Vrije Universiteit Amsterdam
Department of Computer Science
Warsaw University
Faculty of Mathematics, Informatics and Mechanics

Securely Replicated Web Documents

Jan Sacha

Master's Thesis
COMPUTER SCIENCE

Supervisors

Prof. dr. Andrew Tanenbaum
Department of Computer Science,
Vrije Universiteit Amsterdam

Dr. Janina Mincer-Daszkiewicz
Faculty of Mathematics, Informatics
and Mechanics, Warsaw University

July 2003

Abstract

The WWW is experiencing explosive growth and an increasing number of security-sensitive applications make now use of it. To achieve world-wide scalability and reduce latency in handling user requests, many of these applications make extensive use of data replication through caches and Content Delivery Networks. However, such replication mechanisms place data on untrusted hosts, which introduces various security problems. In this thesis we present an architecture that combines data content, replication strategies and security in one unified object model and offers integrity guarantees for web documents replicated on non-secure servers.

Keywords

World-Wide Web, Distributed Systems Security, Object-Based Web Model, Document Replication, Performance Measurements, Globe, GlobeDoc

Contents

List of Figures

Introduction

The World-Wide Web continues to expand at an amazing pace, in terms of both the numbers of sites and users. For the most popular sites the number of users exceeds millions, the servers are often unable to cope with a very high load. In result the access time is excessively long or the servers are not reachable at all.

Various solutions are proposed, mostly based on **caching** and **replication**. One common replication technique is known as CDN (Content Delivery Network), where the websites are propagated in a large network of servers spread across the world. The network is owned and maintained by one company, which does not publish any documents by itself, but supplies the infrastructure for the documents storage and replication.

One drawback of such an architecture is that the CDN itself needs to be trusted, since the web documents are fully controlled by the replicating hosts. Neither the content owners nor the clients have any direct control over these hosts.

Similar situation occurs when the clients use web proxies or mirror sites. It is not possible to verify whether the downloaded documents are authentic or fake. A malicious host of a replica can install an invalid version of a website instead of the original one. The owner may be able to check if the page is correct, but cannot guarantee any protection in advance.

In this thesis we describe a secure model for the web where the documents are massively replicated on **untrusted hosts**. The documents can be moved to any server that agrees to cooperate, similarly as in the peer-to-peer networks. Extending possible locations for the websites to all available hosts can greatly increase scalability and the immense potential of the Internet can be utilized more efficiently.

We consider all intermediate hosts and network devices between the owner of a document and the end-user as insecure. Our system prevents the replicas from violating the integrity of the documents and provides a secure naming mechanism for identifying resources of the web. Most importantly, we offer unified object model for web documents, which combines the data content, replication mechanisms and security policies.

The main contribution of this thesis is a prototype implementation for the security model described in [14]. It is based on the previous insecure prototype, extended by the security mechanisms.

A variety of measurements were performed on the implementation. The results of the measurements are included in this document and are an important part of the thesis. Our conclusions are based on the measurement results.

The rest of this thesis is organized as follows. In chapter 2 we discuss related work on web security. Chapter 3 gives a short description of the Globe system. It is necessary to understand chapter 4, where we explain the design of our new web model, with special emphasis put on security issues. In chapter 5 we describe our prototype implementation The performance measurements results are presented in chapter 6. In chapter 7 we conclude.

Chapter 1

Security of the World Wide Web

The World Wide Web, despite its extraordinary popularity, still runs mostly in the insecure mode. Examples of breaking into famous web servers are numerous, mass-media periodically announce million-dollar losses caused by the attacks of so called crackers. In this chapter we briefly discuss the most common measures applied to increase the security of the web and show scenarios of potential attacks.

1.0.1. DNS

One of the weakest points of the current web is the naming. In most cases it is provided by the **DNS (Domain Name System)** [10, 11]. The DNS was designed about 20 years ago, when the security of the Internet was not yet taken into consideration. As a result the DNS does not contain almost any explicit protection against abuse, many attacks on the websites are based on vulnerability of the DNS.

A typical technique is called **DNS spoofing**. It exploits lack of authentication in the DNS protocol. The attacker manipulates with messages sent to the name server in such a way that incorrect data is placed into the cache. The clients querying the server receive answers prepared by the attacker. This way it is possible to redirect the traffic from a website to any other location, for example to its fake copy. The victims are unaware of the redirection, the original website is cut off from the clients.

DNS spoofing is illustrated on picture 1.1. More detailed descriptions of breaking security of the DNS can be found in [18].

1.0.2. DNSsec

As a replacement for the DNS, a new system was proposed — the **DNSsec (Secure DNS)** [4]. DNSsec uses public-key cryptography to guarantee authenticity and integrity of propagated information. It's primary purpose is the name resolution, but it can also be used to store public keys and other auxiliary data. Similarly to the traditional DNS, it is based on zones of authority. Each zone is responsible for its own domains and maintains its own public/private key pair. Every record stored by the DNSsec server is cryptographically hashed and signed by the zone's public key. This enables clients to verify that the records are valid. The public key of the zone can be acquired from the higher-level zone. The top-level zone's public key is assumed to be world-wide known, it can for example be included to the software querying the DNSsec.

The DNSsec provides secure naming, protects from DNS spoofing, but does not guarantee full security for the web. A problem occurs when a client resolves a name to an address

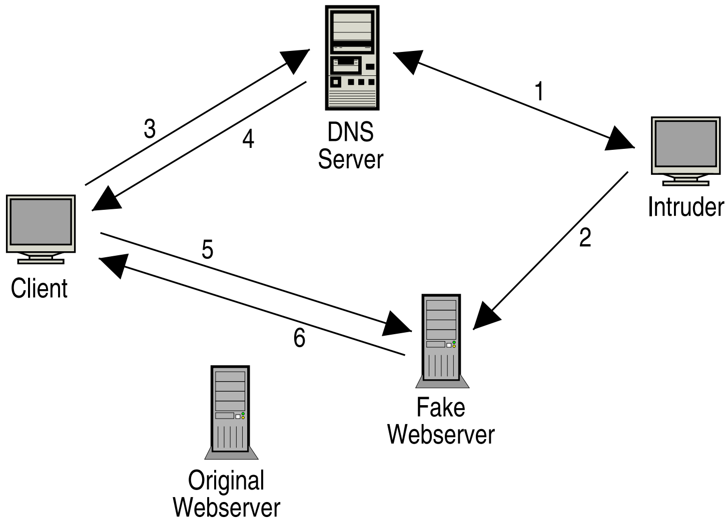

Figure 1.1: DNS spoofing.

The intruder poisons the DNS cache (1) and sets up a fake website (2). Unaware client resolves a name in the DNS server (3), receives a reply prepared by the attacker (4), connects to the fake web server (5) and downloads an incorrect webpage (6).

and connects to the server. This is normally accomplished with the IP protocol, which does not include any encryption or authentication procedures. Even if the name of the server is correctly resolved there is no way to detect an intruder between the client and the server.

Such a situation is called a **man-in-the-middle** attack. The intruder intercepts the connection between two hosts and reads all transferred data. In a worse scenario she can also modify it, so that the client and the server receive false information.

Authentication is required even if both sides use encrypted communication. Otherwise the client may establish connection with intruder instead of with the server. The intruder can negotiate the encryption parameters, for example the shared session key, pretending to be the server. As a result the client may expect to set up a secure connection but in fact all the data is disclosed by the attacker. An example of such situation is demonstrated in Figure 1.2.

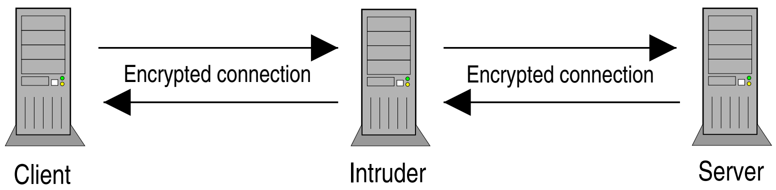

Figure 1.2: The man-in-the-middle attack.

Both the client and the server use encrypted channels, but the connection was set up with an intruder in the middle. All transferred data is under control of the attacker.

1.0.3. HTTPS

To improve the security of the web the **SSL (Secure Socket Layer)** standard was introduced, extended later to **TLS (Transport Layer Security)** [3]. The TLS provides mutual authentication of the client and the server, secret communication and data integrity protection. The two latter goals are achieved by encrypting the network traffic with algorithms negotiated by the communicating peers. The authentication is based on the public-key cryptography. In practice the TLS offers a secure method of communication through an untrusted network, in particular through the Internet. HTTP connection over the TLS is called **HTTPS (Secure HTTP)**.

In order to set up a secure connection the client needs to know the server's public key. There are several ways in which the public keys can be obtained. One way is to use DNSsec. Another way is to transfer the key from a real-world organization to the user on a physical data store, for example on a CD-ROM. The most popular method, however, is to use certificates. Every server publishes a certificate containing the server's public key. The certificate is signed by a **Certificate Authority (CA)**, that is assumed to be trusted by all users. The CA guarantees a certificate is correct. The public key of the CA is widely-known, usually being delivered together with the browser software.

HTTPS in combination with DNSsec and certificate authorities can provide high security for the web.

However, it can be shown that DNSsec does not work well with dynamically replicated websites. Its efficiency and scalability relies on caching. When documents actively migrate between different locations it exhibits poor performance. Moreover, the recent web model does not support replication on untrusted hosts. Usually mirrors are controlled by the same or related organizations and the security is based on the mutual trust. In the CDN all the replicas are trusted.

In our model we consider **dynamic document replication** on **untrusted hosts**. As such assumptions are innovative, a new security architecture was designed.

Chapter 2

The Globe System

2.1. General Model

Our new model of the web is built on top of the Globe [20], a wide-area object-based distributed system. The Globe is implemented as a middleware, it's a framework for our application. Its basic principles are outlined in this chapter.

2.1.1. Distributed Shared Objects

The most distinctive feature of the Globe system is transparent replication. User processes interact and communicate through **distributed shared objects (DSO)**. The objects are physically distributed, their state can be partitioned across several machines. The processes are unaware of the distribution. All the internal communication and synchronization is handled by the middleware.

When a process accesses an object, it first constructs a **local object**. A local object resides in the local address space of the process and implements all the mechanisms needed to perform operations on the distributed object. A local object can be a mere proxy forwarding all requests to other replicas, an RPC-style server accepting connections from other hosts, or any other object performing arbitrarily complex distributed algorithm. A sample DSO consisting of three nodes is illustrated on picture 2.1.

The local objects are divided into five **subobjects**, as shown in Figure 2.2. The subobjects are reusable, each of them encapsulates part of the local object's functionality. The semantic subobject is application dependent, it implements the semantic operations of the DSO ignoring concurrency and distribution. The remaining four subobjects are responsible for communication, replication, synchronization and security. The subobjects define object's **policies**, in particular replication policy and security policy. In the thesis we consider mostly the security subobject.

2.1.2. The Globe Services

The Globe system provides services for naming and locating objects. The **Naming Service** resolves object names to **Object Identifiers (OID)**. Object names are usually human friendly, meaningful and reflect the administrative structure. OIDs are computer generated and in contrary to the names are never changed. Every object is assigned to exactly one OID unique in the system. The **Location Service** resolves the OID to the current object address and contact protocol. By separating services for naming and locating the system is more

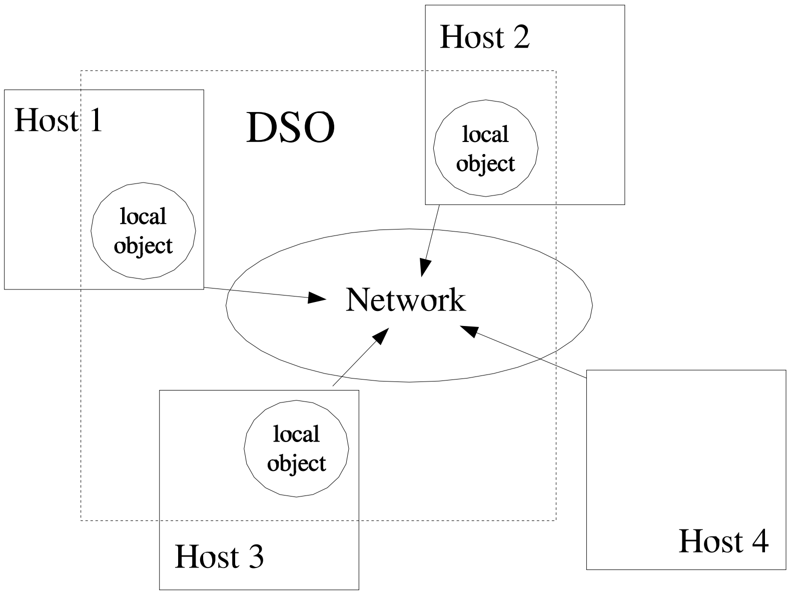

Figure 2.1: The Globe Distributed Shared Object.

The DSO consists of three local objects and is physically distributed on hosts 1, 2 and 3. Host 4 does not access the object.

flexible, supports dynamic object migration with a scalable and convenient naming scheme. This topic is discussed in more detail in [19].

Object replicas which contain part of the object's state are located on **Object Servers**. The servers contain persistent storage for passivation and activation of objects. Object servers support dynamic replication.

2.1.3. Binding

In order to operate on a DSO, every process must **bind** to the object first. The result of binding is a local object placed into the process's address space. As it is explained later, binding is one of the most critical operations for the security of a distributed system. In the Globe it consists of several steps, as presented in Figure 2.3. The process resolves an object name in the Naming Service and the OID in the Location Service. Next the process selects one of the contact addresses and downloads an implementation for the local object from the **Implementation Repository**. When the local object is installed and initialized the process can contact the DSO.

2.2. The Globe Security

The security of the Globe system is based on public/private-key algorithms. Every user, object and replica maintains its own public and private key pair. Access control is implemented with certificates, user's permissions are cryptographically sealed in certificates signed by the object owner. Every entity in the system can verify if a certificate is valid, since only the owner of the private key can release a correct certificate. Such solution is fully distributed, it does not

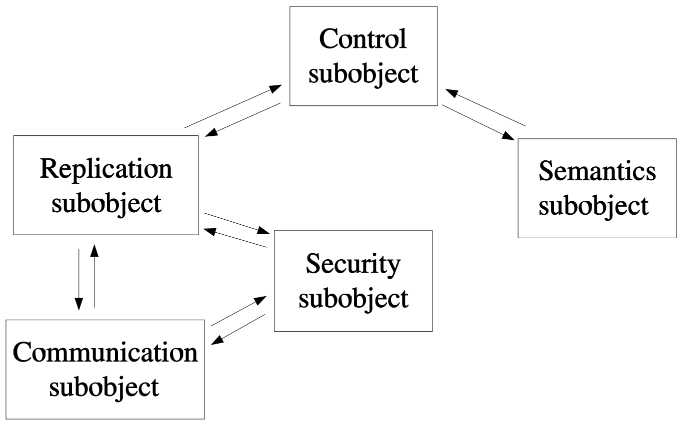

Figure 2.2: The structure of a local object.

require any central authority in the system. Certificates can form chains if one user delegates rights for issuing new certificates to another user. This design is described in [17].

The Globe utilizes certificates also for **Reverse Access Control** [13]. With this mechanism a user can assign permissions for every replica and decide which operations the replica is allowed to perform. The execution of the most critical object's methods can be limited to chosen, most trusted replicas.

As shown in [9], the Globe security design is powerful and general. The web model considered in this paper is much more specialized. Instead of providing a flexible framework for distributed applications it concentrates on web documents only. The requirements for the web are therefore much more specific.

First, the users are not required to authenticate themselves, we assume the documents are available for everyone. For the same reason there is no need to encrypt transmission as all the data is public. Second, we treat all replicas as untrusted. It implies that there is no need to authenticate replicas or apply reverse access control. The documents can be stored on any host. The users are guaranteed though that the documents are not tampered with by any malicious replica.

For our web model we specify new security requirements and a new security design. We do not use the standard Globe security facilities.

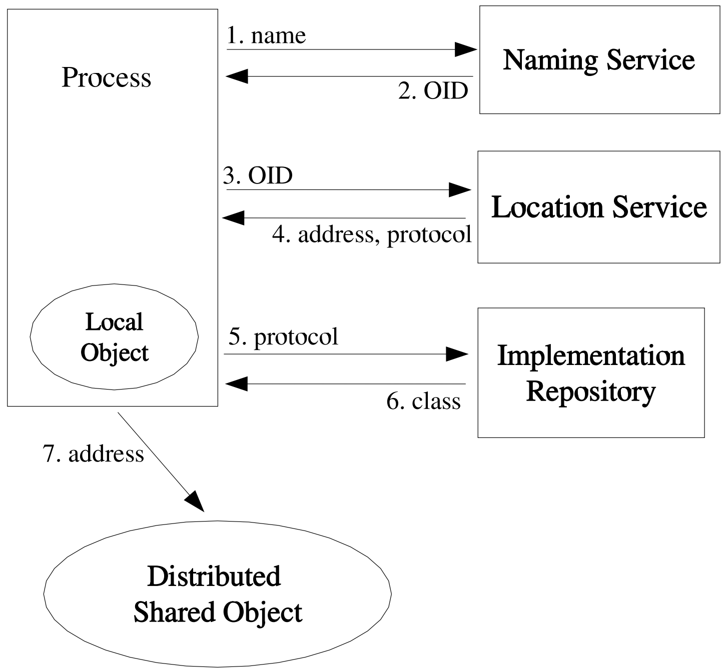

Figure 2.3: Object binding in Globe.

The process resolves an object name in the Naming Service (1) to the object identifier (2). Next, the OID is sent to the Location Service (3) and resolved to the current object's address and contact protocol (4). Subsequently, after address and protocol selection, the process downloads a class implementing the chosen protocol from the Implementation Repository (5,6). Finally, the process installs a local object in its own address space and connects to the DSO at the selected address.

Chapter 3

GlobeDoc — Globe for the Web

Our web model is named GlobeDoc. It is functionally comparable to the WWW, its primary goal is to provide uniform on-line access to distributed documents. It differs in the internal organization, especially in the replication scheme.

GlobeDoc is built on top of the Globe system. It adopts a Globe replication mechanism to the web environment. As discussed in [8] the model is more scalable then other widely applied solutions and can be better adjusted to specific web requirements.

In this thesis we focus mostly on the security aspects of the GlobeDoc. First though, we describe the overall architecture.

3.1. General model

A fundamental assumption of the GlobeDoc is that the **documents** are encapsulated in Globe distributed shared objects. An object contains the document's state and provides an interface to access the document's content.

The GlobeDoc object consists of several closely related files (text, images, video, etc.), forming one structure, usually belonging to one organization and under one administration unit. It's not defined precisely how these files should be grouped into one object, it depends on the particular website. The files belonging to one document are called **page elements**. An example of GlobeDoc objects is presented in Figure 3.1.

As a consequence of encapsulating documents in Globe objects **per-document replication policies** can be assigned. A policy defines how an object interacts with the system, in particular how it is replicated.

Let's consider the example of a popular software archive: it may be heavily replicated across multiple servers in order to decrease average download time. On the other hand, a rarely accessed page with a limited group of visitors should not be mirrored at all but can be actively pushed to the users whenever updated.

The replication, synchronization, communication and all the functionality necessary for the distribution is implemented in the Globe middleware. Therefore there is no need to develop new infrastructure for the GlobeDoc.

The documents are stored on Globe Object Servers. In the GlobeDoc terminology the servers are also called **stores**.

The stores can keep two types of document objects. The **primary replica** (master) provides full functionality of a Globe document, can handle both read and write (update) requests. **Secondary replicas** (slaves) can serve independently the read requests, but forward all update invocations to the primary replica to avoid inconsistencies.

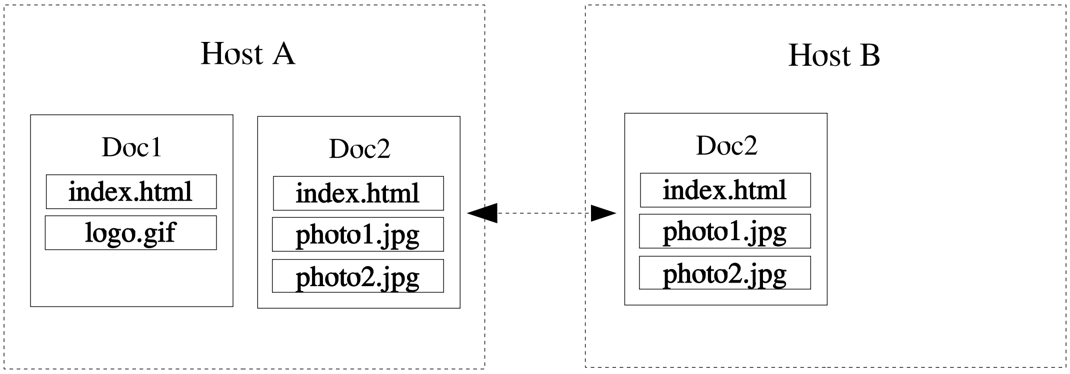

Figure 3.1: An example of a GlobeDoc object.

Document Doc1 consists of two elements: index.html and logo.gif. Doc2 contains three elements and is replicated on two hosts: A and B.

Apart of the object replicas, there are various client-side caches, for example in the user browsers, which are out of control of the stores.

3.2. Integration with the WWW

The GlobeDoc model is integrated with the current web so that GlobeDoc objects can be accessible through standard HTML browsers. To achieve this, clients need to run a special program on their machines called **GlobeDoc Proxy**. The proxy is a process working in the background, providing a standard HTTP proxy interface. When the client's browser is configured to use the proxy, the contents of GlobeDoc objects can be browsed as regular web pages.

Upon receiving a request from the browser, the proxy recognizes if the target document is a standard website located on a HTTP server or a GlobeDoc page element reference. In the former case, the proxy forwards the request to the server and returns the response to the client's browser. In the latter case, the proxy contacts a Globe object, retrieves the document's content and sends it back to the browser. Figure 3.3 shows this scenario.

Another issue is the way the proxy distinguishes between GlobeDoc names and HTTP URLs. Our solution is to embed GlobeDoc object names in HTTP URLs with a constant, configurable prefix, recognizable by the proxy. The URL syntax is presented in Figure 3.2.

The URL contains both the document name and the page element name, which are separated by a special predefined separator. For example the proxy can be configured to use a string *GLOBE* to embed all Globe names in URLs. With such configuration GlobeDoc object */nl/vu/cs/globe/jsacha* is accessible from a browser as *http://GLOBE/nl/vu/cs/globe/jsacha*. Element *MasterThesis* of this document is then specified as *http://GLOBE/nl/vu/cs/globe-/jsacha:/MasterThesis*.

Apart from recognizing URLs in arriving requests the proxy must also return correct URLs in the documents returned to the browser. For example a document may contain a link to a page *globe://nl/vu/cs/globe/jsacha* which cannot be interpreted by the browser. To overcome the problem the proxy has to translate the address to *http://GLOBE/nl/vu/cs/globe/jsacha*. It requires searching whole document's content for Globe URNs and translating them to

http://GLOBE-TAG/object/name:/element	
http://	Protocol name
GLOBE-TAG	Configurable GlobeDoc prefix
/object/name	Document (object) name
:/	Separator
element	Element name

Figure 3.2: GlobeDoc hybrid URLs.

HTTP URLs. This process contributes to higher delay in handling requests.

More detailed description of the GlobeDoc Proxy can be found in the chapter 4.4 of this thesis, where we discuss the implementation of the proxy.

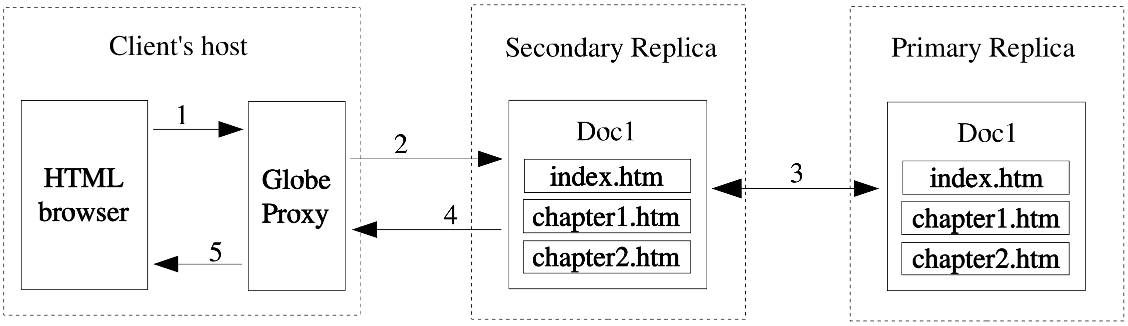

Figure 3.3: Request processing in GlobeDoc.

The browser sends a request for http://GLOBE/Doc1:/chapter2.htm (1). The Proxy translates the request to globe://Doc1:/chapter2.htm, contacts a nearby replica of object Doc1 and requests the element chapter2.htm (2). The secondary replica potentially contacts the primary replica, accordingly to the replication protocol (3) and returns element content to the proxy (4). Finally, the proxy translates all the Globe URNs found in the element's body and sends the data back to the browser (5).

3.3. GlobeDoc Security

The security design for the GlobeDoc architecture is described in [14]. We present it in the thesis though, as it is inherently required to describe the prototype implementation and to explain the results of the performance measurements.

3.3.1. Security Goals

The security objectives in the GlobeDoc can be divided into two categories: **secure naming** and protecting **document integrity**. Secure naming deals with establishing a secure and trustworthy association between the names of web resources and the real-world entities responsible for these resources. The system must guarantee, that when a user receives an OID after requesting a document belonging to an organization, the OID indeed points to the object of this organization. In other words, the system should prevent from supplying fake OIDs.

The secure naming boils down to two more specific requirements.

- Secure associating real-world entities with their public keys.
- Secure associating public keys with the object identifiers.

The document integrity requirement states, that a document received by the client must be identical to the latest version of the document issued by the author. None of the intermediate replicas, network devices or intruders of any kind should be able to tamper with the document without alerting the receiver. This strong requirement results from our assumption, that all the replicating hosts are in general untrusted and are a potential source of an attack. Similarly, all the network connections are considered to be insecure.

We do not require network traffic encryption. We assume, the document's content is always public, so user authentication is not an issue, however we provide a mechanism for verification, to detect any unauthorized alteration of the objects.

The document integrity requirement enforces the following three properties of the GlobeDoc objects.

- **Authenticity** — The document downloaded by the client from a server has been entirely created by the object's owner. No attacker or malicious host can add any part to the document that was not written by the owner.
- **Freshness** — The client always receives the latest published version of the document. Attacks based on replying old versions of the document are not possible.
- **Consistency** — The document received by the client is consistent with the document that she has requested. Any attacker is not able to remove, repeat or mix the content of a fresh document with another valid document.

3.3.2. Secure Naming

In the GlobeDoc model, each document has an **object owner** assigned — a user or organization who created the object and is responsible for keeping it up-to-date. Furthermore, every document object is associated with its own public/private key pair. The **private key**, for security reasons, is stored exclusively on the object owner's machine and is never transferred through the network to other hosts. The **public key** is kept in every replica of a GlobeDoc object and can be shown on request.

We believe, that secure associating public keys with real-world entities is a very complex and difficult task, which in a general case, without further assumptions, is infeasible. It's so security-sensitive, that still the safest known solution to is to leave it for the user to manage it personally. Here we can enumerate 4 methods how to securely associate public keys with real-world entities.

- Contact the entity physically and transport the key manually, for instance on a floppy disk.
- Obtain the key from trustworthy mass-media, for example newspapers.
- Use certificates signed by trusted Certificate Authority. The key of the CA is assumed to be well-known by everyone, it can be built-in in the applications.

- Store public keys in the DNSsec or in any other secure naming system.

In order to securely link object identifiers with object public keys we use the idea of **self-certifying OIDs**. A self-certifying OID is the **secure hash value** of the object's **public key**. The secure hash function guarantees that it is practically impossible (with exponentially decreasing probability) to find two different public keys which give the same hash value. With this property, we guarantee that the OIDs are securely bound to the object public keys. The mechanism does not require any Certificate Authority or any other globally trusted medium. There is also no overhead for additional communication or for performing any distributed algorithm.

The concept of self-certifying OIDs is quite simple and elegant, however, it has some disadvantages. A serious drawback is that whenever the public/private key pair is changed, the OID must also be changed. In the Globe system the names and the locations of objects can be modified but the OIDs are always constant. This implies the document's keys cannot be changed without changing the identity of the object.

3.3.3. Document Integrity

We have so far described how to securely resolve names, now we will show how we enforce document integrity. In the GlobeDoc we use a combination of **state signing** and integrity **certificates**.

Every document's object has to have a integrity certificate signed with the private key of the owner. The signature guarantees the certificate was indeed issued by the object owner and nobody could modify it, as the only person knowing the private key is the owner. Every party communicating with the object can request the certificate and verify its validity.

The certificate contains a **validity date** specifying the time period for which the certificate was released. After the given date the certificate is not valid any more and a new one must be issued. The validity date prevents from malicious or mistaken usage of old certificates, it ensures the freshness property for the GlobeDoc objects.

The certificate contains also the names of elements belonging to the document and the corresponding **secure hash values** of their contents. The hash values guarantee consistency of the document. Every change in any of the document's element can be detected be comparing the original hash value in the certificate with the current hash value of the examined element. It's not possible to add or remove any element from the document as all the element names are stated in the certificate.

With certificates containing all the information described above, the GlobeDoc model fulfills the three requirements of the document integrity. The authenticity and consistency is guaranteed by the signature and secure hash values of the elements. The freshness is enforced by the validity date.

The certificate structure is described in more detail in chapter 4.2, where we present the prototype implementation.

3.4. The Life-cycle of a Secure GlobeDoc object

Almost all the building-blocks of the GlobeDoc model have been already described. Here we present a document's life-cycle, a step-by-step chronological description of creating, publishing, updating and browsing for GlobeDoc objects. This process is illustrated also on picture 3.4.

First, the author of a document prepares the document's state. The state consists of all the static files that belong to the document.

Next, the author creates a GlobeDoc object, becoming the object's owner. The objects are created and manipulated by a program named **Tika**. The user never has to access the document's object directly, all operations are carried out by Tika. Comprehensive description of this program is given in chapter 4.3. In order to create an object, the user needs the document's content and a public/private key pair for the document. The keys can be supplied from an external keystore or can be generated by Tika. The newly instantiated empty primary replica is installed on a Globe Object server (store) and automatically registered with the Globe Location Service. The object's identifier is generated by Tika, so that it contains a secure hash value of the object's public key. This way the OID becomes a self-certifying OID. The OID can be registered in the Naming Service with any chosen names, allowing other user to refer to the document more conveniently.

At this moment the object is fully functional, but does not contain any data. To set the document's content, the owner again uses Tika. The program transfers the files to the server and updates the document's certificate. The user never has to manually compute elements hash values or modify the certificate in any way, although it is possible to control some certificate properties, like validity period.

The owner can also request placing new replicas on other object servers. New object instantiating and the state transfer is handled by the replicating servers.

The receivers of the documents, as it was mentioned before, use standard HTML browser and the secure GlobeDoc Proxy. In order to download a document, the user types a URL in the browser. The proxy retrieves the object's name from the URL and resolves it with the Naming Service to an OID. The OID in resolved with the Location Service to a contact address and protocol. At this moment the proxy can bind to the object and request the document's public key and its integrity certificate. The public key is compared with the OID and in a case of a mismatch the connection is dropped. Similarly, the certificate is examined. If the signature or the validity date are incorrect, the connection is again dropped.

When the OID and certificate are successfully verified, the proxy can start downloading the element. The element is first buffered, and it's hash value is compared with the corresponding entry in the certificate. If the verification succeeds, the element's content is sent to the browser.

Similar steps for the document's validation as performed by the proxy can be carried out by the object servers. The servers can protect this way themselves from accepting and storing incorrect documents.

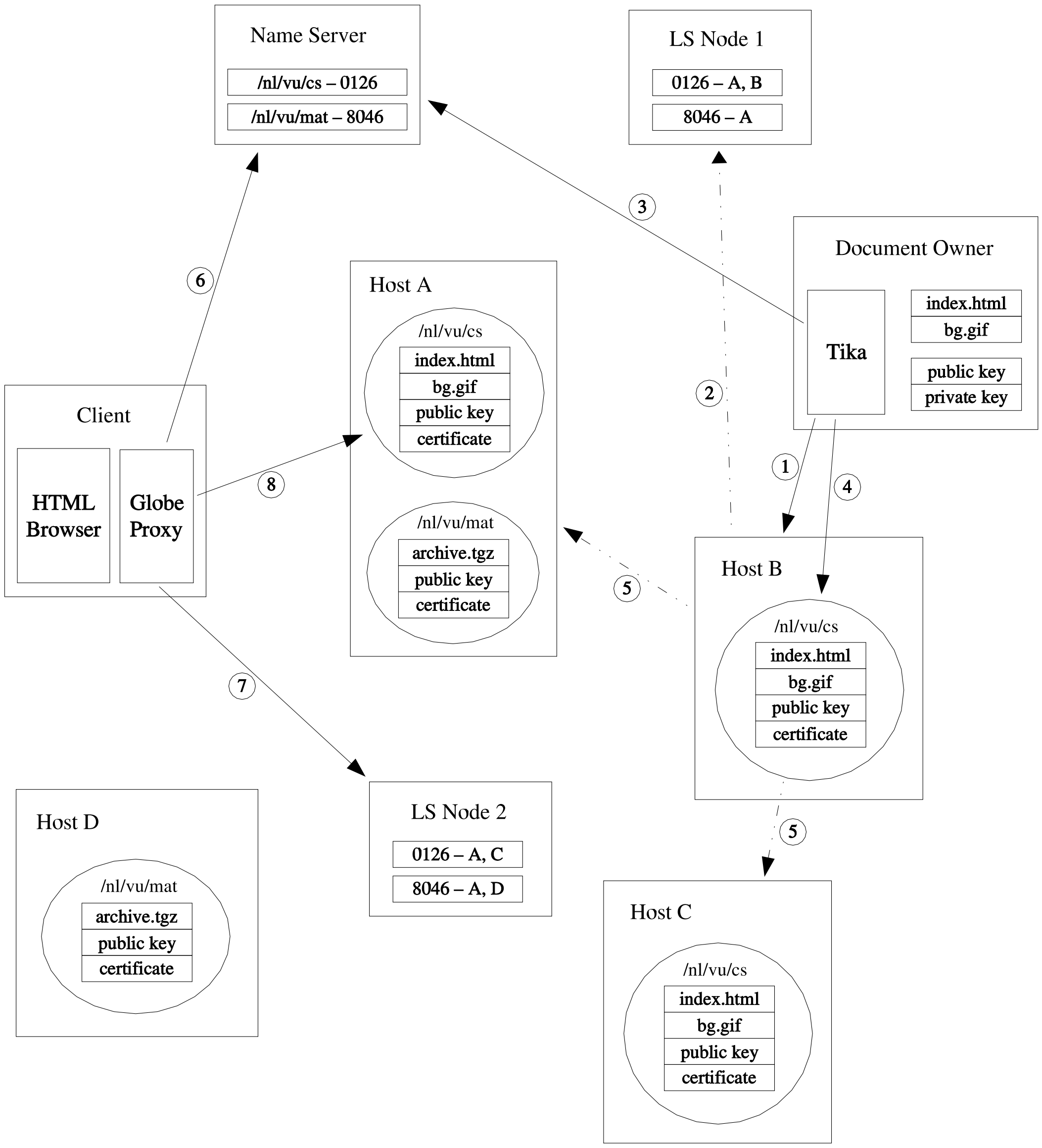

Figure 3.4: A sample GlobeDoc infrastructure.

The document owner starts creating a new document by creating an empty primary replica on Host B (1). The replica is automatically registered in the Location Service with an OID containing secure hash value of the public key supplied by Tika (2). Next, the OID (0126 in this example) is registered in the Naming Service under a name /nl/vu/cs *(3). The object owner sets document's content by transferring to Host B files* index.html, bg.gif, *document's public key and a certificate signed by the object's private key (4). Subsequently, Host B creates 2 replicas on Host A and C (5). One of the other users requests downloading document* /nl/vu/cs *in the browser. User's proxy contacts the Name Server and resolves the name to an OID 0126 (6). The OID is sent to the nearby Location Service Node, where two addresses are returned: Host A and C (7). The proxy chooses Host A, binds to the object and requests document's public key, certificate and* index.html. *After successful verification of the key, the certificate and the element's hash value,* index.html *is sent to the browser. If any of the verification procedures fail, the user can contact the replica on host C.*

Chapter 4

Prototype Implementation

4.1. Environment

The GlobeDoc prototype implementation is written entirely in **Java**. All the cryptographic operations are provided using the **JCE** library (Java Cryptography Extension).

4.1.1. Cryptographic Algorithms

We used the following cryptographic algorithms.

- Public/private key pair — **RSA**
- Self-certifying OID — **SHA-1**
- Certificate signature — **RSA** with **SHA-1**
- Hash value in certificate — **SHA-1**
- Keystore encryption — **JKS** (Java KeyStore)

4.1.2. Terminology

In this chapter we use the following terminology.

Keystore — A database of user's public and private keys. The keystore as a whole and individual keys can be protected with a password.

Keytool — The Java key and certificate management tool. It provides basic operations to manipulate on the keystore.

Truststore — A keystore containing only trusted public keys and certificates. It is used to make a decision if a public key or a certificate is valid and can be accepted by the user.

Certificate Authority — A trusted organization or entity in the system signing certificates. Certificates signed by the CA are accepted by all entities trusting the CA.

GlobeDoc Certificate — In the context of the GlobeDoc certificates do not conform to any standards (e.g. like X.509). A certificate is a structure used to verify document's integrity, specified in section 4.2.3.

4.2. The GlobeDoc Object

The GlobeDoc objects are constructed as the Globe distributed shared objects. The objects are persistent and are located on the Globe Object Servers. The implementation of the communication protocols and the replication mechanisms are provided by the Globe middleware.

The access to the object is available only through the object's interface. The interface is defined in the **IDL (Interface Definition Language)**. With the IDL interface the GlobeDoc is **interoperable**, the implementation of the objects and the client programs can be based on different languages supporting the IDL. It is possible for example, that the server is powered by Java and the client is written in C.

The GlobeDoc objects store the state of the documents. The state is read or written by the users — it is never modified by the the object itself, or by the server. The **write operations** are carried out only by the owner, **read operations** are allowed for every user.

The state of a document consists of the page elements and the object's meta-data. The elements contain blocks of raw data and are never interpreted by the object. The meta-data includes the element names, sizes, properties, the object's public key and its integrity certificate.

Whenever an update is performed, the object owner renews the certificate. Since only the owner knows the private key of the document (for security reasons), the object server is not able to create or modify the certificate, it must always be generated by the object owner. The object itself is merely a storage for the document with almost no processing on its own side. When serving a read request the object sends the document's content, public key and certificate intact. There is no encryption, all cryptographic operations are carried out by the client when verifying the certificate and the hash values.

An implication of such design is relatively low load on the server. It's possible then to support a large number of simultaneous clients without consuming much CPU power.

4.2.1. Consistency model

The GlobeDoc objects are accessed by the clients concurrently. While the replica synchronization is handled by the replication protocol, there is still a synchronization problem for a single object, namely the master. The read requests can always be served in parallel, but simultaneous updates can cause conflicts.

In some systems temporary inconsistencies are acceptable. For example most of typical web servers don't synchronize the updates. If a client happens to download a page when it's being updated a mixture of old and new version may be returned.

In a secure web model however, the coherence plays much more important role. Any inconsistence in the document's state may immediately cause a security alarm. Incorrect setting of the document's public key or certificate may cause the clients to reject the content they have received.

We divide conflicts caused by parallel updates into two categories: **write-write** conflict and **read-write** conflict.

Write-write conflict

The write-write conflict is a result of a concurrent update of the same part of the document, more precisely the same page element. Since only the owner is permitted to update the document, we do not expect the write requests to be very frequent. For this reason we use weak consistency model with pessimistic locking.

In order to modify an element the user must obtain a lock on the element first. Updates without locking are not allowed. The locks are exclusive, any attempt to lock an element that is already locked fails. The owner of a lock has full access to the element and can modify or delete it. After finishing the update, the owner should release the lock to allow other users to modify the element.

In case of a client's machine crash a lock can be removed by one of the other users. The server does not require that the lock must be released by the same user who created it. In the normal mode however, the users shouldn't release the locks that don't belong to them.

In the GlobeDoc we use the CVS terminology. An operation of locking elements of a document is called **check-out**. Opposite operation, unlocking, is called **check-in**. The elements that are locked are referred to as checked-out elements. Figure 4.1 shows an example.

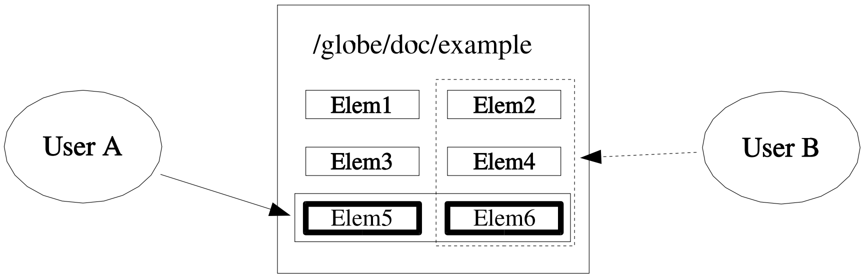

Figure 4.1: Element check-out.

User A checks-out elements 5 and 6 of a GlobeDoc object. Subsequently, user B tries to check-out elements 2, 4 and 6. The operation fails as the element 6 is already checked-out. User B must wait until A finishes the changes and checks-in the elements.

Read-write conflict

The read-write conflict occurs when a read request arrives while the object is being updated. Without synchronization this can result in unpredictable behavior, for example getting an element inconsistent with the integrity certificate.

To avoid such situations the update operations in the GlobeDoc are atomic, in a similar way as the transactions in the databases. To modify or delete an element the user has to check it out first. Newly added elements are already checked-out. The updates of the elements can span on multiple method invocations and can take arbitrary amount of time. The changes however, are not visible for other users until the final check-in. The users can read the object's content as before the update, but the new state becomes available after the check-in, similarly as after *commit* operation in transactions.

As an argument of the check-in operation the object owner passes a new certificate for the document. The certificate cannot be delivered to the object after the check-in, because the document may become inconsistent. The object has no means by which to create the certificate by itself, the responsibility of supplying a correct and coherent certificate lies on the owner. However, the object server can verify whether the received certificate is correct. If it is invalid or doesn't match the document's content, the server can reject the certificate and this way cancel the whole check-in procedure.

The process of document updating is illustrated in Figure 4.2.

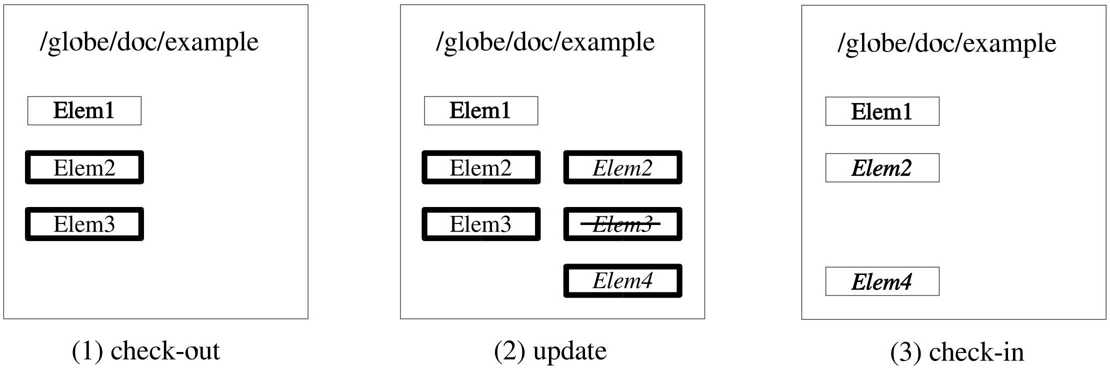

Figure 4.2: GlobeDoc object update.

The process of document updating. First, elements 2 and 3 are checked-out. Since now, other users have only read-access to the elements 2 and 3 (1). Element 2 is updated. Element 3 is removed. A new element 4 is added to the document. Other users can still read the old version of the element 2 and 3 but cannot access element 4 (2). After final check-in all the changes are visible for the clients. A new certificate is set for the object (3).

Read-write conflicts can also occur when an update is performed while the document is read by another user. Such situation is possible because read operation is not atomic, it consists of multiple invocations returning document's content in blocks. In such case the object throws an exception and the read operation is invalidated. The client restarts reading and downloads the new updated version of the document.

As we assume the read operations are much more frequent then the updates, only insignificant percentage of them can be interrupted by a simultaneous write.

4.2.2. Public Key

Every GlobeDoc stores its own public key. The corresponding private key is known only to the object's owner and is never exported from the owner's machine. In the current version of the prototype all the key pairs are implemented as standard RSA keys, but the general concept of the GlobeDoc does not enforce any specific public/private key algorithm.

The secure hash value of the public key is written to the OID (Figure 4.3. The algorithm used to compute the hash is SHA-1. Apart of the hash, the OID also contains a version number and a random string. The string is necessary to distinguish OIDs of different objects with the same public key. The object owner is not required to supply a different key for each document, so it's possible that two objects use the same keys.

The public key is never changed. It can be set only once when the document is created.

Version number	SHA-1 of the public key	Random number

Figure 4.3: The object identifier (OID).

4.2.3. The GlobeDoc Integrity Certificate

The GlobeDoc integrity certificates are used to verify the object's state. By comparing the certificate with the content of a page element the user can detect whether the document was modified after the certificate was issued. It is practically impossible to create two documents that have exactly the same certificates and different content, therefore only the owner can release a valid certificate. The certificate is renewed every time the document is updated.

GlobeDoc Certificate
Release Date
Validity Date
Root Element Name
Element 1 Entry
Element 2 Entry
Element 3 Entry
...
Signature

Figure 4.4: The GlobeDoc integrity certificate.

Element Entry
Element Name
Element Size
Hash Value Block Size
Block 1 Hash Value
Block 2 Hash Value
...

Figure 4.5: An element entry in the GlobeDoc integrity certificate.

The structure of the GlobeDoc certificate is presented in Figure 4.4 and 4.5. It consists of the following fields.

Release Date — The date when the certificate was released. It is specified in the UTC milliseconds. It allows to compare two certificates and choose the most recent one.

Validity Date — The expiry date for the certificate. The certificate is not valid after that particular date. In the current implementation of the GlobeDoc there is no additional certification revocation mechanism, the only way to make a certificate invalid is to wait until the specified date has passed. The validity date prevents using old certificates instead of the fresh ones.

Root Element Name — The name of the document's root element. The root element is sent to the client if none element name is specified in the request. Root element is analogous to the *index.html* file in traditional websites.

Element Entry — An entry created for each element belonging to the document. The element entry consists of a name, size, block size and a series of hash values.

Element Name — The name of one element. It certifies that the element is a part of the document.

Element Size — The size of the element. It is included to the certificate as an optimization, it allows the client program to download the element more effectively, for example by allocating optimal buffers. The element size is not required to verify element's integrity since the hash value is sufficient for this purpose.

Hash Value Block Size — The block size for calculating element's hash values. If the block size is zero, the element is not partitioned to blocks and the hash value is computed over the whole element's content.

Hash Value — The hash value of one block of the element's content. The number of the hash values for an element entry is equal to the number of blocks of the element, each value corresponds to one block.

Signature — The signature of the certificate. It guarantees the certificate was entirely and exclusively written by the object owner.

Certificate representation

The certificate is defined as an IDL object, so it is portable between different GlobeDoc implementations. The format of each field is precisely specified. All *String* values (e.g. element names) are encoded in *ISO-8859-1* format. Hash values are represented as hexadecimal numbers separated by colons. The signature is encoded in *Base64* format, the dates are converted to the UTC standard time in milliseconds. Such measures are necessary to correctly compute and verify the signature of the certificate, any change in the certificate's content may result in an incorrect signature evaluation and certificate rejection.

A drawback of the IDL implementation is more complex data representation and higher cost of certificate creation, marshalling and unmarshalling. As it is described in the measurements chapter, operations on the certificate are relatively expensive.

The certificate includes a validity date and a release date. Both these dates are absolute. We do not impose clock synchronization in the GlobeDoc, so the absolute time can be inaccurate. We presume however, that the clock skew between different machines can reach an order of seconds or minutes in the worst case. On the other hand, the certificates, are released with a validity interval for hours or days. With such assumptions, inaccuracies due to clock skew are negligible.

In the current GlobeDoc implementation there is no certificate revocation mechanism. The validity date is the only limit for the life-time of a certificate. The revocation scheme for the GlobeDoc is an object of current research. This topic is discussed in [16].

Element Entries

The number of element entries in a integrity certificate is unlimited. It is also possible to issue a certificate for a single element. More generally, a certificate can contain an arbitrary subset of elements belonging to a document.

In the current implementation integrity certificates are always created for all the object's elements. Assuming GlobeDoc objects consist on average of several dozens of elements the certificate size is about 3-5 KB. Since integrity certificates are quite small, the time spent on verification of a certificate mostly depends on the signature validation regardless of the number of element entries.