Master's Thesis in Telematics
for the Award of the Academic Degree
Diplom Ingenieur
at the
Graz University of Technology

Shared Bookmarks

Building an Application
on a
Distributed Object System

---

submitted by:

Philipp Zambelli

December 1999

Institute for Information Processing
and Computer Supported New Media

Supervisor: Univ.Prof. Dr. Dr. hc Hermann Maurer
Assessor: Univ.Ass. DI. Dr. techn. Klaus Schmaranz

Abstract

The present thesis discusses the process of building an application on a distributed object system. In particular it describes the design of a shared, multi-user bookmark collection that is accessible on the net. Because a shared bookmark collection can be seen as a kind of collaborative tool, first an overview of groupware and related services is given as a background information. Shared Bookmarks have been implemented using the Dino middleware system. The advantages of using a middleware system as a basis for such applications are figured out by discussing the requirements for the present application and by explaining the capabilities of the Dino middleware system. The most important technologies of todays network application programming in heterogeneous information systems are discussed in detail: Addressing and Linking, Distributed Objects and Security Considerations. The implementation itself is described in the later sections of this thesis.

Zusammenfassung

Die vorliegende Diplomarbeit beschreibt den Entwicklungsprozeß einer Anwendung, die auf ein verteiltes Objektsystem aufbaut. Im speziellen handelt es sich um eine verteilte Sammlung von Documentreferenzen, besser bekannt als "Bookmarks", auf welche von mehreren Benutzer zugegriffen werden kann. Da diese Anwendung die computer-gestützte Zusammenarbeit von mehreren Benutzern unterstützt, werden zuerst andere, verwandte Anwendungen und Dienste vorgestellt. Als Basis für dieses Projekt wurde das Dino Middleware System verwendet. Anhand der Anforderungen die gestellt wurden, werden die Vorteile beschrieben, die es mit sich bringt, wenn man eine solche Anwendungen auf einem Middleware System aufbaut. Die wichtigsten Technologien heutiger Netzwerkprogrammierung in heterogenen Informationssystemen werden im Detail diskutiert: Adressierung und Verknüpfungen, Verteilte Objekte und Überlegungen zur Sicherheit. Die Realisierung selbst ist im hinteren Teil dieser Arbeit beschrieben.

Acknowledgments

First of all I want to thank Klaus Schmaranz, since it is due to him that I could do this project. His ideas and inspirations heavily influenced my work and he supported me whenever I was in trouble. As I was admitted into the Dino team from the very beginning I want to thank all my teammates who accepted me and selflessly helped me in all my needs. Especially I want to thank Heimo Haub, who actually convinced me to start to work at the IICM and who spent a lot of time with me answering my questions. Furthermore I want to thank Didi Freismuth for helping me with many technical problems. I also want to thank all members of the IICM who helped me with all the organizational stuff. Special thanks to Sarah Cotterill for reading this thesis and for correcting my English. Finally I want to express my gratitude to my parents who supported me during all the years of my studies.

I hereby certify that the work reported in this thesis is my own and that work performed by others is appropriately cited.

Signature of the author:

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten Anderer unverändert oder mit Abänderungen entnommen wurde.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

When thinking about the future of Internet applications terms like collaborative work, knowledge-sharing and distance teaching are the most promising ones. There exists a variety of collaborative tools, or simply groupware, and Internet services like newsgroups which in combination allow locally separated users to carry out tasks together. The idea for this project was to develop an application which allows multiple users to share a network-accessible shared bookmark collection.

For brevity I will use the term "Shared Bookmarks" in place of "a shared bookmark collection which is accessible via the net".

The decision was made to use pure Java as programming language and to take the Dino system as a basis for this application. Dino is a middleware system providing access to a distributed object space. Through those Dino objects external systems like file systems or a database can be accessed at a high level of abstraction. A globally unique addressing scheme, a consistent link management and a rule based user-management are some of the main features of the Dino system. Among other things it shields the programmer from implementing different low-level network protocols which are used in a heterogeneous information system like it is in the Internet.

This thesis first gives an overview of collaborative tools and related Internet services like, for instance, mailing lists. After discussing the user requirements for a shared bookmark collection a brief description of the Dino system and its capabilities is given.

The advantages of building an application on a middleware system are figured out by discussing the techniques used in distributed object systems. The features which are important for Shared Bookmarks can be summarized as follows:

- Since bookmarks store the address of a document (e.g. a URL) the addressing mechanisms and link management play a central role.
- Applications running on distributed object systems call methods on remote objects. Therefore, frameworks which allow communication among distributed objects are discussed.
- Security considerations for network applications are presented as long as they are important in a multi-user environment.

Because Shared Bookmarks have been implemented in Java special features of the Java programming language are focused whenever possible.

1.1 Chapter Overview

- Chapter 2, *Background*: Gives an overview of collaborative tools and other related services like newsgroups or yet existing shared bookmark collections. The main features, standards and network protocols are discussed. Although some products are mentioned in this chapter it does not contain a product comparison.
- Chapter 3, *Requirements*: Outgoing from a use-case the user requirements for a shared bookmark collection will be shown. In the second part requirements concerning the administration and organization are presented.
- Chapter 4, *Dino*: This chapter gives an overview of the Dino system and its capabilities.
- Chapter 5, *Addressing and Link Management*: We will see the requirements for addressing mechanisms in a heterogeneous information system. It shows how documents and parts of documents can be addressed and how these addresses can be mapped onto objects. Based on the addressing mechanisms the requirements for an advanced link management are presented. Later in this chapter the XML addressing and linking specifications are discussed. At the end of this chapter we will have a look at the Dino addressing scheme and the Dino link management.
- Chapter 6, *Distributed Objects*: This chapter discusses frameworks used by distributed object systems. A short review at Remote Procedure Call (RPC) is followed by a closer look at the CORBA framework and Java RMI. Since directory services can be used to store and look up remote objects an overview of directory services is given.

- Chapter 7, *Security Considerations*: We will see security requirements for applications in heterogeneous information systems. A survey of cryptography is followed by a closer look at the Java security model. The role of signed Java applets, the Java security API and X.509 certificates are discussed.
- Chapter 8, *Implementation*: This chapter describes the implementation of Shared Bookmarks in general.
- Chapter 9, *Summary and Outlook*: The last chapter summarizes conclusions from this project and my future work on this project is presented.
- Appendix A describes the process of creating signed Java applets and certificates.

Chapter 2

Background

Computer Supported Collaborative Work (CSCW) is carried out using collaborative tools, or simply groupware. Since collaborative tools are often used in combination it is important to know what services these tools provide.

After a classification of the different tools the possible network architectures like client-server and peer-to-peer are discussed. Then we will examine the most popular collaborative tools and some other services.

Generally, a shared bookmark collection can be seen as a service such as newsgroups or mailing lists. In a newsgroup people exchange information concerning one specific subject which can be for example a particular programming language. The information users share in a Shared Bookmark collection is known as: "Where is a document located?"

2.1 Groupware

This section will define what collaboration is and how collaborative tools can be classified. Later a brief introduction to the technical requirements and social aspects of groupware is given.

Collaboration in a non-technical sense can be defined according to: [Bru91]

Collaboration is a process to reach goals that cannot be achieved acting singly (or, at a minimum, cannot be reached efficiently). As a process, collaboration is a means to an end, not an end in itself. The desired end is more comprehensive and appropriate services that improve family outcomes.

The targets of groupware can be summarized as follows:

Groupware should help a group of people to move towards internal col-

laboration with the goal of competing externally (e.g. competing with other companies). [SM96]

Groupware can classified in different manners:

If the users work in a synchronous or asynchronous fashion. In other words if the users have to access a service at the same time or not. Although multiple users may read articles in a newsgroup at the same time, it is not necessary that all users read the newsgroups at the same time. Therefore, newsgroups are an asynchronous service. Chat is a synchronous service since all participants are connected at the same time.[Wul97] Concurrency problems and synchronization of multiple communication channels are mainly problems of synchronous-used groupware.

Another possibility is to classify groupware by the environment in which they are running: Internet-based applications are used by people all over the world, whereas, for example, hi-end video-conferencing is mainly used in local area networks.

A more general approach to classify groupware is to distinguish *Workgroup Computing* and *Workflow Processing* . [SM96] Workgroup Computing is concerned with sharing documents and applications, whereas Workflow Processing deals with the process itself.

If two people work on the same document from their locally separated desktops then they share a workspace and the service is classified as Workgroup Computing. (see Figure 2.1) Examples for Workgroup Computing are multi-user editors or whiteboards.

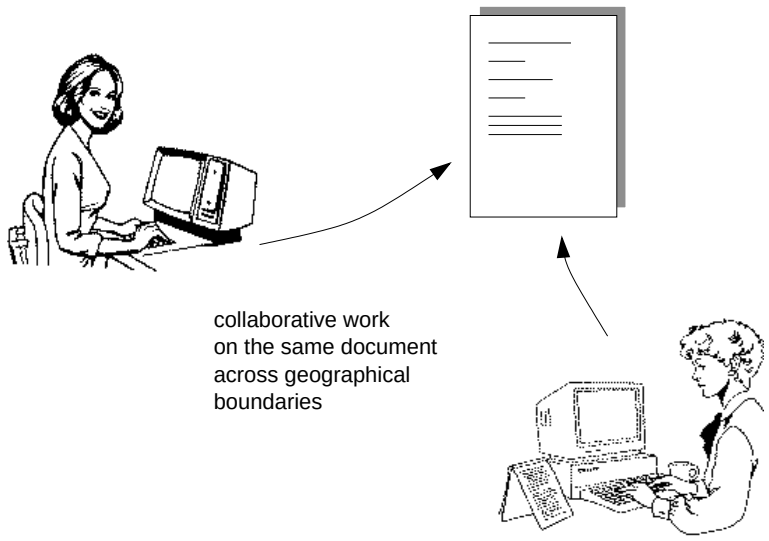

Figure 2.1: Example: Workgroup Computing

Workflow Processing tools control the process of a specific task. If, for instance, a group of people work consecutively on one single document then Workflow Processing controls to whom the document is sent next. (see Figure 2.2)

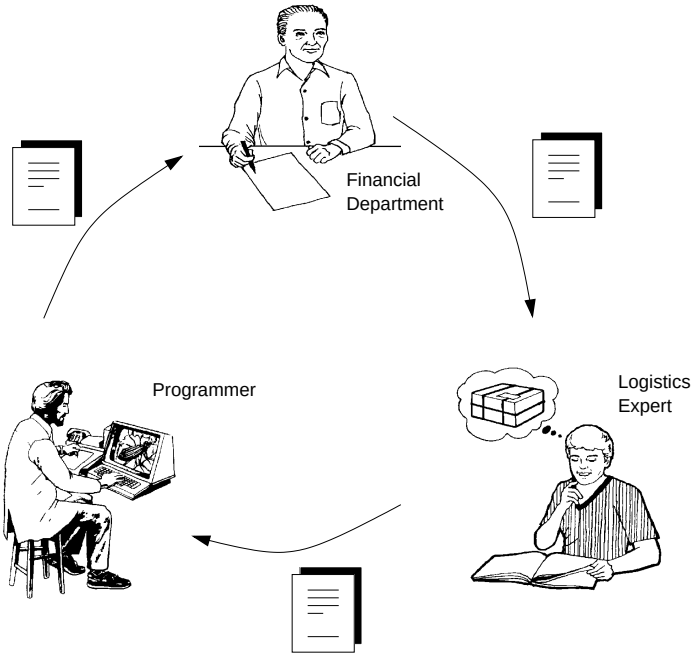

Figure 2.2: Example: Workflow Processing

Shared Bookmarks is part of Workgroup Computing because users share an information space.

Another point of view presents the "4C" classification . 4C stands for the four electronic collaboration functions: [Col97]

1. **Communication:** The participants communicate by sending messages to each other. The communication can be synchronous or asynchronous. Examples are chat and mailing lists.

2. **Coordination:** Tools like shared notebooks, shared calendars and flow-charts are used to coordinate tasks carried out by different people.

3. **Collaboration:** Is working with others for increased productivity by dividing a big task into smaller tasks and sharing them among a group of people.

4. **Corporate Memory:** Is recording and sharing both the tacit and explicit knowledge in an organization.

In this classification Shared Bookmarks is situated in Corporate Memory. Users share the tacit knowledge of where a document is located.

2.1.1 Parameters for Groupware Communication Protocols

Depending on the type of application one or more of the following parameters have to be taken into consideration when designing collaborative tools [Koc98] [SM96]:

- **Amount of Data:** The amount of data which collaborative tools exchange can vary from a few bytes per second (text chat) to several hundred kBit/s (hi-end video conferencing). According to the amount of data the network bandwidth and appropriate data representations has to be chosen. Internet-based applications mostly share one single connection with other applications. It is important to check whether the actual transfer-rate fulfills the requirements if multiple applications are running. If the amount of data is very low then we have to choose network protocols with little header-information.
- **Quality of Transmission:** If a tool plays a central role in a groupware session then the quality of transmission has to be high. As an example, in video-conferencing the video display has a supportive function, whereas the audio signal requires a high quality of transmission. Other services like file transfer for example require a maximum.
- **Timing and Synchronization Requirements:** For real-time applications we need short response times and for some applications multiple data-channels (e.g. audio and video-signal) have to be kept synchronous. For message-based applications the correct sequence of messages is a basic requirement.
- **Group Size:** Depending on the maximum number of simultaneous users appropriate hardware and network bandwidth have to be chosen.

2.1.2 Architectural Basis

Groupware runs on distributed systems, which means that the application runs on locally separated computers which are connected via a network. The most widely used architecture is a client-server relationship.

The server listens for incoming connections from clients. The server either mediates the client to other clients (e.g. a chat client connects to a server and enters a chat room) or assigns workspace and/or information

space to a client. Examples of a client-server architecture are news-servers and news-clients such as the Netscape Messenger.

Another architecture is a peer-to-peer relationship between clients. In a peer-to-peer architecture the clients are coequal participants and no additional server is necessary. [1] [Gre99]

How to realize a peer-to-peer relationship will be discussed in chapter 6.

Groupware may build up on a middleware system. A middleware system is a platform that provides various services like a user-management, security management, directory service and more. It is called middleware because it embeds external systems, like a file system or an Oracle database which means that these systems are "beyond" the middleware system. On the other hand it provides access to these services at a high degree of abstraction [SM96]. Applications which reside "above" the middleware system access the external systems through the services provided by this middleware layer.

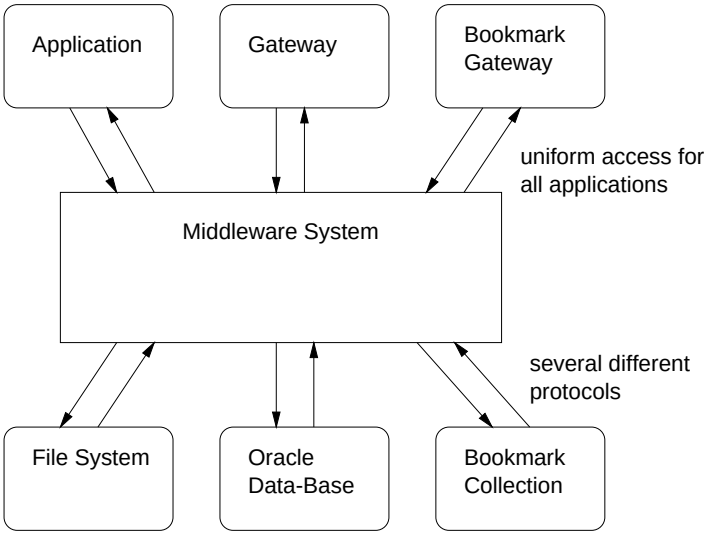

Figure 2.3: Middleware Architecture

Figure 2.3 shows the middleware architecture. Note that applications gain access to the external systems in a uniform fashion. In this example the bookmark collection could be stored in the database as well as in the file-system.

As already mentioned Shared Bookmark have been implemented using the Dino system, which for its part is a middleware system. For more details on Dino and middleware systems please see chapter 4. [dT99]

[1] Nevertheless one of the clients has to connect to the counterpart. Once the connection is established they are equivalent partners.

Groupware may run on multiple hardware and operating systems which requires platform independence. Java, for instance, allows to run applications on almost every operating system and shields the programmer from writing or compiling the same application for different platforms.

Furthermore, it is necessary to adhere to standards. [SM96] Today, not only the formal standards bodies, such as the American National Standards Institute (ANSI)[2] or the International Organization for Standardization (ISO)[3] are involved in standardization, but also smaller organizations like the World Wide Web Consortium (W3C)[4]. Smaller working groups are often more "streamlined" and new features are adopted more quickly. Although standards are subjects of change and many recommendations never get past the working draft stage they should be implemented whenever possible. [Gre99]

2.1.3 Limitations and Conflicts

Although groupware may be used for various purposes there are some limitations. It is, for example, unlikely that an interpersonal dispute can be resolved easier using audio-chat or e-mail. [SM96] In this case it is probably better to arrange a personal meeting. Furthermore resistance of the employees of a company may become a problem. If, for instance, groupware is used to manage large volumes of paper and documents then those whose jobs depend on that work may become "resistance fighters" against the new technology. [SM96]

When developing groupware the designers and programmers should be aware of some groupware-specific conflicts [Wul97]:

1. Users may have problems with the division of labour and the coordination of separately carried out subtasks.

2. Conflicts may result if an employee's position depends on his knowledge about a certain topic and he now has to share this knowledge with others.

3. If resources are limited the question "Who may work with whom at what time using how much resources?" has to be regimented.

[2] ANSI: `http://www.ansi.org`
[3] ISO: `http://www.iso.ch/`
[4] W3C: `http://www.w3c.org`

2.2 Collaborative Tools

In this section the main features of the most common collaborative tools are discussed.

2.2.1 Text Chat

Text Chat is a kind of instant e-mail service that allows users to exchange text messages with other people. It generally needs few resources in comparison to other collaborative tools since the amount of data exchanged are just a few characters. [Sam95] Chat was mainly used by novice users. Today protocoled chat services exist and more experts make use of it.

Most chat applications use a client-server architecture in order to provide so-called "chat-rooms" or "channels". A chat-room has a certain topic and users log-in and take part in the conversation that is going on. Users may create[5] their own chat-rooms and talk in groups or privately.

Internet Relay Chat (IRC) is such a client-server based chat protocol. [Sam95] Several servers[6] all over the world provide a large number of chat-rooms. IRC-clients exist for different platforms (Unix, Mac, Windows). Chat-rooms can be found on Web-pages and special IRC-search-engines[7] look for chat-rooms[8] on a specific topic. It supports file-transfer and it has many games you can play. There are an enormous number of users and occasional rogue operators. Although it is not well managed, IRC remains useful.

Network applications like whiteboards or network games provide built-in text-chat functionality, in order to help users to coordinate their tasks. (e.g. to explain a drawing or to agree upon the next game-session)

2.2.2 Whiteboard

A Whiteboard allows locally separated people to paint drawings on a virtual black-board. Each participant sees that whiteboard on his screen and may either view what the others paint or add his own drawings.

Whiteboards are synchronous tools. Concurrency problems occur if more than one user is able to draw on the board at the same time. Therefore, locking mechanisms together with dead-lock-aware algorithms have to be applied.

[5]Create your own IRC chat room: `http://www.talkcity.com/irc/apply.htmpl`

[6]IRC FAQ: `http://www.mirc.co.uk/ircintro.html`

[7]IRC search engine: `http://www.mirc.co.uk/irc.html`

[8]List of IRC channels: `http://www.liszt.com/chat/`

In order to keep the amount of data low it is important to find an appropriate representation for the different graphical objects.

Some related links to whiteboards:

- A Java whiteboard: `http://groupboard.com/`
- A tutorial for writing a whiteboard in Java `http://www.internetindia.com/wwwboard/`

2.2.3 Audio Conferencing

Audio Conferencing, or audio-chat, allows two or more people to talk to each other as if they would by telephone. Audio chat requires a microphone, a sound-card which converts the analog signal into a digital signal, speakers and an elevated network-bandwidth.

In order to get an idea of the required bandwidth let us consider the following calculation for telephone circuits [SSF97] Telephone equipment is designed for voice transmission of 3000 Hz (3400 Hz in Austria). According to the sample-theorem the minimal sampling rate is:

$$f \geq 2 * f_{max} = 2 * 3000 = 6kHz. \tag{2.1}$$

The quality of transmission increases if the signal is oversampled. Nowadays it is standard to sample voice at 8kHz.

On the other hand we have to answer the question: "How many levels should be quantified?" In other words this means how many bits describe one single sampling point. The accepted standard is 8 bits per sample point. With

$$2^8 = 256 \tag{2.2}$$

and taking in account that 1 bit is used to sign then the amplitude measurement is 1/128. 128 bit is more precise than analog telephones. The bit-rate per channel is as follows:

$$Bit\ rate\ in\ channel = (bits\ per\ sample) \times (sampling\ rate) \tag{2.3}$$

$$B = (8) \times (8000) = 64kbit/s \tag{2.4}$$

That is why most equipment (e.g. ISDN) is designed to handle signals which are multiples of 64 kBit/s.

Nevertheless, most Internet-based applications work with lower transfer-rates since these applications have to share one connection (e.g. one single ISDN channel) with other applications and services. VocalTec's IPhone 5[9] for example requires at least a 14.4 kbps modem connection - they recommend a 28.8 kbps modem.

Depending on the area of inset appropriate audio formats and compression algorithms have to be chosen. The following link is a summary of audio-encoding formats together with a detailed description and sample files.

`http://www.cs.columbia.edu/~hgs/audio/`

Please see the next section for audio-conferencing standards.

2.2.4 Video Conferencing

In video conferences the local separation is bridged by audio and video transmission.

Video conferencing requires an elevated technical infrastructure: In addition to the requirements of audio-conferencing a video camera is necessary. Internet Video Conferencing (IVC) differs from hi-end video-conferences. Hi-end Video Conferences use reserved data-transfer-connections and require data-rates of several hundred kBit/s. Lo-end Internet video conference clients like NetMeeting or CU-See-Me require at least 10 kBit/s. (see subsection 10). Systems that use special audio- and video-cards which encode and decode the audio and video signals reach a throughput of 200-300 kBit/s. Lo-end clients are generally designed for two participants and the quality of the audio signal, the video resolution and the frame rate are lower. [Zue99] The audio signal generally plays the central role in Video Conferences while video normally has a supportive function.

Multimedia Teleconferencing Standards

Table 10 shows the standards by the International Telecommunication Union (ITU-T)[10] which comprise the core technologies for multimedia teleconferencing where the T.120 is the basis for the three H.xxx standards.

[9] IPhone 5: `http://www.vocaltec.com/products/iphone5/features.htm`

[10] ITU-T: `http://www.itu.int/ITU-T/`

Standard	Description
T.120	Real-Time Audio Data Conferencing
H.320	ISDN Video Conferencing
H.323	Audio-Visual Communication in local area networks
H.324	Audio and Video communications over low bit-rate connections

Table 2.1: Multimedia Teleconferencing Standards

Internet-Based Video-Conferencing

CU-See-Me[11] developed by Cornell University has taken the lead in Internet-base video-conferencing. It requires an ISDN connection and has many additional tools such as text-chat and a whiteboard. Other products are Picturetel[12] which offers the highest quality by special (costly) hardware, Intel's Pro Share[13] being a lo-end product and Microsoft's NetMeeting[14].

2.2.5 Other

Apart from the described tools there exists a large number of other collaborative applications. As we have seen, applications like NetMeeting or CU-See-Me come along with various integrated collaborative tools. Additionally they support file-transfer, application sharing (e.g. a multi-user editor) and more. Bulletin boards are used for forums similar to newsgroups.[15] For Unix the Virtual Network Computing X-Server allows multiple users to connect to one single X-server and so work on the same desktop, in order to carry out tasks together. All applications started on this X-server can be viewed and used by all participants.[16]

The following sections describe services and tools which are somehow related to Shared Bookmarks.

2.3 Mailing Lists

Mailing lists are discussion forums based on the Simple Mail Transfer Protocol (SMTP).

[11] CU-See-Me: `http://www.cuseeme.com/`
[12] PictureTel: `http://www.picturetel.com/`
[13] Intel Pro Share: `http://www.intel.com`
[14] NetMeeting: `http://www.microsoft.com/windows/netmeeting/`
[15] Ultimate Bulletion Board: `http://www.ultimatebb.com/`
[16] VNC Server: `http://www.uk.research.att.com/vnc/xvnc.html`

The principle of mailing lists has been introduced at the beginning of the eighties in the BITNET. In order to administrate mailing lists automatically ListServe has been developed. ListServe supports searching inside the mailing list, subscribing and the administration of the mailing list.[Kya94]

Generally mailing lists have two distinct e-mail addresses. The first is used for administration (e.g. to subscribe/unsubscribe). This forum itself is associated with the second e-mail address. The mail server automatically sends incoming mail to all members of the mailing list.[Kya94]

A similar mechanism can be used in Shared Bookmarks if someone wants to become a member of a specific bookmark folder.

The following e-mail header would subscribe myself to the dino.development mailing list on the dinopolis server:

```
to:         mail-admin@dinopolis.org
from:       pzamb@iicm.edu
begin
SUBSCRIBE   dino.development  pzamb@iicm.edu
end
```

One problem of mailing lists is the large number of incoming e-mails if someone has subscribed to different mailing lists. Some mailing lists send several e-mails a day and the amount can be tremendous. [Kya94]

Mailing lists are often used to answer questions by users. The Frequently Asked Questions (FAQs) are often mirrored on news-servers or in the WWW. [Kya94]

Since e-mails are sent un-encrypted across the net most e-mail clients support encryption tools like Pretty Good Privacy (PGP) for security reasons. (see also chapter 7)

Sun offers an extensional package called javax.mail which is an API for e-mail messaging in Java. Shared Bookmarks use a kind of ListServer which allows users to subscribe to certain folders of the bookmark collection. Once subscribed users get a notification if, for example, new bookmarks have been added to the collection. Therefore a summary is computed which is sent to the users from time to time. This user notification is done using an adapted ListServer implementation in Java. See section 8.9 for more details on the Java Mail API and the implementation in Shared Bookmarks.

Group	Subject
comp	Computer related stuff
misc	Miscellaneous
alt	Alternative
news	The News-Service itself
rec	Hobbies, Spare Time (recreational)
sci	Science
soc	Society
talk	Discussions

Table 2.2: Newsgroup Naming Conventions

2.4 Newsgroups

Usenet or newsgroup servers exchange electronic mail tagged with predetermined subject headers using the Network News Transfer Protocol (NNTP) which is similar to the SMTP. The e-mail is referred to as an article and the subjects are the newsgroups. [Sam95] It is possible to annotate messages to an existing article. All annotations and the article itself form so-called threads. It is possible to cross-post articles which means that one article is posted to a number of newsgroups simultaneously. [Sam95]

Usenet newsgroups are named using a set of conventions. Some of the major groups are listed in table 2.4 [Sam95] [SBGK94]:

Today many Newsgroup-providers publish so-called Magazines (Digests) of their Newsgroups. The best and most informative articles are regularly picked out and published. These magazines are often posted as HTML pages on the WWW. Some newsgroups are entirely mirrored on the Web so that the newsgroups can be accessed using a simple browser and WWW based search engines are able to search in newsgroups as well. [Kya94] (as an example see Remarq [17])

Newsgroup policies, also called Netiquette, declare the rules for using a specific news server. These policies contain guidelines for the users like the preferred language, general rules for postings and spamming rules.

[17] Remarq: `http://www.remarq.com`

2.5 Local Bookmark Collections

Local Bookmark Collections like Netscape Navigator's Bookmarks and Microsoft Internet Explorer Favorites are hierarchically organized and store the following data:

- The **title** of a bookmark corresponds to the title of the document it refers to (e.g. the title of the HTML page)
- The **location** which is the URL of the document.
- An additional **description**.
- The date when it was **last visited**.
- The date when it has been **added**.

In the Netscape Navigator it is possible to make an alias to a bookmark which means that a copy is made from that bookmark and the alias can be put into another folder. If the original bookmark changes the alias changes too.

Another feature is the that a new bookmark can be filed in a folder. In the Netscape Navigator it is not possible to create a new folder while adding a bookmark.

The structure of the collection can be organized by adding folders and separators. It is possible to search for one of the first three above properties and the bookmarks can be sorted by different criteria.

It is possible to export and import bookmark collections and there exists various utilities[18] to convert bookmarks from one format to the other and to merge bookmark collections.

Netscape stores the bookmarks in HTML format while Microsoft Internet Explorer stores the bookmarks as concatenations.

These capabilities can be seen as additional user-requirements for Shared Bookmarks since most users are familiar with one of the two browsers.

2.6 Shared Bookmark Collections

Sites like Oneview[19], Coolsync[20] or Clickmarks[21] allow users to upload their local bookmark collections (e.g. exported from Netscape Navigator) to their

[18] Bookmark Converter: `http://tucows.fh-reutlingen.de/bookmark95.html`

[19] Oneview: `http://www.oneview.com/`

[20] Coolsync: `http://bookmarks.coolsync.com/`

[21] Clickmarks: `http://clickmarks.com/gatra/about.html`

servers. The users can make parts of their collections public. Oneview allows users to add bookmarks during browsing. Therefore users have to add the following bookmark to their local bookmark collection:

```
javascript:QuickAdd="http://www.oneview.com/quickadd.phtml?
  url="+document.URL+"&
  title="+escape(document.title);
  window.open(QuickAdd,"ovquickadd";
```

When users want to add a bookmark to the Oneview server they click this local bookmark and the URL of the currently displayed page is sent to a perl script which sends back a HTML form containing the URL and a file dialog to choose the destination folder.

The bookmark collection is represented using HTML tables and JavaScript. The problem is that every time a folder is opened the browser reloads the whole page.

Coolsync provides more or less the same functionality as Oneview. Additionally it is possible to download and install a Windows 9X client application on the local machine. To share bookmarks with others the user may create so-called Circles which are public folders in the collection.

The following URLs refer to Shared Bookmarks projects currently in progress:

- Knowledge Pump is a project that provides a Shared Bookmark collection and a recommender system on top of a WWW browser:
  `http://www.rxrc.xerox.com/research/ct/research/kp/factsheet/kp-factsheet.html`
- Shared Active Bookmarks is part of a project called GROOVE which is an active information service:
  `http://visinfo.zib.de/zb2/`
- A client/server knowledge store which helps sharing information about WWW sites:
  `http://www.ics.uci.edu/~redmiles/ics125-FQ98/projects/bookmarks.html`

Chapter 3

Requirements

The following use-case illustrates how Shared Bookmarks can be used. By analyzing this use-case the user-requirements are discussed in the later sections of this chapter. Additionally, possibilities how each single requirement can be fulfilled are discussed briefly, since in some cases there exist multiple solutions.

3.1 Use Case

A Shared Bookmark collection consists of folders and bookmarks. Each user has access rights to a subset of folders. There are private folders, public ones and folders which can be accessed by a group of people. Now let us see how users might use Shared Bookmarks:

A user searches for some files in the local network using a the HotJava browser and adds bookmarks to a Shared Bookmarks folder which is shared with his/her teammates. For each bookmark a description and keywords are added.

The user browses the WWW using Netscape Navigator and searches for the topic "soundcards". Interesting sites are added to a private folder.

Later a product comparison is written. Outstanding products are picked out and added to a new folder called "Sound Cards". This folder is moved into a folder called "Multimedia" which can be accessed by members of the department. Furthermore, a bookmark is added which refers to the document that contains the product comparison. The product comparison resides on the local file-system. Each product receives a rating (e.g. from 0 to 10) according to the quality of the product.

A colleague is interested in sound cards and wants to have access rights to that folder. After contacting the administrator the colleague is added to the appropriate group and from now on receives a notification if new bookmarks are added to this folder.

The colleague evaluates the different products and the product comparison. Each bookmark is assigned a valuation (e.g. form 0 to 10) according to the quality of the bookmark.

A new internal price-list has been issued. Price-lists are stored in an Oracle database. A bookmark is added to an existing folder called "Price Lists". The price list is marked as "new".

The administrator wants to delete old bookmarks and dead links from the collection. Additionally a backup is made every night at 11 p.m. and the content of frequently accessed bookmarks is mirrored on a local server.

3.2 User Requirements

Platform Independence and GUI

Shared Bookmarks have to be platform independent because it runs on multiple operating systems. Therefore, Java has been chosen as a programming language.

The common way to visualize bookmark collections is a tree representation with hierarchically organized folders and bookmarks.

Other Shared Bookmark collections display the tree within a HTML page. Oneview[1] for example uses JavaScript and HTML tags (hundreds of tables) to represent that tree. A problem of this method is that every time a user opens a folder the whole HTML page is reloaded.

Another possibility is a client application. Users download and install a small program on their local machine. (see 2.6) This is advantageous since the graphical user interface remains on the local computer and only data (the bookmarks) are sent across the net. A more powerful programming language can be used (e.g. C++) to represent the bookmark tree for a better performance. It is obvious that if the user logs-in from another location he has to install the client again.

The advantage of Java is that a program can be run as applet and as application, this means that depending on the circumstances one can install the client on the local machine or access the bookmark collection using a browser. Since Java applets are subject for strong security restrictions signed

[1] Oneview: `http://www.oneview.com/`

applets have to be used. See chapter 7 for more information about signed Java applets. For more details about the implementation of the GUI see chapter 8.

Performance

In order to achieve good response-times network-traffic, should be kept low. As already mentioned, even the GUI of the bookmark client can be sent across the net. In this case the GUI should be as slim as possible.

Meta Data

Folders and bookmarks store a set of meta-data like keywords or a valuation for a bookmark. Each meta-datum is edited either by the user, by the application or by the administrator.

When a user adds a bookmark he assigns a rating. He thereby specifies for his part how important (or how good) he thinks that a bookmark is.

Valuation on the other hand is done by other users who view (or open) the bookmark. When a bookmark is added a standard value is assigned - let us say 5. Every time a user opens a bookmark he is asked for a valuation. He has to enter a value from 0 to 10. The average is computed from all values. Correspondent filters display just those bookmarks which have a high (or low) ranking (e.g. greater than 7).

See chapter 8 for more details about meta-data.

Filters, Searching and Sorting

Various filters have to be provided. (e.g. a filter that shows only new bookmarks) Bookmarks within a folder can be sorted by different criteria. It should be possible to search for bookmarks by title, keywords, author etc. (for a full list of filters see chapter 8)

Subscribing and User Notification

The user wants to get notified via e-mail if, for example, new bookmarks have been added to the collection. The user specifies the notification-interval. If a user wants to get access to a folder he sends a request to the administrator to subscribe to that folder.

Link Consistency

The user does not have to worry about dead or inconsistent links. Shared Bookmarks have to detect those links and delete such bookmarks. See chapter 5 for more details on addressing and link-management.

Multi-Protocol

It has to be possible to bookmark documents on file-systems, in the WWW, in databases and so on. The Dino system "speaks" various protocols and therefore allows access to multiple external systems. (see chapter 4 for more details.

Administration and User Management

Administrators maintain the bookmark collection. Although users may edit and delete bookmarks in their private folders the shared sections are maintained by administrators and sub-administrators. A Shared Bookmarks policy, familiar from news-groups, will help to avoid too much trash. Nevertheless some tools running in the background have to be provided, which for instance, sort out dead links and obsolete bookmarks. See chapter 8 section for a list of tools. The administrator specifies at what time each maintenance-tool is run.

An explorer with integrated maintenance tools should be provided. From within this explorer the administrator should be able to access the user manager in order to add users or change the users' rights.

The User Management has to support users and groups. Users have a user-name and a password and pertain to a group. Each user has access rights to a subset of folders of the whole bookmark collection. Figure 3.1 shows a typical bookmark-tree and the folders of two users. The dark folders may only be viewed by user A, which means that these folders are his private folders. The bright ones are user B's private folders. The two users pertain to the same group and the two-colored folders allow members of that group to access these folders. On the right hand side you can see what parts of the collection each user sees.

Note that this is a special requirement to the user manager, since allowing access to a subdirectory of a inaccessible parent directory is not supported by every access control mechanism. (e.g. Unix file systems do not support it) See chapter 4 for more details about the Dino user management.

Additionally the user's e-mail address has to be stored in the user record. It is used to notify the user if, for instance, new bookmarks have been added

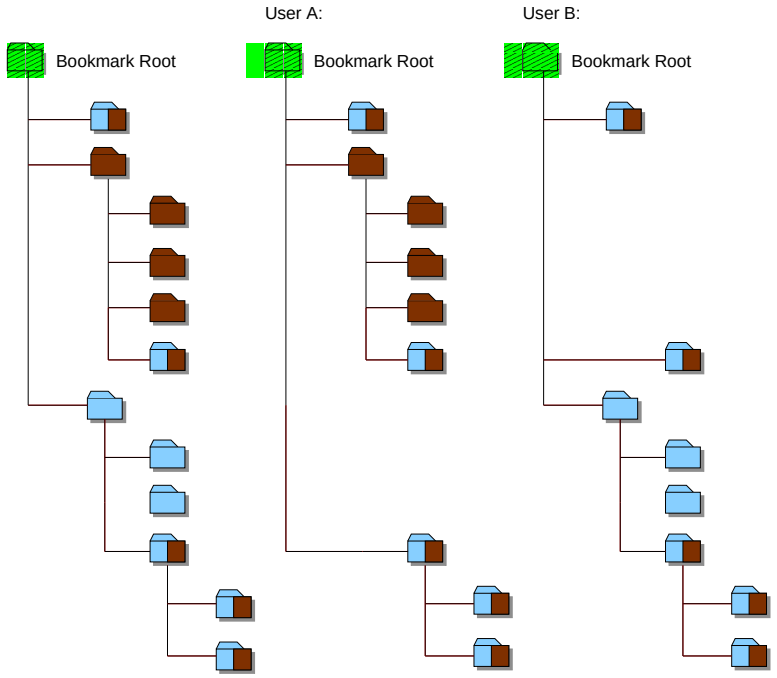

Figure 3.1: Folders in a Bookmark Collection

to the collection.

Security

Since Shared Bookmarks may contain informations that have to be protected (see chapter 7) appropriate security mechanisms have to be provided. Dino supports secure network connections (e.g. SSL). Authentication and privacy should be supported on different levels. See chapter 7 for more details about security considerations.

Logging and Backup and Caching

The users' actions have to be logged on different levels: First a facility to log all users' actions in the Shared Bookmarks collection has to be provided. This is necessary to control what users do and to catch out abusers.

On the other hand some user actions are stored with the bookmarks. If for example a bookmark is opened a counter is incremented that counts all hits to that bookmark. This information again can be used to provide special filters which show just bookmarks that have been visited very often.

If bookmarks are accessed very frequently then the document behind that bookmark should be loaded to a local cache. This cache has to be kept synchronous with the original document.

A Backup facility should be provided which stores (e.g. every week) the current state of the Shared Bookmarks collection.

The Dino system provides logging mechanisms, backup facilities and mirroring.

Interface to other Applications

There has to be a possibility of adding bookmarks to the collection from within other applications. Therefore an interface has to be provided so that other applications can add bookmarks.

Chapter 4

Dino

As already mentioned Shared Bookmarks is implemented using the Dino (**D**istributed **I**nteractive **N**etwork **O**bjects) System. This chapter gives a brief introduction into the goals of Dino and its capabilities. Later the Dino layer model is explained.

4.1 Goals

The Dino project at the IICM began in 1996 and the first version was a message-broadcasting system which has evolved over the years. In version 2 a hierarchical addressing scheme was added. Version 3 was more powerful but some glitches taken along from the previous versions led to version 4 - a redesign of the whole system.

Dino is a middleware system providing access to a heterogeneous information space at a high degree of abstraction and transparency. That information space is formed by underlying external systems like file-systems or databases. The goal of Dino is to support all functionality provided by each single external system and provide a super-set of all capabilities to the user. Moreover, a hierarchical addressing scheme and a powerful link management have been added. Security and user management, versioning and transactions are supported and mapped to the underlying systems whenever possible. In other words the Dino system adds features of one system to those which do not support this capability and provides this overall functionality to the outside world. [Hau99] [dT99]

4.2 Capabilities of the Dino System

The main features of Dino can be summed up as follows: [Hau99] [dT99]

Platform Independence

Since Dino is written in Java it runs on various platforms and operating systems.

Distributed Objects

Dino objects are spread over several Dino systems and represent either passive data or active entities and can be accessed at a high level of transparency and abstraction. In other words several Dino systems can be connected and then form a cluster that allows transparent access to all objects in all connected Dino systems.

The technologies used for distributed objects are discussed in chapter 6. In the last section of that chapter one can find a description of the Dino-Dino protocol which is used to connect Dino systems.

Addressing vs Navigation

Navigation in the structure of a Dino system is completely decoupled from the internal addressing scheme. Even non-hierarchical structures (e.g. graphs) can be represented by powerful linking mechanisms.

Since addressing and link management is important for Shared Bookmarks these topics are discussed in detail in chapter 5. In the last section of this chapter you find a more detailed description of the Dino addressing mechanisms and the Dino link management.

Customization

Dino allows the user to compose configurations which for their part may consist of several individual external systems. Customizing can be done at runtime so that no restarting or recompiling is necessary.

Representation

By providing so-called Views it is possible to represent Dino objects to the outside world. A specific Node is part of the Dino API and pertains to the

corresponding View. So-called Relation objects, on the other hand, are used to represent the relationships among objects in an appropriate manner.

Extensibility

Dino allows one to extend the functionality of the whole system by embedding new external systems and and by providing new Views as applications evolve. Even the internals are designed in a way that new technologies can be adopted and the functionality of the system can be extended when desired.

Gateways

Gateways allow other systems to access objects within the Dino System. (e.g. a browser may access objects via a HTTP gateway)

Security

On the object level all operations coming from outside the Dino kernel are security-checked. On the other hand, secure network connections and document signing is supported. For more details about security considerations please see chapter 7.

User Management

An advanced, rule-based user management allows to define complex access rights. Again, the access rights are mapped to the a external system's user management whenever possible.

Content

The data behind each Dino object is called "Content". Content concludes the document's data and meta information like attributes or properties. Users may ask a Node object for its content and receive either a data-stream which means that the application itself parses and interprets the content. Another possibility is that the content can be provided in an appropriate object representation. The internal object representation adheres to the Document Object Model (DOM) and allows to map the internal structure of documents to a corresponding hierarchical object structure.

It is possible to store documents according to the Extensible Markup Language (XML) specification. Appropriate parsers (e.g. the SAX parser by Sun Microsystems) allow us to directly convert those documents to the internal DOM representation. On the other hand content may be provided to applications in an XML representation even if a document is not stored in such a format in the external system.

4.3 Dino Layer Model

In order to obtain modularity and to avoid side-effects the Dino System is subdivided in 8 layers. Each layer is assigned certain responsibilities. The main goal was to avoid one layer dealing with responsibilities assigned to another.

Each layer contains a set of modules and objects. Figure 4.1 shows the Dino layer model. On the right side you can see the border between kernel and View. On the left hand side you see the four basic objects in the Dino system. Each object has a well defined interface and adds the functionality it is responsible for. The **Node** object is the access-point for applications to objects in the Dino system. Nodes pertain to a specific View and contain specific navigation and representation features. A Node object encapsulates one single **ADIOS**: (**A**ddressable **DI**no **O**bject **S**tub) which is the first internal object. It handles high level filtering and security checks are carried out in this object. An ADIOS encapsulates one single **INOS**: (**IN**ternal **O**bject **S**tub) which is the first kernel object. It combines various **EXOS**es: (**EX**ternal **O**bject **S**tub) which for their part represent the interface to external objects and provide access to entities of the external systems. [dT99] [Hau99]

For more details on the responsibilities of each object please see the Dino Software Engineering Document. [dT99]

Now let us have a closer look at each layer [dT99]:

The responsibilities of each layer can be summarized as follows:

- **Application Layer:** In this layer reside applications called "Dinosaurs". These applications access the Dino system via Node objects which are part of the Dino API. An example for a Dinosaur is a HTTP gateway which allows to access objects in the Dino system with a browser. Since applications access the Dino system via Nodes they cannot access the system internals.
- **API Layer:** The Dino API consists of one single class `Dino` which supports the following:

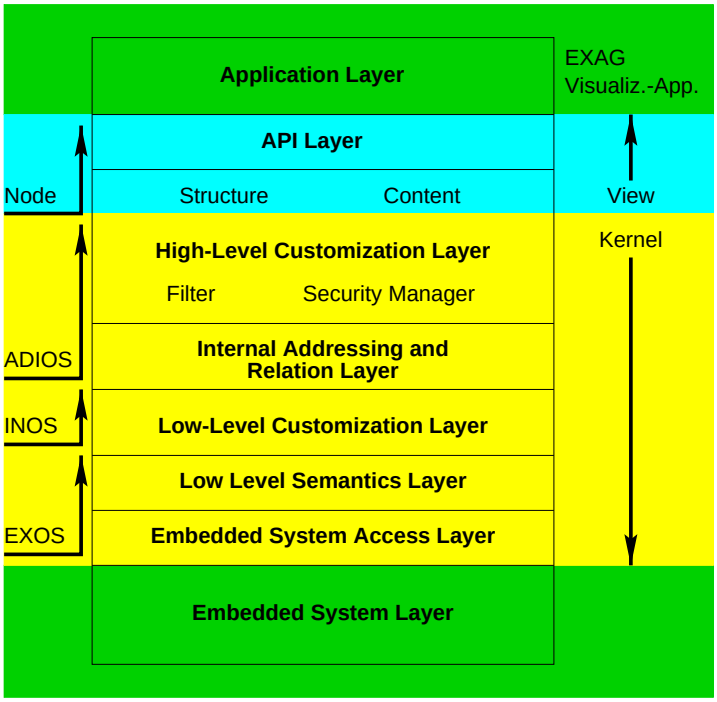

Figure 4.1: The Dino Layer Model

  - request a View
  - embed external systems
  - create configurations of several external systems
  - structure manipulation (e.g. move, copy, etc. . .)

- **High Level Customization Layer:** This layer separates the API layer from the Kernel since security-checks are done in this layer. Security Manager, high-level filters, internal handle management, system bootstrapping as well as Dino kernel configuration management are the main modules in this layer.

- **Internal Addressing and Relation Layer:** Objects in Dino are addressed using globally unique handles. This layer takes the corresponding relative handles from the underlying INOS objects and makes them globally unique by adding system specific identifiers.

- **Low Level Customization Layer:** In this layer a combination of external systems can be configured. The INOS object resides in this layer and provides a functionality super-set of all external systems

contained in its configuration. The user does not have to worry about which external system is responsible for what functionality.

- **Low Level Semantics Layer:** Is responsible for mapping the semantics of the external systems into the Dino world. As an example consider the semantics of a hierarchically structured file-system where the directory structure is mapped into appropriate parent-children relations in the Dino system.
- **Embedded System Access Layer:** This layer is responsible for communicating and exchanging data with external systems using low-level protocols.
- **Embedded System Layer:** This is the layer where all external systems like file-systems and databases reside and it is therefore out of the scope of the Dino world.

For more details please see the Dino Software Engineering Document [dT99].

Chapter 5

Addressing and Link Management

The main task of a bookmark collection is to store links to documents. Netscape Navigator for example stores bookmarks as hyper-links. The target of a hyper-link is a resource which resides anywhere in the underlying information system (e.g. the WWW). A resource is identified by a Uniform Resource Locator (URL). Depending on the underlying system, the resources can be documents, entries or queries in a database, services and so on.

After some considerations about heterogeneous information systems we will discuss names and URLs as they are currently used in the Internet. We will examine the capabilities of URLs and figure out the problems resulting for applications which have to deal with multiple protocols in a heterogeneous information space. Later we will see a model which allows access to a heterogeneous information space through one single, extensible interface. A description of an advanced link management is followed by a final section which illustrates how addressing and linking is done in the Dino system.

5.1 Heterogeneous Information Systems

Every information system has a model or architecture which is determined by its protocols. This architecture influences the accessibility of information for clients. [DS95]

Current information systems are heterogeneous what means that they are a compound of multiple applications and network protocols. The advan-

tage a heterogeneous information system is that distributed systems allow a high degree of parallelism and distributed capabilities increase the value of information.

If Shared Bookmarks is used in a heterogeneous information system like the Internet, then we have to deal with different resource locators at the same time. Resource locators refer to entities coming from physically and administratively distributed locations. "Physically distributed" means that a resource can be located at different servers or just at another location in the host file system. [DS95] "Administratively distributed" means that resources are accessed using different protocols. It is also possible that one single document is accessed using different protocols.

Figure 5.1 shows such a heterogeneous information system and a bookmark collection storing bookmarks (e.g. hyper-links) which refer to entities in physically and administratively distributed domains.

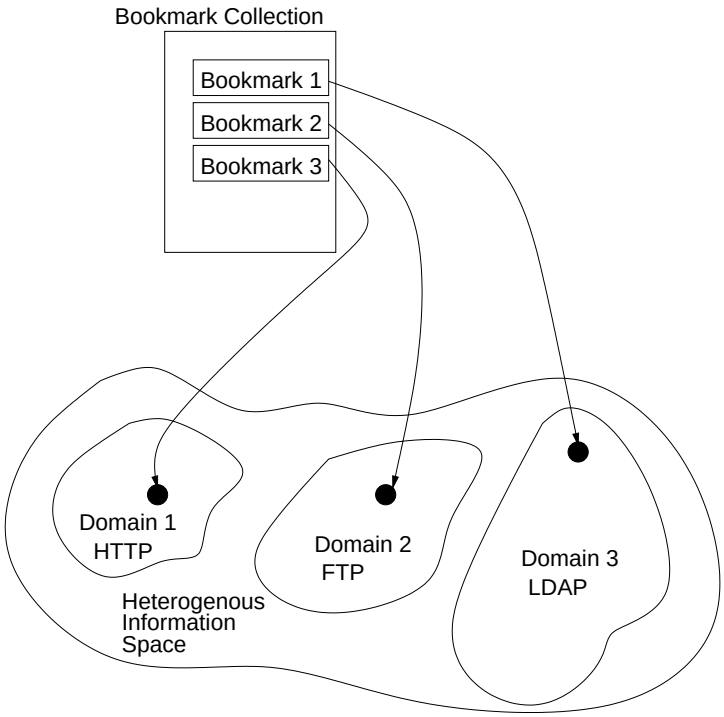

Figure 5.1: Shared Bookmarks in a Heterogeneous Information System

At the same time, as technology evolves applications and/or network protocols change which means that we will be confronted with new protocols and new applications making use of this heterogeneous information system.

On the other hand information systems should support advanced linking since linking allows us to represent relationships among the entities stored in the information system. Relationships are additional information which can be made accessible to applications.

Now let us examine how resources can be located and accessed.

5.2 Names

Names are often used to identify resources without accessing them. If the same name cannot be assigned to more than one resource then if two names are equal they refer to the same resource. On the other hand if one resource is always referred to by the same name resources can be distinguished by comparing their names. [DS95]

Names are also used to describe how the information can be accessed. URLs, as they are used in the WWW describe the location and the the protocol which is used to access the object. [Moc87]

For example `http://www.iicm.edu` describes the location "www.iicm.edu" and expects an application to use the HTTP protocol to access it. [DS95] (see 5.3)

On the other hand names have a descriptive function. A name can be human readable or it may include information used by applications. [DS95] Note that sometimes it is not desired that, for example, humans can draw conclusions from names because of security considerations. [dT99] In other words sometimes names underlie other restrictions and human readability cannot be guaranteed so that human friendly representation should be provided with translations to globally unique identifiers used by applications and/or the underlying protocols.

In any case names should address resources but should not contain other informations about that object what means that additional information should be placed into separate objects which hold all kind of meta-information. [1]

[1] The proposals of the Network Working Group [Kun95] describe globally unique, long-lived identifiers and additionally Uniform Resource Characteristics (URC) which are potential containers for any other information about a resource such as language information, representations an so on.

Scheme	Protocol Description
ftp	File Transfer protocol
http	Hypertext Transfer Protocol
mailto	Electronic mail address
news	USENET news
nntp	USENET news using NNTP access
telnet	Reference to interactive sessions
file	Host-specific file names

Table 5.1: Common URL Protocol Schemes

5.3 Uniform Resource Locators

The common way to address resources in the Internet are Uniform Resource Locators (URLs). An URL contains the location of a resource and the protocol to access it - also called schemes. Table 5.3 shows some common protocols [BLMM94]:

Today there exist more than 80 different schemes[2] and the generic syntax for URLs provides a framework for new schemes when new protocols are developed.

The following URL additionally contains a username, a password the port where the server is running and the path where the document is located on the host's file system [BLMM94].

```
ftp://pzamb:my_password@www.dinopolis.org:5345/index.html
```

When an application wants to access this resource it has to know about the protocol. Generally applications deal with a sub-set of these various protocols. Netscape Navigator for example is able to access http, ftp, file and some more.

We remember that one user requirement was that Shared Bookmarks should load frequently accessed documents to a local cache and that another was to test the integrity of a document. To fulfill these requirements we have to "speak" the appropriate protocol. When using Shared Bookmarks in a heterogeneous information system we probably have to deal with a variety of protocols.

[2] Addressing schemes: `http://www.w3.org/Addressing/schemes`

Applications in a heterogeneous information system are automatically confronted with multiple protocols and the fact that new protocols will be added in the future, and that existing protocols will change, makes that problem even worse.

One problem with currently used hyper-links is that if a resource is not available (e.g. the resource has been deleted) the URL is almost worthless. The hyper-link containing that URL is "dead" and the information provided by the bookmark collection is useless. [BLMM94]

As you could see the functionality and the information content stored in a bookmark collection depends heavily on the addressing and linking mechanisms of the underlying system.

In the next section a model for a heterogeneous information system is introduced which allows applications to access information on a high level of abstraction without running into these side-effects.

5.4 A Model for a Heterogeneous Information System

This model is partially taken from [DS95].

5.4.1 Homogeneity

The information system should allow network-based applications to access information that is physically and administratively distributed across the net. It should provide a single interface for identification, location and access of information for applications.

Figure 5.2 shows that in this model applications use one single interface to locate and access information inside the heterogeneous information system.

5.4.2 Heterogeneity

The information system should be flexible enough to support new network-protocols and should be able to provide services to evolving applications as far as they expect them. Applications should be able to take advantages of increased functionality of network protocols only when desired.

In figure 5.2 you can see that each single administrative domain is connected via its own interface. If protocols change or new protocols are added,

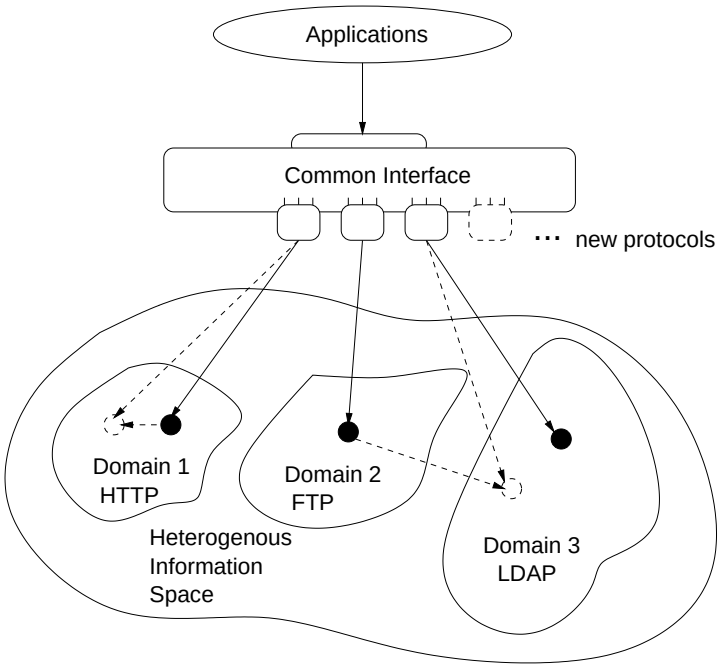

Figure 5.2: Access through a Common Interface

new interfaces can be applied to support the new domains. Applications on the other hand are not influenced by new protocols.

5.4.3 Longevity and Consistency

The information itself and the identifiers (e.g. the addresses) should be able to survive indefinitely, which means that from the point of view of applications the same object represents always the same information and can be accessed with the same address over the time[3].

If some information cannot be accessed at a given time and/or the information is unreliable or corrupt the model should be resilient to such inadequate information.

5.4.4 Mobility

It should be possible to move information from one physical location to another as well as from one administrative domain to another.

[3]Always assuming that the information or the address is not changed on purpose.

5.4.5 Linking

As already mentioned, linking increases the value of information in a system. Currently, information systems in the Internet allow linking in a very restricted way. The WWW, for instance, restricts links to being two-ended and unidirectional and it is therefore not possible to remove the source of a hyper-link if the target document is removed since servers generally do not allow clients to edit the documents stored in the server.

The following requirements for linking and building relationships can be identified: [MD98]

1. **Multiplicity:** Relationships between two or more objects must be possible. The dimension and the direction of links has to allow more than two entities to be related. Bidirectional and 1:n relationships as well as transclusions should be possible.

2. **Link Typing:** The meaning[4] of a link has to be expressed in its type. Additionally meta-information and representations for applications should be provided.

3. **Linking to Links:** In some cases the objects being linked are themselves objects so that it must be possible to relate to links. This requirement makes links to so-called "first-class objects" which again can be addressed and are therefore equal to objects that represent information entities.

4. **Link Persistence:** As described above, the targets of a link may change. Both addressing and link-management have to be persistent and have to resist changes. [DS95] [MD99]

Link Persistence

As described earlier in this chapter link persistence is an important requirement for a general link management.

The stability and robustness of the addressing mechanism is a precondition for consistent links. [MD99] Nevertheless links may break since a link-target may not be accessible at a given time. (e.g. if a network problem occurs) Another problem are structure-operations: If a document is moved to another location then all links pointing to that document have to be updated. A matured link-management has to keep links persistent and has to react predictably if a link target is not available. In general we have to distinguish two types of systems:

[4] HTML now provides REV and REL flags to express some simplistic link types.

First, an open system like the WWW does not allow persistent links. In an open system, link targets can be tested by simply accessing the resource from time to time. A hash-value can be computed from the target-document. If the target is accessed later the hash-value is compared to the one stored previously. If the two values are equal, link-consistence can be supposed. If the values are different the link-source has to be removed.

The second type is a system like Dino, in which all operations are done via the system itself. Links can be kept persistent if the system gets a notification of each operation on documents. In order to be able to react on editing and changes links should be stored externally (e.g. in a database) so that the document itself does not have to be accessed.

It is obvious that holding links persistent can lead to tremendous administrative overheads (e.g. updating the database) and therefore matured algorithms and mechanisms have to be applied. Section 5.6 explains how links are kept persistent in the Dino system.

5.5 Related Work in Progress

5.5.1 Xlink and XPointer

The following example shows a link expressed in XLink [MD98] which is the XML link-language:

```
<mylink
    xml:link="simple"
    title="Remark"
    href="http://www.dinopolis.org/xml/note.xml"
    show="new"
    content-role="annotation"
    ID=''x34''>
</mylink>
```

XLinks can be stored externally. The link has a title and a type and the show property indicates how the link is represented to applications. The content-role describes the function of the referred target as an annotation. Note that the link is addressed by an ID.

5.5.2 Addressing into the Structure of Documents

Until now we were talking about addressing whole documents. When addressing into the structure of documents is required we have to distinguish documents that support addressing into the internal structure and those which do not. A GIF-image for example does not allow anchors to be defined.

As we know, some documents allow addressing into the structure of documents. In the WWW, for instance, parts of documents are addressed using so-called fragments. The following example addresses the paragraph "Introduction" inside the document "index.html":

```
http://www.dinopolis.org/index.html#Introduction
```

It is necessary that the document contains an anchor which is the target of this resource locator:

```
<a name=''Introduction''>
```

If someone wants to address the section called "Introduction" and no anchor has been defined in advance other methods like "the first section in this document" can be applied assuming that we know about the internal structure of given document.

As an example let us consider the Extensible Markup Language (XML)[5] which uses so-called XPointers (The XML Pointer Language) [MD99] for addressing parts of documents.

The following line addresses the first sub-section in section 2 of a document [MD99]:

```
child(2,SECTION).(1,SUBSECTION)
```

where child() in this context means that these sections are "children" of the document.

Another problem is that this method of addressing is not resilient to changes: If, for instance, someone moves sub-section 1 to another location in the document without telling us, the address becomes invalid. A more robust method of addressing is adding an ID to each part of a document [MD99].

[5]XML: `http://www.w3c.org/XML/`

If the ID is moved with that part of the document the addressing method withstands editing and is resistant to changes.

In the above example the addressed sub-section could be referred to by:

```
child(ID,5)
```

However, not all parts of a document actually have an ID added and not all documents support IDs.

Conclusion: If addressing into the structure of documents is required, complete knowledge about the internal structure of the document is necessary and unique identifiers for each part of the document have to be found. [MD99] It is not possible to find a general solution for arbitrary document types but for each type an appropriate solution has to be found.

5.6 Addressing and Link-Management in the Dino System

The Dino addressing scheme is hierarchically structured and allows direct access to objects. It maps all external objects into an internal addressing tree independent of the logical structure of the external system. For example, a graph-based structure in an embedded system could be mapped into a flat tree structure. The addressing tree does not reflect any relationships among the objects. As an example the edges in a graph structure are not represented in the address tree. Relationships between objects are represented by Dino Relations explained later in this section.[dT99]

The Dino system provides one single point of access: The Node object. Applications access information through this object and do not have to deal with multiple protocols.

In figure 5.3 we can see an application accessing the Dino system via a View. The View can be asked for a Node object.

The open design of Dino allows new external systems to be embedded.

Every addressable object is identified by a **G**lobally **U**nique **D**ino **H**andle (GUDH). A GUDH describes the resolution hierarchy in the Dino address space and consists of a chain of so-called Handle Chunks (HC). Each single HC is responsible for addressing one single object in its domain so that beginning with the virtual Dino root each HC addresses a sub-tree of the global addressing tree. It is possible to address parts of documents or a set of meta-data. The following example shows a GUDH which addresses

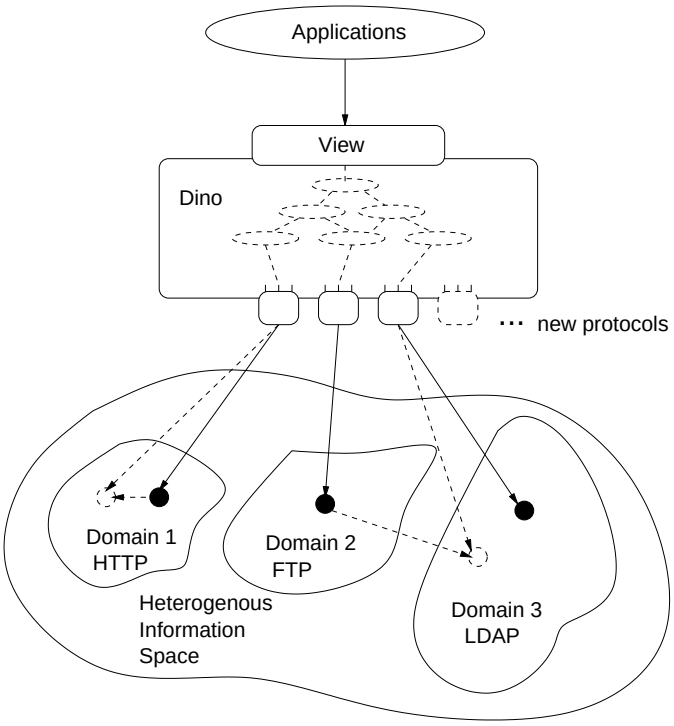

Figure 5.3: Accessing Resources in the Dino System

the "author" meta-datum of a document called "Info.txt". Note that this GUDH is an explanatory example.[dT99]

```
[dinopolis.org]-
  [/]-[home]-[smith]-
  [Info.txt]-[meta-data]-[author]
```

On each level resolver objects exist which are responsible for resolving the corresponding HC. In our example a resolver exists that knows how to resolve the domain "dinopolis.org" and subsequently one that knows how to resolve the directory-structure and another that knows where to find meta-data until the last resolver returns the author object.[dT99]

It is important that one single GUDH object exists which is always the same, even if an object is moved to another location. If, for example, the above file is moved from "smith" to "green" the GUDH object is not replaced but the "smith"-HC is substituted by a "green"-HC. [dT99] In other words if an application holds a reference to a GUDH it always refers to the actual address and if structure operations occur the GUDH object itself does not

have to be replaced.

For each GUDH a String representation is provided which can be human-readable if required. [dT99] This String representation can be used to store GUDHs outside of the Dino system. If the said GUDH is resolved later the Dino system tries to find the corresponding object. It is obvious that if structure operations occur during this external storage and retrieval, it could happen that the referenced object does not exist any more. In this case the Dino system reacts in a predefined manner.

As already mentioned it is forbidden to use the addressing tree for navigational purposes. For instance, it is not allowed (and not possible) to gather from the HC-chain

```
[/]-[home]-[smith]
```

that "smith" is a subdirectory of "home". In Dino these relationships are represented by Dino Relations. A Relation consists of two Pins. Each Pin is attached to an object and knows the address (the GUDH) of its counterpart. In our example, a parent-children relation is represented by two Pins one attached to "usr" and the second attached to "home". Pins are addressable, typed objects.

Relation objects which represent two Pins are used to provide information about relationships to applications.

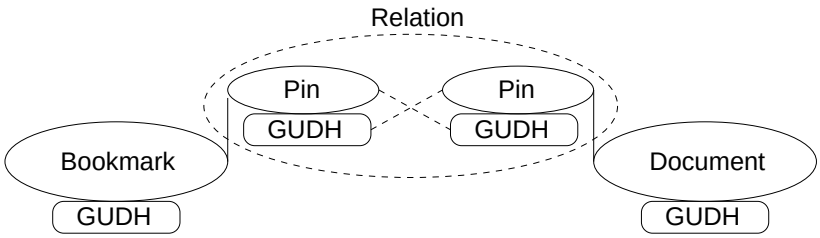

Figure 5.4: GUDHs, Pins and Relations

Figure 5.4 shows a bookmark object (a Node), the object representing the document and two Pins attached to each object. Since Pins are used internally, a Relation object is used to represent relationships to applications.

Pin objects are implemented according to the Proxy design pattern in order to allow us to react on structure operations and to ask objects for their relations to other objects without explicitly resolving the relations.

So-called Pin Containers are used to hold multiple Pins of the same type or sub-type so that relations can be grouped and links of arbitrary dimensions can be realized.

Together with the Pin Manager which, among other things, is used to hold Pins which can not be attached to Dino objects at a given time, Dino Relations are can be hold persistent.

For more details on GUDHs and Relations please see the Dino Software Engineering Document [dT99].

Chapter 6

Distributed Objects

When talking about resources in an information system we think of human readable documents, multi-media objects, services like ftp and so on. Resources can either be passive (e.g. a GIF-image) or active like telnet. In a heterogeneous multi-user information system, people share these resources. Applications make use of resources in different ways. First, they access the said active and passive documents, use them to carry out their tasks or make them available to users. On the other hand, applications should be able to make their own capabilities accessible to other applications. On a local machine, programs are able to interact with other applications. But if programs are running on a distributed architecture we need a framework that allows network boundaries to be crossed. This framework allows distributed programs to call each other as if they ran on one single machine. When thinking in a layered fashion, the whole framework consists of wires, network protocols, services like transactions and a great deal more.

In this chapter two parts of this framework are picked out: Remote Method Invocation (RMI) and Directory Services. The first includes a short review at Remote Procedure Call (RPC) followed by a description of the Common Object Request Broker Architecture (CORBA) and Java Remote Method Invocation (Java RMI). Although they all have their own mechanism for locating distributed objects the X.500 specification for directory services, as long as directory services can be used to store or locate remote objects is introduced. Finally we will see how distributed Dino systems are connected using the Dino-Dino protocol .

6.1 Introduction

In order to understand what distributed objects are, let us consider a bookmark object which resides on a bookmark server and an a client who accesses the bookmark collection from a local machine (see figure 6.1): A stub of the original bookmark object is sent to the client. When the client invokes a method on the stub the request is forwarded to the original object on the server. The original object computes the request and the result is sent back to the stub which then forwards the return value to the client.

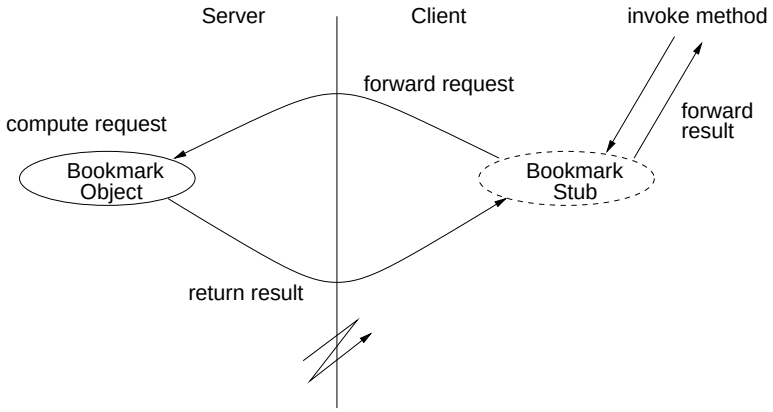

Figure 6.1: Accessing a Remote Object

CORBA or Java RMI allow distributed objects to communicate, to process information in a distributed fashion and to move from one location to another. CORBA can be used for programs not written in an object-oriented programming language while Java is itself a programming language providing an API for RMI and CORBA what directly couples the language with distributed object services. [Gre99]

6.2 Remote Procedure Call

Traditionally, Remote Procedure Call (RPC), which is often implemented over TCP/IP or UDP, is a method for clients to communicate with a remote computer program. By defining a common API, the Interface Definition Language (IDL), and mechanisms to deal with different forms of data representations it allows computers from different manufacturers to call procedures on remote machines. There exists a variety of RPC implementations: For example, Sun NFS is based on Sun's RPC implementation ONC RPC which is one of the most widely implemented RPC mechanisms. [Gre99]

If two computers agree upon the same RPC implementation, a compatible network protocol and a common data representation they can execute procedures independently of the operating systems or hardware platforms beneath. So-called "Stubs" are the communication interface between client and server implementing the RPC protocol. Clients are either statically bound to a server or may find the the server through a directory service, which is called dynamic binding.

6.3 CORBA

Like RPC, CORBA is not a new technology. The Object Management Group (OMG) was founded in 1989 and published version 1.1 of CORBA in 1991. CORBA version 2.1. was published 1997 and offers a standard for distributed object technology. [Gre99] CORBA version 2.0 is multi-platform and language independent but the protocols do not support callbacks and it is difficult to configure firewalls for CORBA communications since CORBA protocols do not provide enough information for TCP/IP or SOCKS based firewalls. [Cob98] Another weakness of CORBA 2.0 is that the protocols provide insufficient support for SSL. [Cob98] In the following sections we will have a closer look at the CORBA architecture. Finally 6.3.3 an overview on changes in the future CORBA 3.0 specification is given.

6.3.1 Architecture

Using the Interface Definition Language (IDL) the programmer defines the objects' interface specification. which describes the capabilities of an object. Interface specifications are stored in so-called Interface Repositories. The implementation behind these interfaces (e.g. the program or object itself) is not part of the CORBA specification. Specific implementations of CORBA are often based on Object Models as for example the Smalltalk Object Model or the C++ Object Model. [Man98]

CORBA objects share the IDL interface specification with each other via the Object Request Broker (ORB). Figure 6.2 shows this architecture. Furthermore, the ORB provides access to the standard CORBA services such as transactions, security features and a naming service.[Gre99]

As already mentioned CORBA is able to encapsulate programs that are not written in an object-oriented programming language, like Fortran or Cobol.

The whole set of CORBA objects can be served by different ORBs spread over the network. ORBs communicate using the General Inter-ORB Proto-

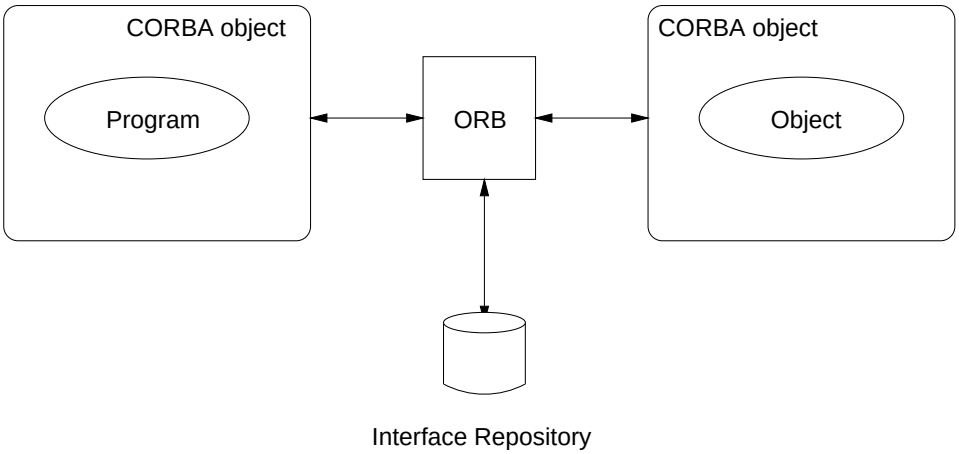

Figure 6.2: Object Request Broker (ORB) and Interface Repository

col (GIOP) which can be based on TCP/IP or other implementations like the DCE Common Inter-ORB Protocol which uses RPC for communication between ORBs [Gre99].

The GIOP implementation based on TCP is called the Internet Inter-ORB Protocol (IIOP). To access a remote program the client must know the address, the Interoperable Object Reference (IOR). Using IIOP, part of the address is based on the server's port number and Internet Protocol (IP) address.[Man98]

6.3.2 Object Discovery

A CORBA object can be configured statically with information about another specific CORBA object. This is called Static Method Invocation (SMI) and allows the IDL compiler to verify data-types and so on at compile time. Another method is Dynamic Method Invocation (DMI) where objects query the ORB for objects which have an appropriate interface specification.

6.3.3 Work in Progress

CORBA 3 will specify an Inter-ORB Protocol that provides more information for firewalls. The new extensible format provides additional information for TCP firewalls, SOCKS proxy firewalls and special GIOP proxy firewalls. GIOP proxies are able to examine the Inter-ORB Protocol and assure that it fulfills special security requirements in order to grant that object access to resources behind the firewall. [Cob98]

Another improvement is that CORBA 3 will support Inter-ORB communication via Secure Socket Layer (SSL).[Cob98]

Furthermore there will be satisfactory solutions for callbacks. The main problem with the present specification of CORBA is that the target host of callback operation invocations is rather a workstation then a server host and firewall configurations generally do not permit workstations to be target of incoming connections. The present solution of opening a separate TCP connection for that purpose fails in this case. The so-called bi-directional GIOP allows a server to reuse an existing connection to invoke callback methods on the client. [Cob98]

The OMG[1] announced that they will concern on supporting the Enterprise Java Beans (EJB) as the default object model behind CORBA.

6.4 Java RMI

Java RMI allows applications to call methods on remote objects which may reside on another virtual machine. The Java RMI architecture is shown in figure 6.3.

The reference to remote objects is obtained using the RMI Registry which is a naming-service that allows a server to "bind" objects to a name. (see step 1 in figure 6.3) Clients get the reference by "looking it up" with that name. (step 2) The RMI Registry can be run on a different machine as the client or the server. Once the client has got the reference it may invoke methods on the remote object.(step 3)

If, for instance, a parameter of a method is again an object the server needs the class file of that object. If the class file is not available on the server the RMI class loader allows to load it from a file server. [Mic98]

6.4.1 RMI Protocol

The standard low-level RMI protocol is called Java Remote Method Protocol (JRMP) and is based on TCP. If RMI determines that the remote call is addressed to a location behind a firewall the protocol is wrapped in a HTTP request and sent via the proxy to the remote location. In this case the communication is slower. The third protocol which can be used is the IIOP of CORBA. Other protocols (e.g. SSL) can be implemented by writing the corresponding implementation for the RMI Socket Factory. If no server-socket can be created (e.g. if the security manager does not allow an applet to create one) the RMI protocol uses an existing connection (e.g. the applet's

[1] Objectwatch: `http://www.objectwatch.com/issue19.htm`

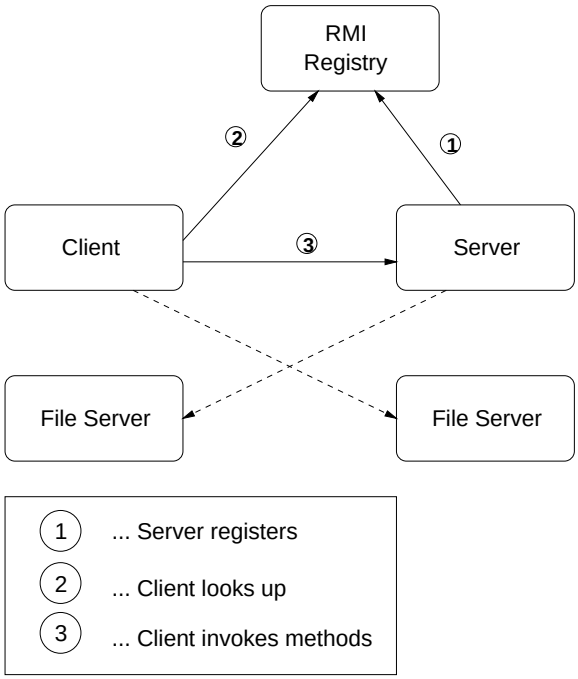

Figure 6.3: RMI Architecture

connection to its host) and provides the server functionality using the RMI Multiplexing Protocol which allows, for example, the codebase host to invoke methods on that RMI object. [Mic98]

6.4.2 Stub and Skeleton

Java RMI uses the so-called stubs and skeletons to provide RMI functionality. If a class directly implements the Remote interface it is possible to generate stubs and skeletons from it. Stub and skeleton are generated using the rmic compiler. The rmic takes compiled Java class-files as input. [Mic98]

The skeletons remain on the server-side and the stub is simply serialized to the client. Now the client may invoke methods on that stub. The stub forwards the method call by sending the request and the parameters to the skeleton where the method is invoked on the remote object. The return value is sent back to the stub which again forwards it to the invoker. Stub and skeleton themselves do not implement any network functionality. Low-level rmi classes take care of these tasks. [Mic98]

Every method in a Remote object may throw a *RemoteException* if for example a network error occurs. A *RemoteException* is a subclass of the

IOException. [Mic98, see RMI Specification section 4.1]

When calling a remote method the parameters and return values have to be of one of the following types [RSL99]:

1. **Remote:** The arguments and the return value can again be Remote objects. A Stub is sent to the counterpart and method invocations are forwarded to the original.

2. **Serializable:** An object that implements the java.io.Serializable or its sub-interface java.io.Externalizable can be serialized to a remote location. That means its state is copied in a byte-stream representation. The receiver of the object may, together with the appropriate class file, "de-serialize" the object by converting it back to an object. When an object is serialized all objects to which it holds strong references are serialized as well, since the behavior of an object depends on his environment. If the corresponding class-files are not available at the receiving party, it is the responsibility of the receiver to locate and (down)load the appropriate class definitions

3. **Marshalled Object:** A java.rmi.MarshalledObject can be obtained from a Serializeable or a Remote Object:

```
java.io.Serializable serial = ...
new java.rmi.MarshalledObject(serial)

or

java.rmi.Remote remote = ...
new java.rmi.MarshalledObject(remote)
```

   Marshalling means to convert an objects state and its codebase(s) into a byte-stream. The codebase is included to allow automatically loading the class-file(s) from the host location. Marshalled objects are available only on Java 2 Platform.

These three objects together with the Java Naming and Directory Interface (JNDI) are the Java objects which are able to cross network boundaries. A JNDI reference is used if it is not possible or not desired to use one of the above representations. If a JNDI object is used not the object itself is sent

across the net but the JNDI object defines how to obtain a reference to the original object.

6.4.3 RMI Class Loader

Java RMI includes a class-loader to get the corresponding class files for parameters and return-values. If the corresponding class file is available on the remote location this class file is taken. If not the class can be loaded from a server. Therefore RMI uses a subclass of the *ObjectInputStream* and *ObjectOutputStream* classes and overrides the *annotateClass()* and *resolveClass()* methods. The *RMIClassloader.loadClass(URL codebase, String name)* method is called up to retrieve the missing class files. [Mic98]

6.4.4 Distributed Garbage Collector

For each remote object exported in the local virtual machine, the garbage collector maintains a reference list which is a list of clients that hold references to the server object. A reference is added to the list when a new client "connects" to the server and is removed when the client disappears. If there are no more entries in that list the server object can be garbage-collected as well. This mechanism is called "Reference Counting". [Mic98]

It is possible to create cyclic references with RMI which cannot be resolved by the DGC. For example, in a peer to peer relationship of two remote objects that reference each other this state is given. Special mechanisms have to be used to "disconnect" these objects. (e.g. through Java weak references) [Mic98]

6.5 Directory Services

Traditionally, Directory Services are used to store data in a way that this information is accessible via the net. For example you can store a person's telephone number and therefore share that information with others. As we have seen in this chapter the kind of data that is typically shared by applications are the objects themselves. CORBA uses the ORB whereas RMI uses the RMI registry to find objects and to obtain a reference to these objects. RFC 2713 [2] defined how Java objects (Serialized, Remote, Marshalled and JNDI references) can be stored in an Lightweight Directory Access Protocol Server (LDAP-Server). This is the reason why a brief overview on Directory Services is given.

[2] RFC 2713: `http://www.faqs.org/rfcs/rfc2713.html`

6.5.1 X.500

The X.500 Standard by the Committee for International Telegraphy and Telephone (CCITT) and the International Standards Organization (ISO) is an open standard for directory services and was first published in 1988. The main functions of X.500 are:

1. **Naming:** The standard defines a scheme for the naming of directory entries. The components are so-called Distinguished Names (DN) which are composed of several Relative Distinguished Names (RDN). Each RDN is relative to its predecessor and subsequently describes the final location more closely. Moreover, it is possible to define rules for what information can be stored in one RDN. (e.g. if a RDN is a telephone number you can specify that this RDN has to consist of numerals) [Gre99]

2. **Structure:** The structure of the internal directory is specified by the Directory Information Tree (DIT) whereas the entries themselves are stored in a database called the Directory Information Base (DIB). The database and the tree can be shared among multiple servers. They can be replicated for reliability and performance purposes. To communicate with other servers a directory server employs Directory System Agents (DSA).

3. **Client Access:** This function defines the Directory Access Protocol (DAP) and the Directory User Agent (DUA). Clients use those to send commands and directory queries to the directory server. Note that the Lightweight Directory Access Protocol (LDAP) is an implementation of DAP and therefore just a part of X.500. (LDAP see: 6.5.2)

4. **Security:** X.500 supports X.509 certificates and public/private key cryptography.

6.5.2 LDAPv3

As already mentioned LDAP, in its present version 3, is an implementation of the X.500 DAP protocol based on TCP/IP and since it is published by the Internet Engineering Task Force (IETF) it is much more popular in the Internet society. RFC 2251 [3] , which describes LDAPv3, describes (apart from the protocol itself) data types and structures, how to submit complex queries and how to interpret search results. APIs for LDAP exist and there is work in progress to provide LDAP APIs in Java 1.3 and other programming languages as defined in the RFC 2251.

[3]RFC 2251: `http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2251.txt`

6.6 Dino-Dino Protocol

As explained in chapter 4 several Dino systems can be connected. The relevant parts of Dino are made network-accessible using either Java RMI, CORBA for objects not written in Java or a self-defined protocol (the Dino-Dino Protocol). This allows the choice of one of the standard protocols as discussed earlier in this chapter or a self-written protocol which may be optimized for performance requirements or for special working conditions. [Dal99]

Chapter 7

Security Considerations

The term security has been mentioned several times throughout this document and we saw that the presented technologies provide various satisfactory security mechanisms. This chapter explains why Shared Bookmarks need security and what kind of security threats an application like Shared Bookmarks can be confronted with. An overview of cryptographic technologies is followed by a discussion of some technologies used in the Internet such as digital signatures or X.509 certificates.

Besides this theoretical part of the chapter we will have a closer look at security capabilities of the Java programming language. Additionally Java provides some useful command line tools to create signed applets and to manage certificates and crypto-keys.

To answer the question "Why do Shared Bookmarks need security?" consider the following: Shared Bookmarks are used in a health care information system to collect medical patient records. We have to ensure that no unauthorized person gets access to this collection nor is able to manipulate, delete or add entries. Even if each single report is protected, the whole collection could give information about the progress of illness of a person. If, for instance, the last report of the collection says that the patient has recovered, and this report is deleted, then the information, which the whole collection represents, is wrong. Next we will see the requirements which users and/or servers should demand in a secure information system.

7.1 Requirements

Generally users should have at least some simplistic requirements to the system they use, independent from the importance of their data or a specific

threat against their personal data. In some cases lots of small pieces of data concerning one particular person or organization can be put together like a puzzle in order to obtain a general image of that entity. In other words although some information seems to be worthless for others a collection of many single "unimportant" pieces of information could be misused to harm a person. The following list shows the five main security requirements: [KK97]

1. **Server Authentication:** It is not enough that the user confirms his authentication but the server has to authenticate as well. We have to prevent a third person making a user think that the server he accesses is the "right" one but it is not.
2. **Client Authentication:** The user has to confirm his identity by providing a password, a card or his fingerprints. Note that this information is data which should be protected as well.
3. **Secure Transmission:** Data should be sent across the net confidentially which means that no third person should be able to access it.
4. **Prevent Misusage:** Data residing on the server has to be prevented from manipulation at all times.
5. **Non-Repudiation:** It has to be possible to verify the sender of specific information who is responsible for the content of a message. It has to be prevented that someone says: "I did not send this message!".

Note that these requirements are not only user-requirements but also servers will require them - especially if users may upload data to a server. [KK97] These requirements are often summarized by two terms: Authentication and Privacy. [GS96]

7.2 Security Leaks

The main problems in the existing Internet infrastructure are the IP protocol, as it does not support encryption or authentication, and the Domain Name System (DNS) which is the basis for addressing in the Internet. Applications or protocols which directly use IP and DNS are un-secure as long as they do not provide any security features by themselves.[1] It is possible to eavesdrop IP packets, read and manipulate the contained data, misdirect packets, and add new faked packets into the stream. (e.g. add a packet with a "guessed" TCP sequence-number) [GS96] So-called "Sniffers" log the first packets of new connections (like ftp or telnet) which contain user-name and

[1] This conclusion is valid for other insecure protocols as well - not just for IP and DNS.

password. This information is used later to gain access to servers and local networks. On the other hand "IP-Spoofing" are hacker attacks using faked IP-Headers (e.g. faked IP Addresses) to open connections to a server or to pass firewalls. [KK97]

7.3 Cryptography Basics

When talking about authentication and privacy we have to keep in mind that we are not just talking about the data sent across the net but also about passwords. [Van96] It is even more dangerous if someone finds out a user's password as he will be able to read e-mails.

To hide information from others there exist two possibilities: cryptography and steganography. The first (and most important) is to encrypt data with a key so that others are not able to read it. The latter is to make data appear to be something else by embedding it in an unimportant carrier. (e.g. code information into a GIF-image or sound recordings) [Van96] [GS96] [2]

We can distinguish between three categories of cryptography used today by many Internet protocols and applications: [GS96]

- **Cryptographic Checksums:** Similar to hash values checksums are generated from a set of data and supply a kind of fingerprint. It is important that each single byte or each single character of the whole data-set is used to calculate the checksum. If one single character is changed the whole checksum changes. That is the way in which manipulations can be detected. Most Internet applications use the Message Digest 5 (MD5) algorithm which generates a 128 bit checksum. Some digital signature algorithms use a message digest algorithm to compute the hash values of the data that is being signed, and then digitally sign the hash value rather than the original data, since digitally signing the whole data could be very slow. [GS96] [KK97]

- **Symmetric Algorithms:** Symmetric key techniques use a key that is shared by all participants of the communication. The key is called shared key or secret key. Symmetric algorithms are fast and easy to implement. As an example, the Digital Encryption Algorithm (DEA) (used in DES) uses a 56 bit long key (64 bit - 8 bit parity) to encrypt messages. Triple DEA uses two 56 bit long keys. The message is encrypted using the first key, decrypted with the second and again encrypted using the first. It is more secure and used because existing

[2] More information on steganography can be found at the Berkeley University: `ftp://ftp.csua.berkeley.edu/pub/cypherpunks/steganography` This software allows to encode data in and retrieve data from Macintosh PICT-format files.

chips and algorithms can be reused. IDEA uses one 128 bit long key. RC5 by RSA Laboratories [3] is the predecessor to the RC6 standard which is a finalist of the contest organized by the National Institute of Standards and Technology (NIST). The result of this development effort will be called the Advanced Encryption Standard (AES) and will be the official successor of the DES algorithm. [4]

- **Asymmetric Algorithms:** These algorithms use two keys belonging together: a Public Key and a Private Key. If a message has been encrypted with one of the two keys this message can be decrypted only with the appropriate counterpart. The private key belongs to one single person and has to be kept secret. The public key can be distributed to others. If someone wants to send a message to the one who owns the private key the sender encrypts the message using the corresponding public key. The only person who is able to decrypt this message is the owner of the private key. On the other hand the private key can be used to create certificates. The sender generates a checksum or hash-value from the message and encrypts that checksum using his private key and adds it to the original message (which is i.e. encrypted using the other's public key) The receiver is able to verify the author of the message using the senders public key to decrypt the checksum. How the receiver is able to verify that a specific public key belongs to a specific person is discussed later in this section. Asymmetric algorithms use key sizes greater than 2048 bit and are much slower than symmetric algorithms.[KK97] [GS96] [5] .

For performance reasons the public key principle is often used to exchange a symmetric key called session key which is then applied to the messages. [GS96]

Asymmetric algorithms are for example used in Pretty Good Privacy (PGP) for secure e-mail messaging, in Secure Shell (SSH) for secure logins from remote computers [6] and in Secure Socket Layer (SSL) for secure WWW access.

[3] RSA: `http://www.rsa.com/rsalabs/aes/`

[4] AES: `http://csrc.nist.gov/encryption/aes/aes_home.htm` This algorithm will have 128-, 192- and 256 bit size and will support block cipher at a minimum block size of 128 bit.

[5] Here you can find a collection of related links:
`http://www.ietf.org/html.charters/pkix-charter.html`

[6] SSH which is mainly used on Unix platforms also allows to provide secure Unix X11-server connections.

7.4 Public Key Management

Public Key Management or Public Key Infrastructure (PKI) deals with the distribution of public keys and allows the verification that a specific public key belongs to a specific person. It can not be assumed that the participants of a communication have already exchanged their public keys. Certificates were invented as a solution to the public key distribution problem. A Certification Authority (CA) can act as a Trusted Third Party. They testify that a public key belongs to a person. CAs are hierarchically organized and, for example, LDAP directory services are used to store public key informations together with these certificates. [Van96] [Jaw96][7] There are many public Certification Authorities, such as VeriSign, Thawte, Entrust, and so on.

The result are so-called certificate chains where one CA certificates that the next CA is valid. The format is standardized in the X.509 standard. The later sections will refer to certificate(s) where the "(s)" stands for the possibility of certificate chains. [GS96] [KK97]

Certificates are not indefinitely valid. Therefore, certificates contain a valid period parameter. After some time the certificate is added to a Certificate Revocation List (CRL) and is, until then, not valid anymore. CRLs are time-stamped and signed by the CA. [GS96]

7.4.1 X.509 Certificates

The X.509 standard defines the content of certificates and Certificate Revocations List (CRL) and their data format. The main part of a certificate is a the digital signature itself. Furthermore it contains the following data: [Inc]

- **Version:** Defines of which version the certificate is. Actually there exist three versions. Version 1 has been issued in 1988 and is the most general one. Version 2 introduced the concept of unique identifiers for issuers which can be reused instead of names. Version 3 (issued 1996) is an extensible format and allows users to add their own extensions to the standard specification.
- **Serial Number:** This number is assigned by the author of a certificate when it is issued. It is placed in a CRL when its validity period is expired.
- **Signature Algorithm Identifier:** Is used to identify the algorithm used by the CA to sign the certificate.

[7] PKI: `http://www.ietf.org/html.charters/pkix-charter.html`

- **Issuer Name:** Is the X.500 Distinguished Name of the entity (e.g. the CA) who signed the certificate.
- **Validity Period:** Certificates are only valid for a given period. This period may vary from a few seconds for critical certificates to a whole century. After that period the serial number of the certificate is added to a CRL.
- **Subject Name:** Is the name of the entity whose public key the present certificate confirms. Again the X.500 naming standard is used to represent this name.
- **Subject Public Key Information:** Is the public key the entity which is described by the Subject Name.

7.5 Law Restrictions

Strong encryption algorithms often underlie law restrictions since they can be used to hide information from the government. The government of the United States, for example, considers some algorithms as dangerous as weapons or drugs and restricts or entirely forbids their export to other countries. That is the reason why the extensional Java package for cryptography can not be exported from the US. [Van96]

7.6 Work in Progress

The IETF still works on an integration of security mechanisms in the present IPv4 protocol[8] while IPv6 provides supports the above described mechanisms.

The Domain Name System Security working group[9] works on security integration in the DNS.

The IETF works on a framework which allows applications to transparently use these security mechanisms and on the other hand allows the exchange of modules (e.g. the algorithms).

[8]IPV4: `http://www.ietf.cnri.reston.va.us/html.charters/ipsec-charter.html`

[9]DNS: `http://www.ietf.cnri.reston.va.us/html.charters/dnssec-charter.html`

7.7 Java Security

As already mentioned, applications which directly use the provided Internet infrastructure are not secure unless they implement security mechanisms by themselves. Java 2 provides a variety of security features through its security API and three extensional packages which enhance the security capabilities of the Java programming language. As Shared Bookmarks can be run as an applet we will see what kind of restrictions an applet is confronted with. We will have a short review at the security architecture of Java 1.1 - the sandbox model - and signed JAR files. Finally, an overview of the standard Java 2.0 Security API and some additional tools such as the JarSigner or the Policy tool is given.

7.7.1 Java 1.1. The Sandbox Model

Applets are applications that can be downloaded from the net and are executed on the local machine. Therefore applets run inside a special environment which prevents the applet from accessing local resources. Figure 7.1 shows the so-called Sandbox Model. [Inc]

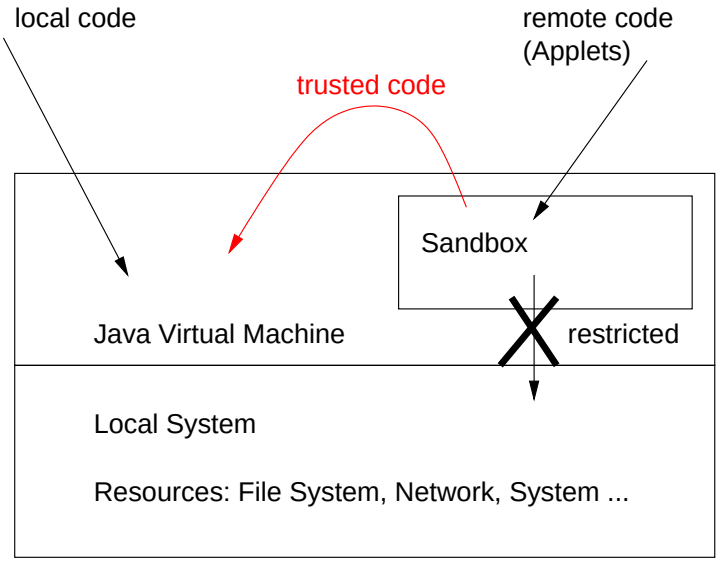

Figure 7.1: Java 1.1 Sandbox Model

This model distinguishes code loaded from the local classpath and code loaded from a location on the net. The local code may access all resources required but applets are confronted with the following restrictions: [Jaw96]

- no access to the local file system.
- no access to the local system such as system properties, etc...
- restricted Network access - for example, an applet is not allowed to open a network connection to any other than the host it has been downloaded from.
- thread manipulation - an applet is not allowed to access any threads running on the local machine.
- no factory object creation - an applet is not able to, for instance, replace a network related socket factory because this would allow to bypass restrictions of point 3.
- an applet is not allowed to add its own ClassLoader - again this would bypass the system ClassLoader which checks all classes that are loaded on the local virtual machine.
- window creation is restricted

In Java 1.0 this Sandbox Model it was not possible to explicitly give certain permissions to an applet running inside this restrictive environment.

In Java 1.1 the concept of the trusted or signed applet was introduced. Applets are packed into a Java ARchive File (JAR) and signed using a certificate. The system that receives the applet then decides if it trusts that applet. (e.g. by showing the user the digital signature of the company which programmed the applet) If the user (or the system) judges that applet as trusted it is executed under the same conditions as local code and is able to access local resources without any of the above restrictions. [Inc] If an application wants to specify a more fine-grained security policy than this "allow nothing" or "allow everything" policy, the programmer has to subclass the SecurityManager and ClassLoader classes to specify permissions. (e.g. to define a new permission the programmer has to add a new "check" method in the SecurityManager class) This requires in-depth knowledge of computer security and the Java Virtual Machine. [Van96]

These disadvantages of the sandbox model led to the security model of Java 1.2 shown in figure 7.2.

7.7.2 Security Model in Java 1.2

The big difference to the old model is that local code and remote code is treated equally. The restrictions for a piece of code are made up by a policy and the ClassLoader. The policy allows to define fine-grained permissions

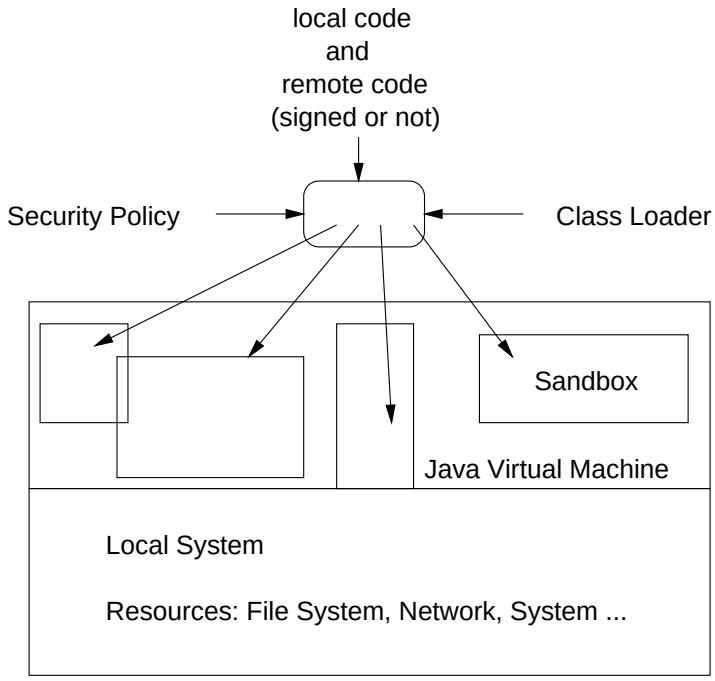

Figure 7.2: Java 1.2 Security Model

for domains. A policy file stored in the local file system is used to define these permissions. As an example the following policy file allows a class called "Test", which is downloaded from the domain www.dinopolis.org, to write on the local file system: [GS96]

```
grant
  SignedBy "Dino Team"
  CodeBase "http://www.dinopolis.org/"
  {
    permission java.io.FilePermission
               "Test",
               "write",
               "Dino Team";
  };
```

The domain www.dinopolis.org corresponds, for example, to one of the boxes shown in figure 7.2. The sandbox model was a protection domain with a fixed boundary. In the new model two general domains are distinguished: The system which contains all external resources like the file system, screen

and keyboard and so on. The other one is the application domain containing application and applet code. Code belonging to the application domain is able to access external resources only via the classes from the system domain. By defining the above permissions the application domain is divided into sub-domains. Each class (or object) belongs to a protection domain and underlies the corresponding security restrictions. [GS96]

The CodeBase argument is used to indicate where the code is downloaded from. Additionally it indicates that a certificate is stored with the class file. This is necessary to assure that the downloaded code is really programmed by that person or organization. As an example the Dino team could add the Dino certificate to the Test.class file so that everyone can be sure that this file is originally programmed by the Dino team.[GS96]

The certificate allows one to find out the author of a piece of code which can be compared with the SignedBy parameter of the policy entry.[GS96]

Policies and permissions can be created either using JDK's policy-tool or by using the corresponding classes in the java.security package, which are: The Permission.class with its subclasses (e.g. java.io.FilePermission, java.net.SocketPermission, etc. . .) the Policy.class and the CodeSource.class. [Van96]

The Policy.class is an abstract class what allows implementations to store the policy files in a database or, as it creates the default implementation, in a ASCII file like in the above example. Generally, it defines the format of the entries in the policy file. [GS96]

The CodeSource.class encapsulates the code-base which is a java.net.URL and certificate(s) containing public keys that should be used to verify signed code originating from that location.

Additionally policy files may contain keystore entries. Keystores are databases which store certificates together with the corresponding public keys of others or the own public/private key-pairs together with the corresponding certificate. [Inc] In our example the Dino team certificate is stored with an alias to "Dino Team". Note that the keystore is used for two different things: First your own key-pairs (if you need more than one key) together with the corresponding certificate(s) and an alias are stored. Second the certificate(s) of other parties are held in that database. The own private keys are encrypted and protected by a password.

To generate keys and certificates and to manage the keystore database you can use JDK's keytool. It is possible to create self-signed certificates or to generate a request which is then sent to a CA. Self-signed certificates will not be accepted by everyone. The CA checks your identity and returns the certificate after some time. The returned certificate can be imported with the keytool. At the moment requesting a certificate is not supported by the

Java API. [Jaw96] [Van96]

The corresponding Java package for certificates is the java.security.cert package. The following gives an overview of the classes in this package: [Inc]

- the **CertificateFactory** class defines the functionality of a certificate factory, which is used to generate certificate and certificate revocation list (CRL) objects from their corresponding files.
- the **Certificate** class is an abstract class for managing different certificates. It is an abstraction for certificates that have different formats but have important common uses. For example, different types of certificates, such as X.509 and PGP, share general certificate functionality (like encoding and verifying) and some types of information like the corresponding public key.
- the **CRL** class is an abstract class for managing a variety of Certificate Revocation Lists (CRLs).
- the **X509Certificate** class is an abstract class for X.509 Certificates. It provides a standard way to access all the attributes of an X.509 certificate.
- the **X509Extension** interface is an interface for an X.509 extension. The extensions defined for X.509 v3 certificates and v2 CRLs (Certificate Revocation Lists) provide mechanisms for associating additional attributes with users or public keys, such as for managing the certification hierarchy, and for managing CRL distribution.
- the **X509CRL** class is an abstract class for an X.509 Certificate Revocation List (CRL).
- the **X509CRLEntry** class is an abstract class for a CRL entry.

7.7.3 Security Manager vs Access Controller

Applications are currently started without a security manager. Whenever an application requires security management the security manager is started either by a command-line argument or by the corresponding setSecurityManager method. If an applet is started (e.g. in a browser) a security manager is installed automatically (e.g. Netscape starts one). Once installed all system classes call the corresponding checkPermission methods. [Inc]

The new security mechanisms is now called AccessController. For backward compatibility reasons the default implementation of the SecurityManager is that the old check methods call the new checkPermission methods of

the SecurityManager which for their part call the corresponding methods of the AccessController.

7.7.4 Signing Applets

We have seen that signed code together with an appropriate security policy may gain additional permissions on a remote system. The process of generating keys, certificates and signed applets is explained in appendix A.

Netscape Navigator and Microsoft Internet Explorer require different formats of signed applets: For Navigator, applets have to be signed with a Netscape Object Signing ID by creating a so-called manifest. The files and the manifest are packed into a .JAR archive. For Microsoft Internet Explorer, the files must be wrapped into .CAB archives and then signed with a Microsoft Authenticode ID. Currently, Authenticode signing can only be done under Microsoft Windows. Netscape Object Signing can be done under Windows and a variety of Unix platforms. [GS96]

Chapter 8

Implementation

Shared Bookmarks have been implemented in Java using the Dino middleware system. The application uses various features of the Dino system like addressing and linking mechanisms or the user management. The following sections explain the architecture of Shared Bookmarks and the functionality of the bookmark server and the bookmark client.

8.1 Used Dino Features

First the Dino system is used to store the bookmark-data (e.g. the links and properties) and in order to access the bookmarks the Dino addressing mechanisms are used. Client and server communicate via the Dino-Dino protocol (see 6.6). Access rights, user and group management is done via the Dino user-management. Shared Bookmarks use the Dino linking facilities to reference documents. (see Links 8.5) If the users log in via an HTML form (see next section 8.2) the Dino HTTP server is used to parse the submitted form.

Additionally Shared Bookmarks may load frequently accessed documents to a local cache. Dino is used to store the cached documents.

8.2 Architecture

The present implementation of Shared Bookmarks is a client-server architecture. Figure 8.1 shows this architecture. The bookmark client may connect directly to the server or users can log in using an HTML login form which is submitted to the Dino HTTP server. The Dino HTTP server has been

adapted so that it is able to parse HTML forms. It parses the form and calls the user-manager to set the access rights for this session. Next an HTML page containing an applet is returned to the client. Note that in this case the applet itself (e.g. the Graphical User Interface GUI) is sent across the net. Once running in the clients browser the applet connects back to the bookmark server and the corresponding bookmarks and folders are sent to the client. The applet displays the bookmark collection as a tree. The users' actions (e.g. if the user opens a bookmark) are forwarded to the bookmark server which then returns the according document and the client's browser opens the document in a new window.

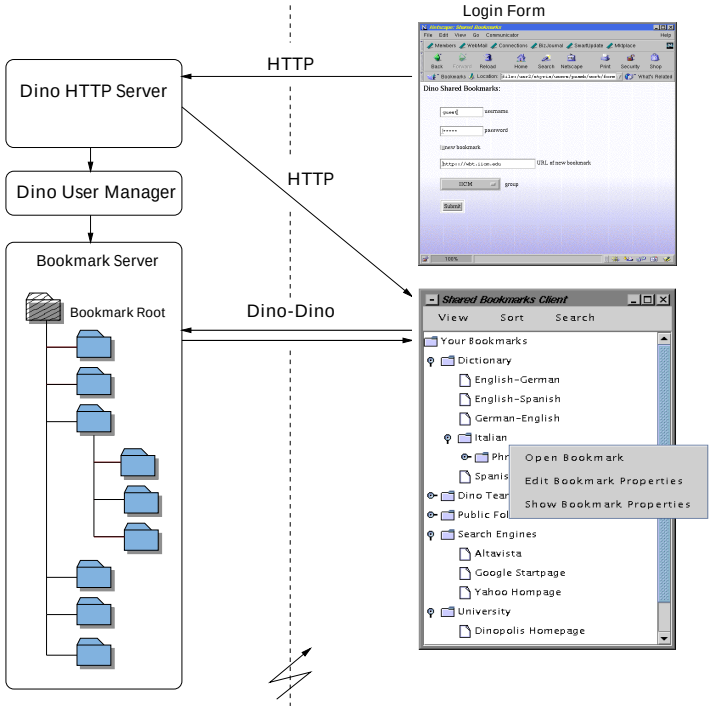

Figure 8.1: Shared Bookmarks Architecture

The protocol used to pass requests to the server is the Dino-Dino protocol (see 6.6 for a description of the Dino-Dino protocol). Another possibility is to run the client as an application what means that users have to install a client program on their local machine. The application connects directly (without accessing the HTTP server) to the server and the bookmark collection is sent to the client. Note that in this case the application itself (e.g. the GUI) is not sent across the net.

Since Shared Bookmarks can be run as an application as well as an applet, one can choose between these two possibilities depending on the

circumstances: If Shared Bookmarks are always used from the same location it is probably better to install the client on the local machine. The advantage is that the only data sent across the net are the bookmarks themselves. The class files for the GUI remain on the local machine and do not have to be sent to the client. If users access the bookmark collection from different locations or if they are not allowed to install the application then they can access the collection via their browser. The performance is much lower in this case since the whole client has to be downloaded from the server.

The current implementation uses Java Swing for the representation of the GUI. Some virtual machines like, for instance, Netscape Navigator's need an extra plug-in to display Java Swing Components within an applet. If it is not possible to install plugins again the application can be installed on the client machine.

When the bookmark client opens a document it sends it to the appropriate application. If it is, for example, an HTML page the document is sent to a browser. Other documents are displayed by other applications. The default implementation is to send the document to the default browser.

Detail: If the client is running as an applet within an HTML page the Applet's `getAppletContext()` method and the AppletContext class' `showDocument(URL)` method can be used to open a document in the browser.

8.3 User Management

The access rights are specified using the Dino usermanager. The Dino usermanager allows the definition of complex access rules for users and groups as well as for files and directories (e.g. bookmarks and folders). This makes it possible to store the bookmarks in one single tree. Each user may access exactly those sections to which he has access rights. It is possible to define access rights which allow users to access sub-directories without having access rights to the parent directory. This is a special requirement for the usermanager since this is not possible in Unix file-systems.

The bookmark server additionally stores a list of members with each folder. This information is used for user notification. If for example a new bookmark is added to a folder all members of this folders are notified via email. The users e-mail itself is stored in the corresponding user record together with the notification interval which is specified by the users.

To receive notifications users may subscribe to folders. Subscribing means that the user gets a notification if, for instance, a new bookmark is added. Generally users are subscribed to those folders to which they have access rights. Users may not subscribe to other folders.

Please see section 8.9 for more details on user notification.

8.4 The Bookmark

Each leaf in the bookmark tree is represented by a bookmark object. A bookmark object stores various properties which are used to log the user's actions and to provide additional information about the referenced document. From the bookmark object a Dino link points to the document it references. Using the Dino link management means that all internal links are persistent so that moving a document, for example, within the Dino system does not produce any dead links. The current implementation stores the bookmark entries in a file system. Using the Dino system makes it possible to physically store bookmarks in a database or even in an LDAP server.

8.5 Links

In the Dino system we distinguish two fundamental links: The first describes the relationship between two objects inside the "Dino world". Inside the Dino world means that all operations on objects are done via the Dino system. In the case of a move operation links pointing to the moved document can be updated. The second general link type is a link to an external object. Dino has no control over these objects. An example is a document residing on a foreign HTTP server. In the first version Shared Bookmarks were storing the URL for these external links. That is why this property is still listed in this document. Now it is just used to provide this information to the user.

The consistency of these external links can be tested by generating a hash-value from the document. Later the hash-value is compared to the one computed during the last test. If the two values are identical we can assume that the document has not changed since the last test.

8.6 Meta Data

The later following description of the bookmark server and the bookmark client requires a detailed definition of the meta-data stored with the corresponding objects. The properties of bookmarks and folders can be classified according to their origin:

1. **Author:** The author of a bookmark may enter properties like key-

words or a description.

2. **User:** On the other hand users who access the bookmark enter, for example, a valuation.

3. **System:** These properties are used by the system. For example, if a user opens a bookmark a counter is incremented so that frequently accessed bookmarks can be shown to the users.

4. **Administrator:** The administrator may edit certain bookmarks. For instance, a flag is set which declares a bookmark as "old". The Administrator may run several maintenance-tools 8.7. These tools may set the corresponding properties as well.

8.6.1 Bookmark Properties

The following set of meta-data is attached to each bookmark:

- **Title of Bookmark:** Is a String and should be the title of the document. If, for example, the document is a HTML document, this property should contain the title of the HTML document. This property is set to a default value by the system. The author may edit this property. It is a required property since it is used in the graphical representation in the bookmark client.

- **Description:** The author may enter a description of the bookmark or of the referenced document. It is optional.

- **Keywords:** This optional property is an array of Strings and is edited by the author.

- **Creation Date:** Is required and set by the system.

- **Author:** The username of the author is required and set by the system when the author adds the corresponding bookmark.

- **New Tag:** This boolean-type value indicates that a bookmark is new. It is set by the system when the bookmark is added. Administrators and authors may edit this property. The administrators may run a tool which removes new tags from bookmarks residing in the collection for a period of time. This argument is required. Note that it is not used to decide whether a bookmark will be deleted or not.

- **Obsolete Tag:** Is a boolean value which is set to false when the bookmark is added. After some time an administration tool sets this value to true. In a next step bookmarks containing an obsolete tag

are removed from the collection. It is required, set by the system and may be edited by the administrator.

- **Visited by User:** Is a list that holds a list of all users that have already visited the bookmark. It is empty at the beginning. So it is possible to determine for each user if he has visited the bookmark or not. It is added to and maintained by the system.
- **Times Visited:** Is a counter of type integer which is incremented by the system whenever a user opens the Bookmark. It is required and set to zero when the bookmark is added.
- **Last Visited:** When users open a bookmark the current date/time is set by the system.
- **Rating:** Is used to specify how important a Bookmark is. The range from 0 to 10 can be chosen. 10 is the highest 0 the lowest value. It is set to 5 per default and may be edited by the author. A filter will allow users to view, for instance, only bookmarks with a high rating.
- **Not Reached Counter:** An administration tool allows to test if a certain document can be accessed. If a document cannot be read this counter is incremented. Another administration tool deletes bookmarks from the collection if the value of this counter is higher than a certain threshold. When a bookmark is added this counter is set to zero. If users open a bookmark and the resource cannot be accessed the counter is incremented too.
- **Valuation:** This float-type parameter is set to a default value (e.g. to 5) by the system when a bookmark is added. If users open the bookmark they may set this value again. The average of all entered values is calculated. The range is from 0 to 10. 10 means highest quality 0 is the lowest. This idea is taken from Juergen Horwarth's thesis [Hor99].
- **Valuation Counter:** Is an integer used to calculate the average of the Valuation property. It is incremented whenever a user enters a new Valuation.
- **Location:** This property offers additional information and indicates where the referenced document resides. It corresponds to the endpoint of the Dino-link. Note that this property is not used to locate the document but it is an additional information for users.

8.6.2 Folder Properties

A folder is a kind of directory containing bookmarks. There are personal folders, folders which can be accessed by a group of people and public ones. The following meta-data is stored with folders:

- **Title:** Describes the content of a folder. Similar to newsgroups, folder names should reflect the hierarchy of the bookmark collection. The name is set by the person who adds the new folder.
- **Creation Date:** Is set by the system when the folder is added.
- **Author:** This String contains the name of the user or administrator who added the folder.
- **Contains new Bookmarks:** Whenever a new bookmark is added to a folder this boolean-type value is set to true. Administration tools reset this value if no more bookmarks with the new tag set remain in the folder.
- **Notification List:** Is a list of subscribed members of that folder. This list does not control access rights but it is used to notify users when new bookmarks are added. By subscribing and unsubscribing users may add and remove themselves from these lists.

8.7 The Bookmark Server

The server-side part of Shared Bookmarks stores the bookmark collection, and allows administrators and sub-administrators to manage the collection. Generally folders are created by an administrator who sets the users' access rights using the Dino user-manager. [Hau99] Sub-folders should contain Bookmarks that are somehow related to the parent folder.

For administrators an administration panel is provided: The Dino Explorer is opened showing the whole bookmark tree. Additional menu-entries have been added to the Dino Explorer to start and configure administration tools.

The Bookmark Server provides the following functionalities:

- **Add New Folder:** Allows one to add a new folder. After choosing the location the required properties have to be added. Additionally the user-manager can be called to specify access rights.
- **Edit Folder Properties:** The administrator may use this method to edit the folders' meta-data.

- **Remove New Tags:** If a bookmark is added it's New Tag is set to true. When this function is run the administrator enters a date and the new tag of bookmarks is set to false if the bookmark is older than the specified date.
- **Search For Obsolete Bookmarks:** This tool marks all bookmarks as "old" which have been added before a certain date. It is possible to exclude the following bookmarks from this search:
  - bookmarks with a high priority
  - bookmarks that are visited frequently
  - bookmarks with a high rating or a high valuation
  - bookmarks that have been visited a short time ago
- **Delete Obsolete Bookmarks:** In a second step all bookmarks with an Obsolete Tag set to true are deleted. If a bookmark is deleted the user who created the bookmark is notified via e-mail.
- **Test Bookmarks:** This function tries to access the document. If this it not possible (e.g. if the server is unreachable) the Not Reached Counter is incremented. If this counter reaches a defined level the bookmark is marked as old and deleted when the Delete Obsolete Bookmarks function is called the next time.
- **Load Local:** If a bookmark is accessed very frequently the administrator may store a copy of the referenced document to a local cache.
- **Search:** The administrator may search for one or more matching properties in the collection.
- **Sent Notification:** If this function is called one single summary containing the changes is generated and sent via e-mail to all users who want to be notified at this time.
- **Add Bookmark:** This function is not a tool for the administrator but an interface for applications which want to add new bookmarks. If a bookmark is added the target folder has to be specified: In the interactive case a dialog is shown that allows us to choose a folder and enter the new folder's properties. The non-interactive case takes the parent-folder and the properties as parameters.

All administrative functions described above can be run in the background (e.g. during the night). It is possible to auto-run these functions at a specific time, in the whole bookmark collection, a branch or just in one specific Folder.

8.8 The Bookmark Client

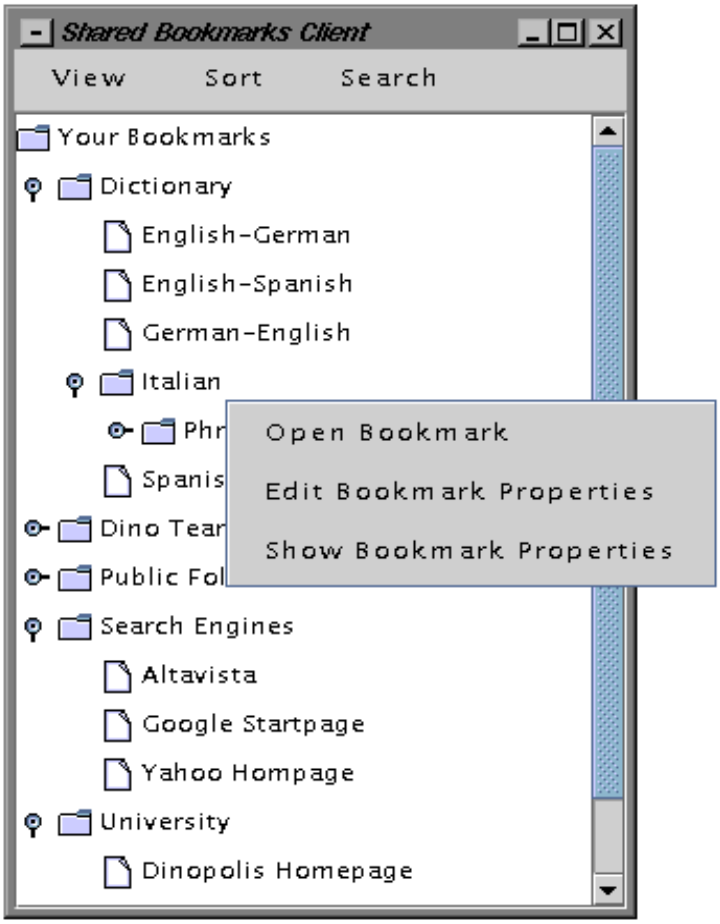

Figure 8.2: The Bookmark Client User Interface

The Shared Bookmarks client can be run as an applet and as application. Furthermore, the client can be run local which means that the server runs on the same local machine. All user actions (e.g. open a bookmark or enter a bookmark evaluation) are forwarded to the bookmark server. Currently the client does not store any data on the client machine. By Clicking the right mouse-button users may edit and view the bookmark's properties. The "View" menu allows to choose among various filters. For example, the user may use a filter which shows only new bookmarks.

The following list shows the functionality of the bookmark client:

- **Open Bookmark:** When users browse the bookmark collection they can open bookmarks by clicking the left mouse button. If he clicks the right mouse button a popup menu is displayed which again contains the Open Bookmark function. When a bookmark is opened the content of the target document is sent to an application running on the clients machine, according to the MIME-type of the document. The current implementation opens the document in the default browser.
- **View/Edit Properties:** Users can view and edit the properties of bookmarks and folders.
- **Filters:** Various filters only show those bookmarks which match certain characteristics: These filters can be combined which means that one can specify more than one filter at the same time. Filters are selected via checkboxes and appropriate dialogs allow one to enter the filter parameters:
  - **New Bookmarks:** Only bookmarks with the New Tag set are shown.
  - **Date:** The user can specify a date and only bookmarks that have been added after this date are shown.
  - **Rating:** The Rating property is used to determine if a bookmark is shown in the tree or not.
  - **Valuation:** The Valuation property is used as a threshold.
- **Sort Bookmarks:** It is possible to sort out the bookmarks depending on various properties. Sorting is only possible within one folder.
- **Search Bookmarks:** Shared Bookmarks allows one to search the bookmark collection for different properties. (e.g. keywords or creation date)
- **Notification Interval:** This function allows users to specify how often they want to get notified via e-mail if new bookmarks have been added.
- **Propose New Folder:** If users wish to add a new folder they enter the folder's name, a short description and the parent folder. The responsible administrator receives an e-mail and decides if the folder is added or not.

8.8.1 Modularity

The bookmark client is split up in three main modules. First the BookmarkClient class, which is the base-class, can be run as an applet and as

application. It is used to obtain a reference to the local browser and to open documents in the browser. The BookmarkClient instantiates the GUI which is the second main part described below. Finally, a module called BookmarkDataHandler class requests the bookmark data. Filters, sorting and searching is implemented in this class.

8.8.2 Graphical User Interface

As already mentioned Shared Bookmarks use Java Swing for the Graphical User Interface (GUI). The main components of the GUI can be summarized as follows:

- **BookmarkScrollPane:** This pane is either placed into the applet's frame or the applications frame.
- **BookmarkTree:** Is the graphical representation of the bookmark collection. The nodes and leaves of the tree are BookmarkTreeNodes.
- **BookmarkTreeNode:** BookmarkTreeNodes may either be a folder or a bookmark. A pop-up menu is opened if the right mouse button is clicked.
- **BookmarkMenuBar:** The menu bar is added to the frame and allows one to choose filters or to search the collection.
- **BoomarkPopupMenu:** Is the pop-up menu which is shown when the user clicks on a BookmarkTreeNode. Which methods are called by the corresponding action listener depends on whether the BookmarkTreeNode is a folder or a bookmark.

8.9 E-mail Notification

If the administrator starts the Send Notification tool the bookmark collection is searched for bookmarks which have been added since the last time this tool was run. An e-mail is sent to those users that are subscribed to the corresponding folder. Depending on the specified notification interval one single e-mail containing a list of new bookmarks is sent to the users. The notification interval is stored in the user record together with the user's e-mail address.

8.9.1 Java Mail API

The Java Mail API allows programmers to enable their applications for e-mail messaging. Basically it supports the common e-mail protocols such as POP, SMTP and IMAP. Other protocols may be supported by implementing new provider classes. Shared Bookmarks use the POP3 and the SMTP protocol. The API handles the said protocols and allows to generate and parse messages in a format according to the standard for the format of ARPA Internet text messages (RFC 822). [NH99] [Cro82]

Since the API consists of a big number of classes we will concentrate on the most important ones of the core API, which include *javax.mail.Session*, *javax.mail.Store*, *javax.mail.Transport*, *javax.mail.Folder*, and *javax.mail.Message*.[NH99]

The *Session* class represents the top-level entry point to the Java Mail API. Its methods allow one to load and control the Service Provider Implementations (SPI) for various protocols like POP3 or SMTP. At the moment implementations for POP3, SMTP and IMAP4 are available from Sun and others may be obtained from third parties. A service provider implements the *Store* class for a particular protocol which allows the programmer to access the protocol for read, write, monitor and search. Another class implemented by the service provider is the *Transport* class which is used for sending messages over a specific protocol. The third main class implemented by a provider is the *Folder* class; it gives hierarchical organization to mail messages and provides access to the *Message* class. (e.g. to the message objects). The *Message* class is again implemented by a service provider and handles all the details of the e-mail message, such as the subject, sender, recipient, sent data and many more.[NH99]

The Java Mail API makes use of the Java Activation Framework (JAF). It is used to manage the multitude of data formats such as images, audio or video objects which can be contained in e-mails.[NH99]

Shared Bookmarks use a Java Mail ListServer implementation which simply adds and removes users to and from the notification list of a specific folder. Changes in a folder like new bookmarks or if a folder location changes are sent to the subscribed users by the administrator via the ListServer.

8.10 Configuration

When the bookmark server is started it reads a configuration file which allows us to specify the server location, ports an so on. The state of the server is stored in a status file.

Detail: This status file stores, among other things, the location of the

bookmark collection's root folder and a counter which allows us to bookmark the same document several times.

8.11 Interface to other Applications

Shared Bookmarks provides an interface to other applications so that they can, for instance, add bookmarks. In the Dino Shell for example I added the commands "createbookmark" and "createfolder". These commands call the corresponding methods of the Shared Bookmarks' interface.

On the other hand Shared Bookmarks need to "send" documents to different applications when the users open a bookmark. Depending on the document's MIME type the content is either sent to a browser (which is currently the default implementation) or to another application (e.g. an image viewer). Currently Shared Bookmarks can be used from Dino applications (e.g. the Dino Shell). How to integrate Shared Bookmarks into other applications will be part of future work. (see chapter 9 for more details)

Chapter 9

Summary and Outlook

We have seen that applications running on a heterogeneous information system have to deal with various resources. These resources may reside on the local machine or anywhere in the Internet and are accessed using different protocols. One possibility is to implement the protocols needed in the applications itself which means that if other applications exist which want to access these resources, they would have to implement these protocols as well. Therefore, it is better to implement protocol handlers only once and make these implementations available to applications. A middleware system, like Dino, is the appropriate place for common protocol implementations. Furthermore applications may access external resources through a uniform interface which shields the programmer from protocol-specific programming. If an application needs a new protocol which has not yet been implemented, the protocol is added to the underlying middleware system.

The functionality of Shared Bookmarks depends heavily on the addressing and linking mechanisms of the underlying system. When running in an open environment like the WWW the information content provided by the application will be reduced (e.g. by dead hyperlinks). Link-typing, multi-endpoint relations and linking to links increases the functionality since it allows the representation of complex relationships among objects.

Distributed object systems extend the object-oriented approach to network applications. With some basic restrictions object-oriented software design for network applications can be done as if the programs would execute on the same machine.

Multi-user environments require an advanced user-management and various security features like a fine-grained access control mechanism or secure network connections. Application programmers should not have to implement user-management or security in their applications but the underlying

middleware system should provide them.

This is in fact the philosophy of Dino. Access to the information system on a high degree of abstraction and transparency provides an ideal platform for network applications. The open design and the extensibility allows to modify and extend the system as applications evolve. Improvements added to the middleware system might enhance other applications as well.

Future work on this project encloses the development of new features such as a recommendation mechanism for bookmarks, new filters, extended search facilities or secure e-mail messaging for notification and subscribing. Moreover, Shared Bookmarks should permit one to convert and upload existing local bookmark collections and vice versa.

One task will be a better integration into applications like the Dino Shell (Dish), the Dino Explorer or non-Dino applications like Netscape Navigator.

Alternative representations for different purposes like a low-end representation with JavaScript or an integration in Netscape Bookmarks Menu will be considered.

An extended testing period in a multi-user environment will be followed by a reconsideration of the present design in order to increase performance, functionality and usability of Shared Bookmarks.

A lot of this work will concentrate on improvements and extensions of the Dino middleware system. The integration of new protocols by providing new drivers and semantic handlers as well as new Views together with new link-types and new security features will enhance the functionality of Shared Bookmarks and of the whole platform.

The Dino system supports new technologies like the Extensible Markup Language (XML) or Java RMI and CORBA. XML together with its link language XLL and its addressing specification called XPointer will improve addressing on a document level and XML parsers allow the mapping of the internal structure of documents to the Document Object Model (DOM).

Both, Java RMI and CORBA, remain useful since they are not competing systems but each of them provide features which the other does not. In a heterogeneous system made up of components in languages other than Java CORBA will be applied. Systems partially written in Java and other languages use a hybrid from RMI and CORBA together with the Internet Inter ORB Protocol (IIOP) which can be used by both technologies and can be seen as the the bridge between Java RMI and CORBA.

Even in 100% pure Java applications, there is still a place for CORBA. RMI is currently not capable of handling massively distributed systems. Services such as transaction management and replication are a requirement of very large distributed object solutions. RMI will support these services

in the future and simultaneous use of both frameworks will allow us to build more evolved applications on heterogeneous information systems.

Security is important for Shared Bookmarks since it is thought to be used in security critical environments like in health care systems. Security mechanisms can be applied if necessary and are omitted if the environment does not demand them.

All these technologies are integrated into the Dino system in a way that they can be replaced by new, improved technologies. For some time-critical modules own techniques like the proprietary Dino-Dino protocol are used to reach better performance and functionality.

Finally, related projects like the ones listed in section 2.6 will be observed and new, intelligent solutions will be compared with the present implementation.

Appendix A

Creating Signed Applets

We have seen that signed code, together with an appropriate security policy, may gain additional permissions on a remote system. Now let us have a look how to create a signed applet.

The first thing to do is to put the Java class file(s) (e.g. Test.class) into a JAR file using the jar command. Next we have to generate a key pair and a certificate:

```
keytool -genkey -alias DinoApplets
        -keypass aaa2bbb
        -keystore mystore
        -storepass ccc3ddd
```

This command creates a key pair aliased with "DinoApplets" and stores them in a keystore called "mystore". The -keypass argument protects the private key and the -storepass argument the keystore itself.

In addition, this command creates a self-signed certificate. The user is prompted for his name and address. The certificate contains the public key and the the additional information about the user. How to request a certificate at a Certification Authority is described later. Next we sign the JAR file with the private key called DinoApplets:

```
jarsigner -keystore mystore
          -signedjar SignedTest.jar Test.jar DinoApplets
```

The result is the SignedTest.jar file which already contains the certificate.

The receiver has to possess the certificate in advance. Therefore the certificate is exported and sent to the receiver:

```
keytool -export
        -keystore mystore
        -alias DinoApplets
        -file DinoApplets.cer
```

The returned certificate is called DinoApplets.cer:

```
Owner:  CN=Philipp Zambelli, OU=IICM, O=TU-Graz, L=Graz, ST=Austria, C=AT
Issuer: CN=Philipp Zambelli, OU=IICM, O=TU-Graz, L=Graz, ST=Austria, C=AT

Serial number: 383e5050

Valid from:    Fri Nov 26 10:18:08 GMT+01:00 1999
      until:   Thu Feb 24 10:18:08 GMT+01:00 2000

Certificate fingerprints:
         MD5:  16:CD:12:D5:66:46:7F:7D:05:B9:E6:CA:E2:66:9E:7B
         SHA1: 27:D9:93:8C:56:A3:33:C9:75:5F:ED:43:4D:44:36:E6:D0:52:D9:09
```

Next the certificate is sent (e.g. via e-mail) to the receiving party.

The receiver again uses the keytool with the -import argument to store the certificate in his own keystore.

Next the receiver has to define the appropriate permissions in his security policy as shown in section 7.7.2 and the applet can be downloaded and run.

A.1 Request a Certificate at a CA

Self-signed certificates are not accepted by everyone so that you would have to request a certificate form a Certification Authority (CA). Therefore you generate a certificate signing request:

```
keytool -certreq
        -keystore mystore
        -alias DinoApplets
        -file DinoApplets.request
```

This request is sent to a CA (e.g. VeriSign[1]) and the CA will check your identity according to their verification strategy and return a signed certificate authenticating your public key. The returned certificate may contain a certificate chain where the certificate of the CA you contacted is confirmed by another CA and so on.

To import the reply we use the keytool again:

```
keytool -import -trustcacerts
        -keystore mystore
        -alias DinoApplets
        -file DinoApplets.reply
```

For more details on signing applets please see the Java Security Tutorial [SM99] or the Java Security Specification [Inc].

[1] Verisign: `http://www.verisign.com/`

References

[BLMM94] T. Berners-Lee, L. Masinter, and M. McCahill. *Uniform Resource Locators*, December 1994. `http://www.w3.org/Addressing/rfc1738.txt`.

[Bru91] Charles Bruner. *Thinking collaboratively: Ten questions and answers to help policy makers improve children's services.* Education and Human Services Consortium., Washington DC, April 1991.

[Cob98] Ed Cobb. *CORBA Firewall and Security*, 1998. `http://www.omg.org/news/pr98/compnent.html`.

[Col97] David Coleman. *Choosing Collaborative Tools*, 1997. `http://www.collaborate.com/hot_tip/tip0497.html`.

[Cro82] David H. Crocker. *Standard for the Format of the ARPA Internet Text Messages*, 1982. `http://www.faqs.org/rfcs/rfc822.html`.

[Dal99] Christof Dallermassl. *Aspects of Integration of Heterogenous Server Systems in Intranets - The Java Approach.* Master's thesis, IICM, Graz University of Technology, December 1999.

[DS95] Jeffrey R. Van Dyke and Karen R. Sollins. Linking in a global information system. In *World Wide Web Jornal - A Publication of the W3C*, page 493. World Wide Web Consortium, O'Reilly, December 1995. `http://ana-www.lcs.mit.edu/anaweb/pdf-papers/tr-659.pdf`.

[dT99] The Dino Team. *Dino Software Engineering Document.* IICM, Graz University of Technology, 1999.

[Gre99] Eric Greenberg. *Network Application Frameworks Design and Architecture.* Addison Wesley, 1999.

[GS96] Simson Garfinkel and Gene Spafford. *Practical Unix and Internet Security*. O'Reilly & Associates, Inc., 1996.

[Hau99] Heimo Haub. *Aspects of Access Management in Heterogenous Distributed Object Systems*. Master's thesis, IICM, Graz University of Technology, December 1999.

[Hor99] Juergen Horwarth. *Personalised Recommender System*. Master's thesis, IICM, Graz University of Technology, 1999. `http://www.iicm.edu/thesis/koch`.

[Inc] Sun Mircrosystems Inc. *Java Security Specification*. `http://java.sun.com/products/jdk/1.2/docs/guide/security/index.html`.

[Jaw96] Jamie Jaworski. *Java Developer's Guide*. Sams, Indianapolis, first edition, 1996.

[KK97] Stefan Kelm and Klaus-Peter Kossakowski. Kryptographische anwendungen in offenen netzen. In *Deutscher Internet Kongress 1997*. DFN-CERT, dpunkt Verlag, 1997.

[Koc98] Thomas Koch. Groupware. Master's thesis, IICM, Graz University of Technology, 1998.

[Kun95] J. Kunze. *Functional Recommendations for Internet Resource Locators, Network Working Group RFC 1736*, February 1995. `ftp://ds.internic.net/rfc/rfc1736.txt`.

[Kya94] Othmar Kyas. *Internet Zugang-Utilities-Nutzung*. Datacom, 1994.

[Man98] Frank Manola. *Technical Committee H7 Object Model Features Matrix*, 1998. `http://www.objs.com/x3h7/fmindex.htm`.

[MD98] Eve Maler and Steve DeRose. *XML Linking Language (XLink) World Wide Web Consortium Working Draft*, 1998. `http://www.w3.org/TR/NOTE-xlink-req/`.

[MD99] Eve Maler and Steve DeRose. *XML Pointer Language (XPointer) World Wide Web Consortium Working Draft*, 1999. `http://www.w3.org/TR/1998/WD-xptr-19980303`.

[Mic98] Sun Microsystems. *Java Remote Method Invocation Specification*, 1998. `http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.h\%tml`.

[Moc87] P. Mockapetris. *Domain Names - Implementation and Specification*, 1987.

[NH99] Tony Nemil and Anil Hemrajani. *Introduction to the Java Mail API.* Technical report, Sun Microsystems, 1999. `http://www.javaworld.com/javaworld/jw-06-1999/jw-06-javamail.html`.

[RSL99] V. Ryan, S. Seligman, and R. Lee. *Schema for Representing Java Objects in an LDAP Directory*, October 1999. `http://www.ietf.org/rfc/rfc2713.txt`.

[Sam95] Sams. *The Internet unleashed.* Sams, second edition, 1995.

[SBGK94] Matin Scheller, Klaus-Peter Boden, Andreas Geenen, and Joachim Kampermann. *Internet: Werkzeuge und Dienste.* Springer-Verlag, 1994.

[SM96] Alan R. Simon and William Marion. *Workgroup Computing.* McGraw-Hill, 1996.

[SM99] Inc. Sun Mircrosystems. *The Java Security Tutorial*, 1999. `http://java.sun.com/docs/books/tutorial/security1.2`.

[SSF97] Richard L. Shuey, David L. Spooner, and Ophir Frieder. *The Architecture of Distributed Computer Systems.* Addison-Wesley, 1997.

[Van96] Glenn Vanderburg. *Tricks of the Java Programming Gurus.* Sams, 1996.

[Wul97] Volker Wulf. *Konfliktmanagement bei Groupware.* Vieweg, 1997.

[Zue99] ETH Zuerich. *Internet Videokonferenzen*, 1999. `http://www.iha.bepr.ethz.ch/pages/support/meth/videoconf/overview.htm\%`.

Index

X