Facility. OMG specification, 2001.

[81] Object Management Group. Security Service Specification. OMG specification, 2001.

[82] Object Management Group. Common Object Request Broker Architecture : Core Specification. OMG specification, 2002.

[83] Object Management Group. Persistent State Service Specification. OMG specification, 2002.

[84] Object Management Group. Property Service Specification. OMG specification, 2002.

[85] Object Management Group. Transaction Service Specification. OMG specification, 2002.

[86] Frank Halasz and Mayer Schwartz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, 1994.

[87] Wendy Hall, Hugh Davis, and Gerard Hutchings. *Rethinking Hypermedia : The Microcosm Approach*. Kluwer Academic Publisher, 1996.

[88] Wendy Hall, Gary Hill, and Hugh Davis. The Microcosm Link Service. In *Proceedings of the Fifth ACM Conference on Hypertext*, pages 256–259. ACM Press, 1993.

[89] Lynda Hardman, Dick C. A. Bulterman, and Guido van Rossum. Links in Hypermedia : The Requirement for Context. In *Proceedings of the Fifth ACM Conference on Hypertext*, pages 183–191. ACM Press, 1993.

[90] Lynda Hardman, Dick C. A. Bulterman, and Guido van Rossum. The Amsterdam Hypermedia Model : Adding Time and Context to the Dexter model. *Communications of the ACM*, 37(2):50–62, 1994.

[91] Knut Hegna and Eeva Murtomaa. *Data Mining MARC to Find : FRBR?* BIBSYS/HUL, 2002.

[92] Michi Henning. Binding, Migration, and Scalability in CORBA. *Communications of the ACM*, 41(10):62–71, 1998.

[93] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, 1999.

[94] Beth Hetzler. Beyond Word Relations : SIGIR '97 Workshop. *ACM SIGIR Forum*, 31(2):28–33, 1997.

[95] Thomas B. Hickey, Edward T. O'Neill, and Jenny Toves. Experiments with the IFLA Functional Requirements for Bibliographic Records (FRBR) . *D-Lib Magazine*, 8(9), September 2002.

[96] David L. Hicks, John J. Leggett, Peter J. Nürnberg, and John L. Schnase. A Hypermedia Version Control Framework. *ACM Transactions on Information Systems (TOIS)*, 16(2):127–160, 1998.

[97] Gary Hill and Wendy Hall. Extending the Microcosm Model to a Distributed Environment. In *Proceedings of the 1994 ACM European Conference on Hypermedia Technology*, pages 32–40. ACM Press, 1994.

[98] Soochan Hwang and Sukho Lee. Modelling Semantic Relationships and Constraints in Object-Oriented Databases. In *Proceedings of the 1990 ACM SIGBDP Conference on Trends and Directions in Expert Systems*, pages 396–416. ACM Press, 1990.

[99] IEEE. IEEE Standard for Learning Object Metadata. IEEE standard 1484.12.1, September 2002.

[100] IFLA Study Group on the Functional Requirements for Bibliographic Records. Functional Requirements for Bibliographic Records. *UBCIM Publications - New Series*, 19, 1998.

[101] David Ingham, Steve Caughey, and Mark Little. Fixing the "Broken-Link" Problem : The W3Objects Approach. In *Proceedings of the Fifth International World Wide Web Conference*. Elsevier Science, 1996.

[102] Carnegie Mellon Software Engineering Institute. Software Technology Review : Middleware, January 1997. Available online at http://www.sei.cmu.edu/str/descriptions/middleware.html

[103] International Federation of Library Assosiations and Institutions. ISBD(G) : General International Standard Bibliographic Description. *UBCIM Publications - New Series*, 6, 1992.

[104] International Federation of Library Assosiations and Institutions. ISBD(M) : International Standard Bibliographic Description. Standard 2002 Revision, Approved by the Standing Committee of the IFLA Section on Cataloguing, 2002.

[105] International Organization for Standardization. Standard Generalized Markup Language (SGML). International standard ISO 8879:1986, 1986.

[106] International Organization for Standardization. Information Processing Systems : Open Systems Interconnection Reference Model : Part 2 : Security Architecture, 1989. International standard ISO/IEC 7498-2:1998, 1989.

[107] International Organization for Standardization. International Standard Book Number (ISBN). International standard ISO 2108, 1992.

[108] International Organization for Standardization. Information and Documentation : Format for Information Interchange. International standard ISO 2709:1996 Third edition, 1996.

[109] International Organization for Standardization. Hypermedia/Time-Based Structuring Language (HyTime). International standard ISO/IEC 10744:1977, 1997.

[110] International Organization for Standardization. Information Retrieval (Z39.50) : Application Service Definition and Protocol Specification. International standard ISO 23950:1998, 1998.

[111] International Organization for Standardization. International Standard Serial Number (ISSN). International standard ISO 3297, 1998.

[112] International Organization for Standardization. Topic Maps. International standard ISO/IEC 13250:2000, 2000.

[113] Tomas Isakowitz, Edward A. Stohr, and P. Balasubramanian. RMM : A Methodology for Structured Hypermedia Design. *Communications of the ACM*, 38(8), 1995.

[114] Joint US/EU ad hoc Agent Markup Language Committee. DAML+OIL. Language specification, DAML, March 2001. Available online at http://www.daml.org/2001/03/daml+oil-index.html

[115] Robert Kahn and Robert Wilensky. A Framework for Distributed Digital Object Services, 1995. Available online at http://www.cnri.reston.va.us/home/cstr/arch/k-w.html

[116] Hanna Kemppainen. Designing a Mediator for Managing Relationships between Distributed Objects. In Andrej Vckovski, Kurt E. Brassel, and Hans-Jörg Schek, editors, *Proceedings of the Second International Conference on Interoperating Geographic Information Systems, INTEROP'99*, volume 1580 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[117] Manuel Kolp. A Metaobject Protocol for Reifying Semantic Relationships into Open Systems. In *Proceedings of the 4th Doctoral Consortium of the 9th Internation Conference on Advanced Information Systems Engineering, CAiSE'97*, pages 89–100, 1997.

[118] Manuel Kolp. *A Metaobject Protocol for Integrating Full-Fledged Relationships into Reflective Systems*. Ph.d. thesis, Université libre de Bruxelles, 1999.

[119] Richard W. Kopak. Functional Link Typing in Hypertext. *ACM Computing Surveys*, 31(4es), 1999.

[120] Carl Lagoze and Herbert Van de Sompel. The Open Archives Initiative : Building a Low-Barrier Interoperability Framework. In *Proceedings of the first ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 54–62. ACM Press, 2001.

[121] Ulrich Lang and Rudolf Schreiner. *Developing Secure Distributed Systems with CORBA*. Computer Security Series. Artech House, 2002.

[122] R. Langacker. Nouns and Verbs. *Language*, 63:53–94, 1987.

[123] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital Libraries and Autonomous Citation Indexing. *IEEE Computer*, 32(6):67–71, 1999.

[124] Paul J. Leach and Rich Salz. UUIDs and GUIDs. Internet draft, Internet Engineering Task Force, February 1998. Work in progress.

[125] Gregory H. Leazer and Richard P. Smiraglia. Toward the Bibliographic Control of Works : Derivative Bibliographic Relationships in an Online Union Catalog. In *Proceedings of the 1st ACM International Conference on Digital Libraries*, pages 36–43. ACM, 1996.

[126] Tim Berners Lee. Information Management : A Proposal. Conseil Europeen pour la Recherche Nucleaire (CERN), 1989.

[127] John J. Leggett and John L. Schnase. Viewing Dexter with Open Eyes. *Communications of the ACM*, 37(2):76–86, 1994.

[128] Barry M. Leiner. The NCSTRL Approach to Open Architecture for the Confederated Digital Library. *D-Lib Magazine*, December 1998.

[129] Paul H. Lewis, Wendy Hall, Leslie A. Carr, and David De Roure. The Significance of Linking. *ACM Computing Surveys*, 31(4es), 1999.

[130] Library of Congress. MARC Standards, 2000. Available online at http://www.lcweb.loc.gov/marc/

[131] Jean Tague-Sutcliff Lisa Baron and Marc T. Kinnucan. Labeled, Typed Links as Cues when Reading Hypertext Documents. *Journal of the American Association for Information Science*, 47(12):896–908, 1996.

[132] Li min Liu and Michael Halper. Incorporating Semantic Relationships into an Object-Oriented Database System. In *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences, HICSS-32*, pages 1–10, 1999.

[133] Gary Marchionini. *Information Seeking in Electronic Environments*, volume 9 of *Cambridge Series on Human-Computer Interaction*. Cambridge University Press, 1995.

[134] M. Mercedes Mart, Pablo de la Fuente, Jean-Claude Derniame, and Alberto Pedrero. Relationship-Based Dynamic Versioning of Evolving Legal Documents. In J. G. Carbonell and J. Siekmann, editors, *Web Knowledge Management and Decision Support, Proceedings of the 14th International Conference on Applications of Prolog, INAP 2001*, number 2543 in Lecture Notes in Artificial Intelligence, pages 290–305. Springer-Verlag, 2001.

[135] Mercedes Martinez, Jean-Claude Derniame, and Pablo de la Fuente. A Method for the Dynamic Generation of Virtual Versions of Evolving Documents. In *Proceedings of the 17th ACM Symposium on Applied Computing*, pages 476–482. ACM Press, 2002.

[136] M. Mealling and R. Denenberg. Report from the Joint W3C/IETF URI Planning Interest Group : Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names. RFC 3305, The Internet Engineering Task Force, August 2002.

[137] M. Mealling, P. Leach, and R. Salz. A UUID URN Namespace. Internet draft, Internet Engineering Task Force, October 2002. Work in progress.

[138] D. E. Millard, S. Reich, and H. C. Davis. Reworking OHP : The Road to OHP-Nav. In U. K. Wiil, editor, *Proceedings of the 4th Workshop on Open Hypermedia Systems*, pages 48–53, 1998.

[139] Dave E. Millard, Luc Moreau, Hugh C. Davis, and Siegfried Reich. FOHM : A Fundamental Open Hypertext Model for Investigating Interoperability Between Hypertext Domains. In *Proceedings of the Eleventh ACM Conference on Hypertext and Hypermedia*, pages 93–102. ACM Press, 2000.

[140] R. Moats. URN Syntax. RFC 2141, The Internet Engineering Task Force, May 1997.

[141] Graham Moore. RDF and TopicMaps : An Exercise in Convergence. In *XML Europe 2001*, 2001. Available online at http://www.topicmaps.com/topicmapsrdf.pdf

[142] Graham Moore and Luc Moreau. From Metadata to Links. In S. Reich and K.M. Anderson, editors, *Proceedings of the 6th International Workshop on Open Hypermedia Systems, OHS-6*, number 1903 in Lecture Notes in Computer Science, pages 77–86. Springer-Verlag, 2000.

[143] Jocelyne Nanard and Marc Nanard. Should Anchors be Typed too? : An Experiment with MacWeb. In *Proceedings of the Fifth ACM Conference on Hypertext*, pages 51–62. ACM Press, 1993.

[144] Rodolfo Nassif, Yuping Qiu, and Jianhua Zhu. Extending the Object-Oriented Paradigm to Support Relationships and Constraints. In Robert A. Meersman, William Kent, and Asmit Khosla, editors, *Proceedings of the IFIP TC2/WG 2.6 Working Conference on Object-Oriented Databases : Analysis, Design & Construction (DS-4)*, pages 305–329. Elsevier Science Publisher, 1990.

[145] Michael L. Nelson, Kurt Maly, Mohammad Zubair, and Stewart N. T. Shen. SODA : Smart Objects, Dumb Archives. In *Proceedings of the Third European Conference on Research and Advanced Technology for Digital Libraries, ECDL 99*, volume 1696 of *Lecture Notes in Computer Science*. Springer, 1999.

[146] Ted Nelson. A File Structure for the Complex, the Changing and the Indeterminate. In *proceedings of the ACM 20th national conference*. ACM, 1965.

[147] Christian Nentwich, Licia Capra, Wolfgang Emmerich, and Anthony Finkelstein. xlinkit : A Consistency Checking and Smart Link Generation Service. *ACM Transactions on Internet Technology (TOIT)*, 2(2):151–185, 2002.

[148] Craig Nevill-Manning. The Biological Digital Library. *Communications of the ACM*, 44(5):41–42, 2001.

[149] Tien Nguyen, Satish Chandra Gupta, and Ethan V. Munson. Versioned Hypermedia can Improve Software Document Management. In *Proceedings of the Thirteenth Conference on Hypertext and Hypermedia*, pages 192–193. ACM Press, 2002.

[150] James Noble and John Grundy. Explicit Relationships in Object-Oriented Development. In *Proceedings of the 18th Conference on Technology of Object-Oriented Languages and Systems, TOOLS 18*. Prentice-Hall, 1995.

[151] P. J. Nürnberg. HOSS : An Environment to Support Structural Computing. Ph.d. thesis, Department of Computer Science, Texas A&M University, 1997.

[152] Peter J. Nürnberg, John J. Leggett, and Erich R. Scneider. As We Should Have Thought. In *Proceedings of the Eighth ACM conference on Hypertext*, pages 96–101. ACM Press, 1997.

[153] Peter J. Nürnberg, Uffe K. Wiil, and John J. Leggett. Structuring Facilities in Digital Libraries. In *Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries, ECDL 98*, volume 1513 of *Lecture Notes in Computer Science*. Springer, 1998.

[154] OASIS. UDDI Version 3.0 . UDDI Spec TC Committee specification, OASIS, July 2002.

[155] Library of Congress. *Displays for Multiple Versions from MARC 21 and FRBR*. Network Development and MARC Standards Office, Library of Congress, 2002.

[156] Charles Kay Ogden and Ivor Armstrong Richards. *The Meaning of Meaning : A Study of the Influence of Language upon Thought and of the Science of Symbolism*. Routledge & Kegan Paul, 1923.

[157] Open Archives Initiative. The Open Archives Initiative Protocol for Metadata Harvesting, 2001. Available online at http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm

[158] Rolf Oppliger. *Internet and Intranet Security.* Computer Security Series. Artech House, 2002.

[159] Kasper Østerbye and Uffe Kock Wiil. The Flag Taxonomy of Open Hypermedia Systems. In *Proceedings of the Seventh ACM Conference on Hypertext*, pages 129–139. ACM Press, 1996.

[160] Andreas Paepcke, Michelle Q. Wang Baldonado, Chen-Chuan K. Chang, Steve Cousins, and Hector Garcia-Molina. Using Distributed Objects to Build the Stanford Digital Library Infobus. *Computer*, 32(2):80–87, 1999.

[161] Sandra Payette, Christophe Blanchi, Carl Lagoze, and Edward Overly. Interoperability for Digital Objects and Repositories : The Cornell/CNRI Experiments. *D-Lib Magazine*, 5, May 1999.

[162] Sandra Payette and Carl Lagoze. Flexible and Extensible Digital Object and Repository Architecture (FEDORA). In Christos Nikolaou and Constantine Stephanidis, editors, *Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries, ECDL 98*, number 1513 in Lecture Notes in Computer Science, pages 41–60. Springer, 1998.

[163] A. Pearl. Sun's Link Service : A Protocol for Open Linking. In *Proceedings of the Second Annual ACM Conference on Hypertext*, pages 137–146. ACM Press, 1989.

[164] Joan Peckham and Fred Maryanski. Semantic Data Models. *ACM Computing Surveys (CSUR)*, 20(3):153–189, 1988.

[165] Steve Pepper. The TAO of Topic Maps : Finding the Way in the Age of Infoglut. In *Proceedings of XML Europe 2000*. GCA, 2000. Available online at http://www.ontopia.net/topicmaps/materials/tao.html

[166] Walter D. Potter and Robert P. Trueblood. Traditional, Semantic, and Hypersemantic Approaches to Data Modeling. *Computer*, 21(6):53–63, 1988.

[167] Awais Rashid and Peter Sawyer. Dynamic Relationships in Object Oriented Databases : A Uniform Approach. In Trevor Bench-Capon, Giovanni Soda, and A Min Tjoa, editors, *Proceedings of the 10th International Conference on Database and Expert Systems Applications, DEXA'99*, number 1677 in Lecture Notes in Computer Science, pages 26–35. Springer-Verlag, 1999.

[168] Edie Rasmussen. Clustering Algoritms. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval*, chapter 16. Prentice Hall, 1992.

[169] Sigi Reich. Definitions and Examples of the Open Hypermedia Protocol (OHP), 1999.

[170] Han Reichgelt. *Knowledge Representation : An AI Perspective*. Ablex Publishing Corporation, 1991.

[171] Antoine Rizk and Louis Sauter. Multicard : An Open Hypermedia System. In *Proceedings of the ACM Conference on Hypertext*, pages 4–10. ACM Press, 1992.

[172] Martin Roscheisen, Michelle Baldonado, Kevin Chang, Luis Gravano, Steven Ketchpel, and Andreas Paepcke. The Stanford InfoBus and Its Service Layers : Augmenting the Internet with Higher-Level Information Management Protocols. Technical report, Stanford, 1997.

[173] David C. De Roure, Leslie A. Carr, W. Hall, and G. Hill. A Distributed Hypermedia Service. In *Proceedings of the 3rd Workshop on Services in Distributed and Networked Environments (SNDE'96)*. IEEE, 1996.

[174] David C. De Roure, Nigel G. Walker, and Leslie A. Carr. Investigating Link Service Infrastructures. In *Proceedings of the Eleventh ACM on Hypertext and Hypermedia*, pages 67–76. ACM Press, 2000.

[175] James Rumbaugh. Relations as Semantic Constructs in an Object-Oriented Language. In *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications*, pages 466–481. ACM Press, 1987.

[176] James Rumbaugh, Ivar Jacobsen, and Grady Booch. *The Unified Modelling Language Reference Manual*. Addison-Wesley, 1999.

[177] Lloyd Rutledge and Lynda Hardman. Applying the HyTime Model to the Open Hypermedia Protocol LocSpec. In Uffe K. Wiil, editor, *Proceedings of the 3rd Workshop on Open Hypermedia Systems*, CIT Scientific Report no. SR-97-01. The Danish National Centre for IT Research, 1997.

[178] Peter Schäuble and Alan F. Smeaton. An International Research Agenda for Digital Libraries : Summary Report of the Series of Joint NSF-EU Working Groups on Future Directions for Digital Libraries Research. DELOS, 1998.

[179] Jürgen Schlegelmilch. An Advanced Relationship Mechanism for Object-Oriented Databases. Technical Report 19/1996, University of Rostock, Computer Science Dept., 1996.

[180] Robert C. Seacord. Replaceable Components and the Service Provider Interface. In J. Dean and A. Gravel, editors, *Proceedings of the First International Conference on COTS-Based Software Systems, ICCBSS 2002*, number 2255 in Lecture Notes in Computer Science, pages 222–233. Springer-Verlag, February 4-6 2002.

[181] Keith Shafer, Stuart Weibel, and Jon Fausey Erik Jul. Introduction to Persistent Uniform Resource Locators. OCLC Online Computer Library Center, Inc. Available online at http://purl.oclc.org/docs/inet96.html

[182] A. V. Shah, J. H. Hamel, R. A. Borsari, and J. E. Rumbaugh. DSM : An Object-Relationship Modeling Language. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 191–202. ACM Press, 1989.

[183] Chien-Chung Shen and John Y. Wei. Network as Distributed Object Database. In *Proceedings of the 1998 IEEE Network Operations and Management Symposium*, volume 2, pages 540–548. IEEE, 1998.

[184] Dirk Slama, Jason Garbis, and Perry Russell. *Enterprise CORBA*. Prentice Hall, 1999.

[185] Dagobert Soergel. A Framework for Digital Library Resaearch. *D-Lib Magazine*, 8(12), December 2002.

[186] Arne Sølvberg. Conceptual Modeling in a World of Models. In R. Kaschek, editor, *Entwiclungsmethoden für Informationssysteme und deren Anwendung : EMISA '99*, pages 63–92. B.G. Teubner, 1999.

[187] Herbert Van de Sompel and Patrick Hochstenbach. Reference Linking in a Hybrid Library Environment : Part 2 : SFX, a Generic Linking Solution. *D-Lib Magazine*, 5(4), April 1999.

[188] Norbert Streitz, Jörg Haake, Jörg Hannemann, Andreas Lemke, Wolfgang Schuler, Helge Schütt, and Manfred Thüring. Sepia : A cooperative hypermedia authoring environment. In *Proceedings of the ACM conference on Hypertext*, pages 11–22. ACM Press, 1992.

[189] Sam X. Sun and Larry Lannom. Handle System Overview. Internet draft, The Internet Engineering Task Force, 2002. Work in progress.

[190] Sam X. Sun, Sean Reilly, and Larry Lannom. Handle System Namespace and Service Definition. Internet draft, The Internet Engineering Task Force, 2002. Work in progress.

[191] Clemens Szyperski. *Component Software : Beyond Object-Oriented Programming*. ACM Press / Addison Wesley, 1999.

[192] Zahir Tari and Omran Bukhres. *Fundamentals of Distributed Object Systems : The CORBA Perspective*. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, 2001.

[193] Zahir Tari and Herry Hamidjadja. A CORBA Cooperative Cache Approach with Popularity Admission and Routing Mechanism. In *Proceedings of the Thirteenth Australasian Conference on Database Technologies*, pages 177–186. Australian Computer Society, Inc., 2002.

[194] Zahir Tari, Herry Hamidjaja, and Qi Tang Lin. Cache Management in CORBA Distributed Object Systems. *IEEE Concurrency*, 8(3):48–55, July–September 2000.

[195] The Open Group. DCE 1.1 : Remote Procedure Call. Technical Standard C706, The Open Group, August 1997.

[196] Barbara B. Tillett. A Taxonomy of Bibliographic Relationships. *Library Resources & Technical Services,*, 35(2):150–158, 1991.

[197] Barbara B. Tillett. Bibliographic Relationships. In Carol A. Bean and Rebecca Green, editors, *Relationships in the Organization of Knowledge*, number 2 in Information Science and Knowledge Management, chapter 2, pages 19–35. Kluwer Academic Publishers, 2001.

[198] TopicMaps.Org. XML Topic Maps (XTM) 1.0. Topicmaps.org specification, 2001. Available online at http://www.topicmaps.org/xtm/1.0/xtm1-20010806.html

[199] Randall H. Trigg. A Networked Approach to Text Handling for the Online Scientific Community. Ph.D. thesis TR-1346, University of Maryland, Department of Computer Science, University of Maryland, College Park MD 20742, November 1983.

[200] Manolis Tzagarakis, Nikos Karousos, Dimitris Christodoulakis, and Siegfried Reich. Naming as a Fundamental Concept of Open Hypermedia Systems. In *Proceedings of the Eleventh ACM Conference on Hypertext and Hypermedia*, pages 103–112. ACM Press, 2000.

[201] Amjad Umar. *Object-Oriented Client/Server Internet Environment*. Prentice Hall, 1997.

[202] J. Verbyla and C. Watters. Cooperative Hypermedia Management Systems. *Journal of Digital Information*, 1(4), January 1999.

[203] Janet Verbyla. Unlinking the Link. *ACM Computing Surveys*, 31(4es), 1999.

[204] Steve Vinoski. Web Services Interaction Models : Current Practice. *IEEE Internet Computing*, 6(3):89–91, May/June 2002.

[205] Stephen Wagner and Zahir Tari. A Caching Protocol to Improve CORBA Performance. In *Proceedings of the 11th Australasian Database Conference*, pages 140–148. IEEE, 2000.

[206] Robert Wall. *Introduction to Mathematical Linguistics*. Prentice Hall, Inc., 1972.

[207] Bing Wang. A Hybrid System Approach for Supporting Digital Libraries. *International Journal on Digital Libraries*, 2(2/3):91–110, 1999.

[208] Mark J. Weal, Gareth V. Hughes, David E. Millard, and Luc Moreau. Open Hypermedia as a Navigational Interface to Ontological Information Spaces. In *Proceedings of the Twelfth ACM conference on Hypertext and Hypermedia*, pages 227–236. ACM Press, 2001.

[209] Peter Weinstein. Ontology-Based Metadata: Transforming the MARC Legacy. In *Proceedings of the Third International ACM Digital Library Conference*, pages 254–263. ACM, 1998.

[210] E. James Jr. Whitehead. Uniform Comparison of Data Models Using Containment Modeling. In *Proceedings of the Thirteenth Conference on Hypertext and Hypermedia*. ACM, 2002.

[211] Uffe K. Wiil and David L. Hicks. Requirements for Development of Hypermedia Technology for a Digital Library Supporting Scholarly Work. In *Proceedings of the 2000 ACM Symposium on Applied Computing 2000*, pages 607–609. ACM Press, 2000.

[212] Uffe K. Wiil, Peter J. Nürnberg, and John J. Leggett. Hypermedia Research Directions : An Infrastructure Perspective. *ACM Computing Surveys (CSUR)*, 31(4es):2, 1999.

[213] Uffe Kock Wiil. Evaluating HyperDisco as an Infrastructure for Digital Libraries. In *Proceedings of the 1998 ACM Symposium on Applied Computing*, pages 491–497. ACM Press, 1998.

[214] Uffe Kock Wiil and John J. Leggett. The HyperDisco Approach to Open Hypermedia Systems. In *Proceedings of the Seventh ACM Conference on Hypertext*, pages 140–148. ACM Press, 1996.

[215] Uffe Kock Wiil and Kasper Østerbye. Using the Flag Taxonomy to Study Hypermedia System Interoperabilty. In *Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia : Links, Objects, Time and Space : Structure in Hypermedia Systems*, pages 188–197. ACM Press, 1998.

[216] George Wilkie. *Object-Oriented Software Engineering : The Professional Developer's Guide*, chapter 4. Addison Wesley, 1993.

[217] Ian H. Witten, David Bainbridge, and Stefan Boddie. Greenstone : Open-Source DL software. *Communications of the ACM*, 44(5):47, 2001.

[218] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. W3C recommendation, February 1998.

[219] World Wide Web Consortium. Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. W3C recommendation, June 1998.

[220] World Wide Web Consortium. HTML 4.01 Specification. W3C recommendation, December 1999.

[221] World Wide Web Consortium. Resource Description Framework (RDF) : Model and Syntax Specification. W3C recommendation, February 1999.

[222] World Wide Web Consortium. XML Path Language (XPath) Version 1.0. W3C recommendation, November 1999.

[223] World Wide Web Consortium. XHTML 1.0 : The Extensible HyperText Markup Language : A Reformulation of HTML 4 in XML 1.0. W3C recommendation, January 2000.

[224] World Wide Web Consortium. XML Linking Language (XLink) Version 1.0. W3C recommendation, June 2001.

[225] World Wide Web Consortium. SOAP Version 1.2 Part 1 : Messaging Framework. W3C candidate recommendation, December 2002.

[226] World Wide Web Consortium. RDF Vocabulary Description Language 1.0 : RDF Schema. W3C working draft, January 2003.

[227] World Wide Web Consortium. Resource Description Framework (RDF) : Concepts and Abstract Syntax. W3C working draft, January 2003.

[228] World Wide Web Consortium. Web Services Description Language (WSDL) Version 1.2. W3C working draft, March 2003.

[229] W. Yeong, T. Howes, and S. Kille. Lightweight Directory Access Protocol. RFC 1777, The Internet Engineering Task Force, March 1995.