A Distributed Directory Service for Greenstone

George Buchanan, Annika Hinze
Department of Computer Science, University of Waikato
{g.buchanan, a.hinze}@cs.waikato.ac.nz

Abstract

Greenstone is a software for creating and maintaining distributed digital library collections. It provides a sophisticated federation mechanism for the collections. In order to support alerting notification about changes in the distributed collections, we propose a distributed directory service for the management of the distributed Greenstone installations. The Greenstone directory service (GDS) acts on top of the distributed Greenstone structure for the management of collections. This paper describes both, the initial distributed Greenstone structure and the distributed directory service.

1 Introduction

Greenstone is a software for creating and maintaining distributed, federated digital library collections. Notifying users about changes in collections is a much sought-after design feature, called alerting. This document gives the technical details about how the distributed structure of Greenstone servers can cooperate to allow for alerting over federated collections. We assume that the reader is familiar with basic techniques of alerting (publish/subscribe systems), for an introduction see, e.g., [3].

The implementation of the alerting-related communication for the distributed digital library software faces several challenges:

1. Network fragmentation and dynamic: The Greenstone network is highly fragmented; most servers are solitary installations with only a few references to other servers. References to other servers can be lost once a collection is restructured, i.e., the Greenstone network structure is not stable.

2. Unified single access point: Users interacting with Greenstone servers want to be notified about changes in (potentially distributed) collections residing at different hosts (i.e. federated collections) but they need a single unified interface for defining profiles. That is, users should not be forced to redefine their profile at several servers in order to avoid false negatives.

3. Dangling profiles: After a profile has been deleted, users no longer want to be notified about the respective events. Therefore, no profile information should be stored in a server that could be unreachable, leading to false positives.

We propose a design and an implementation of a Distributed Directory Service structure that supports alerting in Greenstone: the Greenstone Directory Service (GDS). Using the Greenstone network and the GDS, alerting over distributed servers and federated collections can be implemented. This paper serves three purposes: (1) it gives details about the structure of the distributed Greenstone network, (2) it introduces the Greenstone Directory Service, and (3) it describes the functionality of an alerting service for Greenstone using both the network and the directory service.

The remainder of the paper is structured as follows: Section 2 describes the conceptual and implementation details of how a distributed Greenstone installation manages federated collections of electronic documents. Section 3 introduces the design and implementation of the distributed Greenstone Directory Service. In this section, we also describe the concept of distributed alerting over federated Greenstone collections. Section 4 summarises the paper and indicates future work.

2 Distributed Greenstone: Managing federated collections in a Digital Library

In order to fully explain the mechanisms of distributed alerting in Greenstone, it is necessary to have detailed insight into the nature of Greenstone's management of federated collections.

2.1 Distributed Greenstone: Conceptual Level

This section explains the concept of a distributed Greenstone managing federated collections. The section follows a bottom up approach: first, simple collections on a single host are explained and then the more complex structure of distributed collections over several hosts is shown.

Simple Collections on a Single Host. A typical simple Greenstone digital library installation may be structured as shown in Figure 1. For a given computer, we refer to the local Greenstone software installation as *Greenstone server* (see the solid circle in Figure 1). The computer with the server installation in referred to as *Greenstone host*. Each host can manage several *collections* (hollow circles in Figure 1). A collection consists of a number documents (e.g., articles, music files); the document set is depicted as a square (shown here with example document files) . A user can gain access to the collections offered by a host via a receptionist (hached circles in Figure 1). A receptionist can give access to several Greenstone hosts. The receptionist, in cooperation with the hosts, presents the user with a simple access point to the collections offered by several hosts where the underlying storage and distribution structure is transparent to the user.

Hierarchic Collections on a Single Host. Collections can contain sub-collections. This concept is explained in Figure 2: We abstract here from the detailed view on the data sets (as presented in Figure 1) and subsequently refer to them simply by using

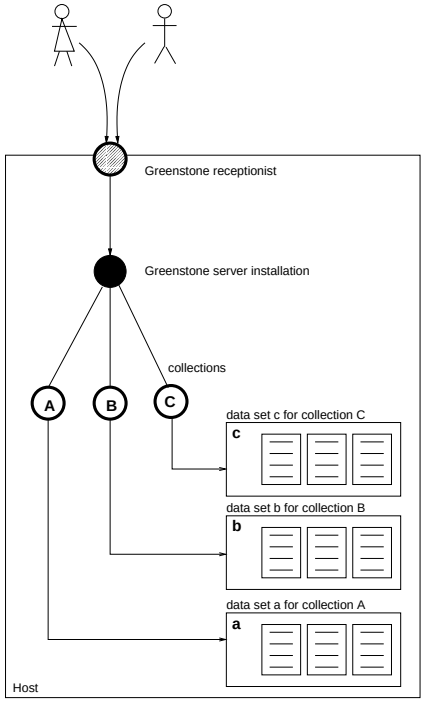
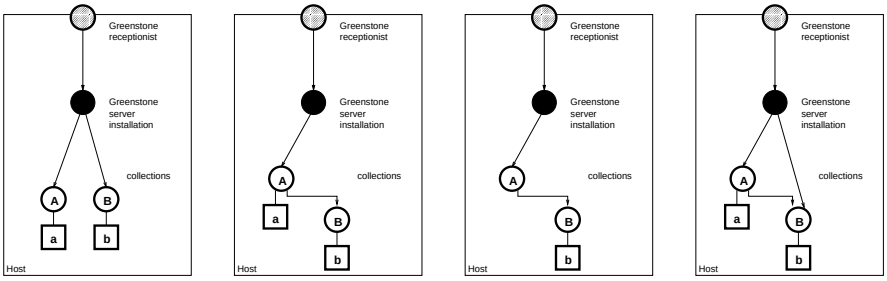


Figure 1: Conceptual structure of a simple Greenstone installation with single host

squares. Figure 2(a) shows two independent collections A and B on a single host. Configurations and data sets are distinct and independent. Figure 2(b) shows a collection A that combines two data sets into a single collection: data set a and data set b. collection B is not publicly available. Both data sets a and b are represented by collection A and the users are not aware of there being two distinct data sets. Figure 2(c) shows a virtual collection: collection A consists only of the sub-collection B without an own data set a; A is called a *virtual collection*. Again, users are not aware of collection B but regard data set b as the content of collection A. Figure 2(d) shows again a collection A with a sub-collection B (as in Figure 2(b)); this time collection B is also visible to the user as the independent collection B. The data set b serves as (sub)data set for A and as data set for B. Both A and B are publicly available.

Collections that are not publicly available such as collection B in the constellation shown in Figure 2(b) are referred to as *private collections*, whereas collections that can be directly accessed by the users are called *public collections*.

Distributed Collections over Multiple Hosts. The receptionist does not have to reside on the same computer as any of the servers it presents. Subsequently, we abstract from the receptionists' locations and specific users (see Figure 3). For simplicity, we



(a) Independent collections A and B

(b) collection A with sub-collection B

(c) Virtual collection A

(d) collection A with sub-collection B, and independent collection B

Figure 2: Conceptual structures of single host installations using various collection hierarchies

also abstract from the concrete document sets (shown as squares in Figure 2) that are represented by collections and sub-collections. Here in Figure 3, the hierarchical relationship between the collections is shown by arrows between the collections (depicted as hollow circles).

A Greenstone host can be presented by several receptionists to users, giving access to the same, overlapping, or distinct sets of collections. Conceptually, the host presents a certain view[1] on the set of offered collections: That is, a server can present a certain virtual set of collections through one receptionist and a different virtual set through another receptionist. See Figure 3 for an illustration of the concept of views as virtual collections. Collections can consist of several sub-collections as introduced in Figure 2. These can be distributed over several Greenstone hosts. A single (sub-)collection always resides entirely on one host. The distribution of sub-collections is transparent to the users: a sub-collection is presented as being part of a collection regardless of the actual location of the data. A collection cannot be accessed without its sub-collections; sub-collections are accessible without their super-collections (i.e., they can be public collections in their own right as shown in Figure 2(d)). For the distributed case, this is illustrated for (sub-)collections C, D and E in Figure 3: collection D can be accessed without collection C (in collection $Hamilton.D$) but never without its sub-collection E (collection $Hamilton.C$ and $Hamilton.D$). Note that the collection name always refers to the entry point collection (e.g., to collection D in $Hamilton.D$ consisting of D and E). Collections can also be created as virtual collections consisting of remote sub-collections only (similar to the local constellation shown in Figure 2(c)): The local collection is empty and refers to one or more remote sub-collections.

[1] We use the term 'view' here in the database sense of abstracting from the concrete storage presenting a certain access format to some clients and a different one to others. This can be done, e.g., for reasons of convenience or security.

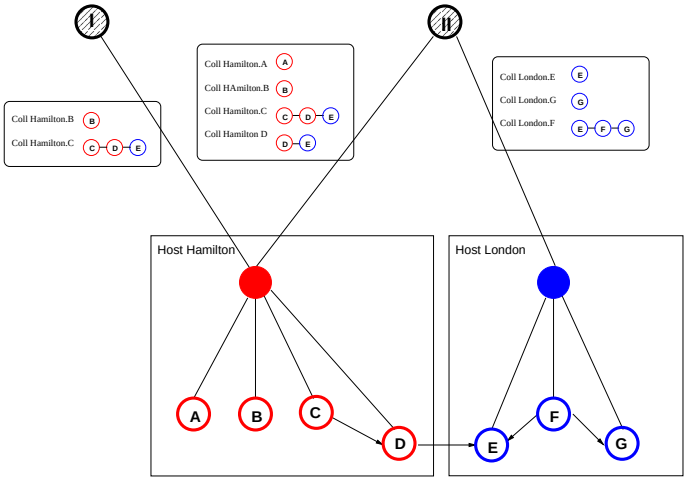


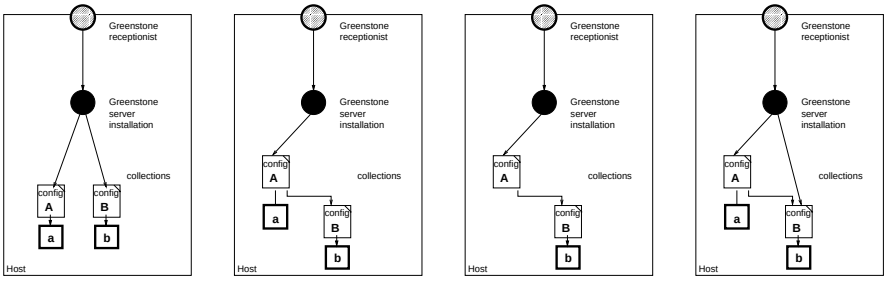
Figure 3: Conceptual structure of federated Greenstone collections residing on several Greenstone hosts

Federated Collections. The concept of federated collections is adopted from federated databases [4]: Federated databases and database systems are a classes of heterogeneous databases and database systems that are joined in order to meet certain organizational requirements and because they require their respective application specificities, integrity constraints, and security requirements to be upheld. Similar concepts apply to federated collections: they are heterogeneous collections that are joined under a single entry point to meet certain requirements. Greenstone is the digital library software that provides the management of the data storage, retrieval, and access. For more details on the concepts of Greenstone see, e.g., [1, 10, 11].

2.2 Distributed Greenstone: Implementation Level

This section describes the implementation level of Greenstone's collection management. Following the bottom-up approach from the previous subsection, we start with the implementation of collections on a single host and subsequently explain the more complex structure of federated collections residing on multiple hosts.

Hierarchic Collections on a Single Host. For each collection, the structure of the collection and its sub-collections is described in a *configuration file* (see Figure 4). A configuration file refers to a collection's data set. Independent collections have independent configuration files and data sets: In Figure 4(a), collection A is defined in $configuration\ file\ A$, which refers to data set a, and collection B is defined in $configuration\ file\ B$, which refers to data set b. In Figure 4(b), the collection B is defined in $configuration\ file\ B$. collection A consist of the data set a and the sub-collection B: $configuration\ file\ A$ refers to a and the configuration file of the



(a) Independent collections A and B

(b) Collection with sub-collection (A and B)

(c) Virtual collection A

(d) collection with sub-collection (A and B) and independent collection B

Figure 4: Implementation details of single host installations using various collection hierarchies

sub-collection B. Virtual collections have no primary data set – only one or more sub-collections, such as in Figure 4(c): collection A (defined in $configuration\ file\ A$) consists only of the sub-collection B (defined in $configuration\ file\ B$ and the corresponding data set b. Collections can be private or public. This is defined in the collection's configuration file: The configuration files for collections B in Figures 4(b) and 4(d) are different.

Distributed Collections over Multiple Hosts. The implementation of distributed collections over several hosts is very similar to the non-distributed case, using configuration files for the collections (see Figure 5). The collection $Hamilton.A$ is formed by the configuration file for A and the data set a (square a in Figure 5).

Here, we will explain in detail the implementation of the data access to distributed collections. Users access collection data via the receptionist's interface. The receptionist talks to the respective servers via the SOAP-based Greenstone protocol. The receptionist issues a request for collection data towards the Greenstone host where the collection resides that the user wants to access. Collections do not have direct access to other collections. Therefore, a collection itself consists only of its data set and configuration file. It possesses no software and cannot be executed. Without being run by a server, a collection is therefore unable to access its sub-collections. Only servers can interact with each other and access collections using the SOAP-based Greenstone protocol.

For example, to allow a user access to collection $Hamilton.A$, the receptionist issues a request towards the server on host $Hamilton$ (see left side of Figure 3). The server installation on $Hamilton$ then accesses the configuration file for collection A and follows the links provided there for the location of the data set a. The Greenstone server accesses the data set a and sends a response back to the receptionist, which in turn displays the data to the user.

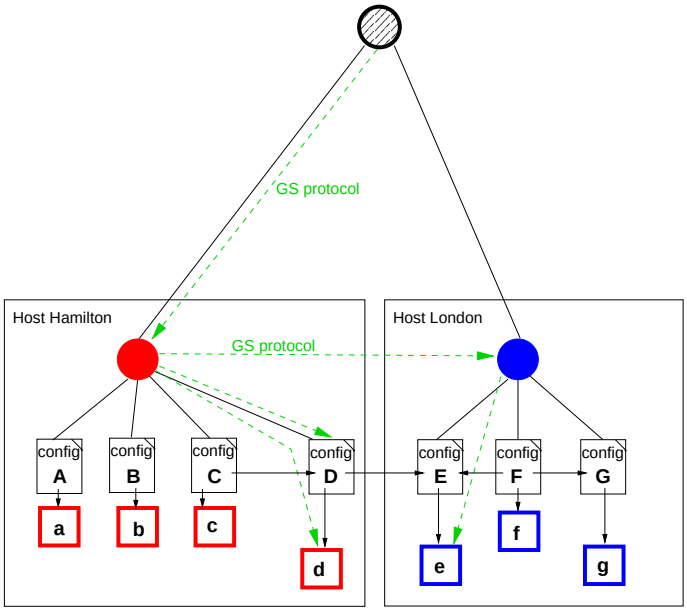


Figure 5: Implementation details of federated Greenstone collections residing on several Greenstone hosts

To allow a user access to collection $Hamilton.C$, the receptionist issues a request towards the server on host $Hamilton$ (see middle of Figure 5). The server installation on $Hamilton$ then accesses the configuration file for collection C and follows the links provided there for the location of the data set c and the link to the sub-collection D. The data of Sub-collection D is accessed via reading the configuration file for D first and then accessing the data set d. The Greenstone server accesses the data sets c and d and sends back a response containing data from both data sets to the receptionist, which then in turn displays the data to the user. The user is not aware of the sub-collection structure within C.

To allow a user access to collection $Hamilton.D$, the receptionist issues a request towards the server on host $Hamilton$ (see centre of Figure 5). The server installation on $Hamilton$ then accesses the configuration file for collection D and follows the link provided there for the location of the data set d. The $Hamilton$ server accesses the data set d. The server also learns about the existence of sub-collection E on host $London$. The Greenstone server $Hamilton$ sends a request to $London$ asking for the data in collection E. For the communication between servers, the Greenstone protocol is used. $London$ accesses $London.E$'s configuration file and subsequently the data set e. $London$'s response is to send the data of e back to $Hamilton$. The $Hamilton$ server has now access to the data sets d and e; it sends a response containing d and e back to the receptionist, which in turn displays the data to the user. The user is unaware of the distributed sub-collection structure within D.

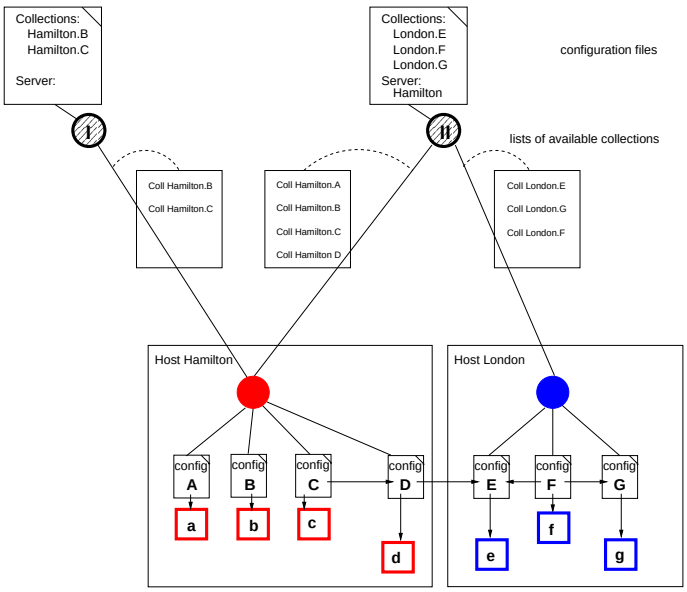


Figure 6: Implementation details of federated Greenstone collections residing on several Greenstone hosts accessed via several receptionists

Multiple Receptionists and Collection Registration. We distinguish between private and public collections. Typically, a private collection cannot be accessed as an independent collection but only as sub-collection of another collection.[2] Thus, private collections are not directly registered with receptionists. A collection's configuration file defines whether the collection is private or public. Public collections can be registered with more than one receptionists.

A receptionist can learn about collections in two ways: registration by server or registration by collection.

Registration by server: Typically, a receptionist learns about the existence of Greenstone servers/hosts, rather than about single collections. A Greenstone administrator registers a host with a certain receptionist. All public collections on this host are automatically known to, and registered with, the receptionist. A receptionist re-scans the list of public collections for all its registered hosts when it is restarted. When a new collection on a registered Greenstone host is created, the receptionists learns about it the next time it is restarted.

Registration by collection: A receptionist can learn about the existence of single collections. This method only works if the host that the collection resides on is not

[2]One exception to this occurs for collections that are work in progress: While a librarian builds a collection or for test purposes, a collection might not yet be registered with a receptionist. Thus, the collection is currently private. At a later point in time it might become a public collection.

registered with the receptionist. The collection administrator registers the new collection with the receptionist by entering it into the receptionist's collection list. The receptionist does not automatically learn about new collections on the same host. On restart, the receptionist scans all registered collections. Currently, registration by collection is being implemented for Greenstone 3 (it was already in place for Greenstone 2)

The same collection can be registered at several receptionists. Similarly, the same sub-collection can appear in several collections and as independent public collection at different receptionists or the same receptionist. Each receptionist can handle several hosts and additionally a list of registered collections from different hosts. If on startup a collection is not available (e.g., due to deletion or unavailability of the host or other technical issues), the user will not see this collection in their interface. For each receptionist, both the list of collections and the list of hosts are held together in the configuration file. In the example in Figure 6, the receptionist on the left hand side, receptionist I, has no registered hosts but a list of registered collections ($Hamilton.B$ and $Hamilton.C$) as shown in the configuration file for the receptionist. For the receptionist II on the right hand side, the Greenstone server $Hamilton$ and has been registered and the single collections $London.E$, $London.F$, and $London.G$ have been registered. At the startup of a receptionist, the list of available collections is compiled for each Greenstone server registered with it. (shown as the three lists in the centre of Figure 6).

3 Distributed Alerting: Greenstone Directory Service

This section introduces the design and implementation of the distributed Greenstone Directory Service in Section 3.1. We also describe the concept of distributed alerting over federated Greenstone collections in Sections 3.2 and 3.3.

3.1 Greenstone Directory Service

The Greenstone Directory Service is a distributed directory service. The concept of a directory service for group communication in distributed applications and virtual file systems is well established [5, 8]. Recently, it has also been introduced on the application level for asynchronous communication between changing participants in a mobile environment, e.g., for mobile agents [7][3].

3.1.1 Directory Service

A distributed directory service organizes access to a set of content servers by providing a logical network of directory servers. It acts as an authority that can securely authenticate resources and manage identities and relationships between them.

[3]Moreau follows a formal approach in their design

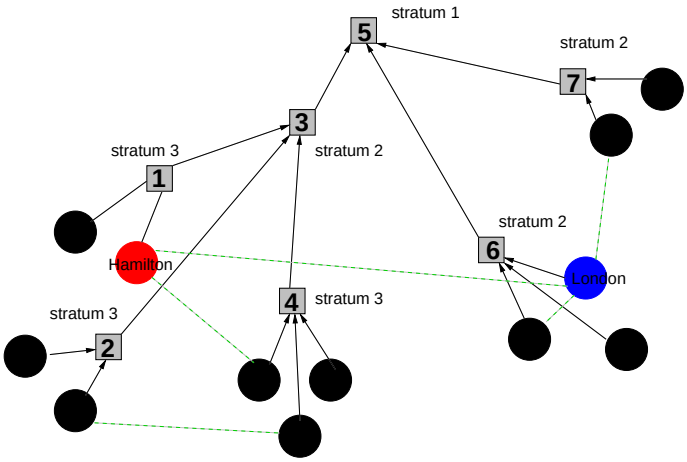


Figure 7: Concept of the Greenstone Directory Service: The arrows indicate GDS connections both between GDS servers and GS servers, which are the client of the GDS, and internally between GDS servers. The dashed line indicates the original fragmented GS network.

A directory service maps the names of network resources to their respective network addresses. Each resource on the network is considered as an object on the directory server. Information about a particular resource is stored as attributes of that object. Information within objects can be made secure so that only users with the available permissions are able to access it.

A directory service defines the namespace for the network. A namespace is a set of rules that determine how network resources are named and identified. The rules specify that the names be unique and unambiguous. In LDAP, such a name, called as distinguished name (DN) is used to refer to a collection of attributes which make up a directory entry.

3.1.2 Greenstone Directory Service: Concept

The conceptual organization of the Greenstone Directory Service is depicted in Figure 7. Currently, on each Greenstone host runs only one Greenstone server. The figure shows a realistic scenario of Greenstone servers running on disconnected hosts. Most Greenstone servers are stand-alone installations. If a server holds distributed sub-collections, it holds a direct reference to one or more other Greenstone servers (see description of communication using the Greenstone protocol in Section 2.2). These references and connections are depicted in Figure 7 with dashed lines. We see, for example, the connection between the $Hamilton$ server and the $London$ server that has been described earlier.

Each Greenstone server is registered at exactly one service node on the distributed

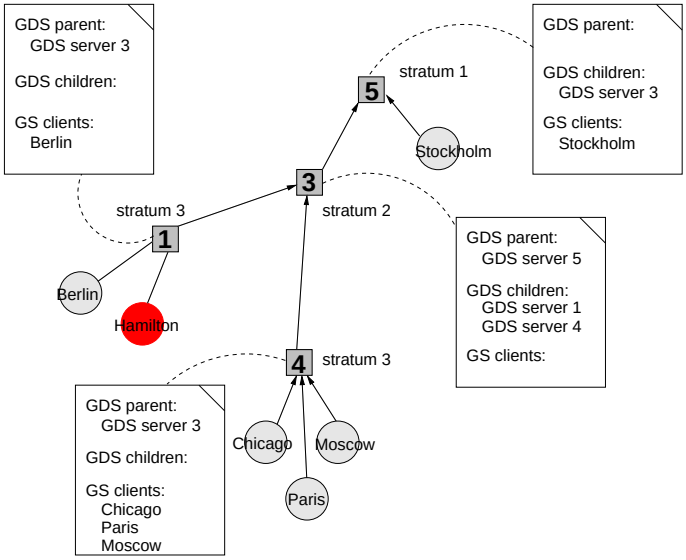


Figure 8: Implementation of a GDS server network (fragment): servers maintain link list for parents, children, and clients

Greenstone Directory Service (GDS). Each service installation is depicted as shaded square with an identifying number. Each GDS installation resides on a certain stratum. The concept of strata is adopted from the Network Time Protocol (NTP) [6]. A GDS primary server, also called of $stratum1$, is a computer equipped with a GDS software. A GDS Primary Server has access to all Greenstone servers within the network, following the tree towards the leaves. Other computers, at $stratum2$, equipped with GDS software, have similar access to all Greenstone servers in their sub-trees. They have to query the primary server to obtain access to other branches of the tree. The GDS serves as an analog to the Domain Name Service, where Greenstone servers can be accessed by their network-internal name without the requesting service having to know the actual address or location of the service.

In order to distribute messages to all Greenstone servers, a Greenstone server forwards the message to its GDS server. The message is distributed upwards within the tree and downwards to all tree leaves.

3.1.3 Greenstone Directory Service: Implementation

The GDS service is implemented as a tree structure of auxiliary GDS servers (shown as squares in Figure 7). A GDS server consists of a Java application that runs on a host computer, and at present any computer can host only one GDS server. Host computers for GDS servers may or may not be also host computers for Greenstone servers.

Greenstone servers that provide standard Greenstone features connect onto the

GDS service by registering at a single GDS server. The Greenstone servers register with GDS servers in order to be identified in the Greenstone network. By use of the GDS, the Greenstone servers may forward messages to other Greenstone servers on the same GDS network. A list of initial GDS servers is available to the GS servers[4]. The initially contacted server could recommend a list of GDS servers nearby the GS server. This will be part of future work. Each GDS server holds a *link list* of both local GDS servers (child servers in the tree) as well as local GS servers. An example is shown in Figure 8. In the figure, we show the fragment of a network: Each GDS server's list contains references to its parent within the GDS network, its children and its GS clients (GS servers). As this example illustrates, the servers on each stratum can maintain links to GS clients, not only the tree leaves (see GS server $Stockholm$ being linked as client to $GDSserver5$). Thus, the GDS tree is a doubly-linked tree. GDS servers could also cache information, e.g., about GS clients registered at all servers lower down in the hierarchy or about all GS client servers in the network. However, at present a specific a caching strategy is not implemented but planned for future extensions.

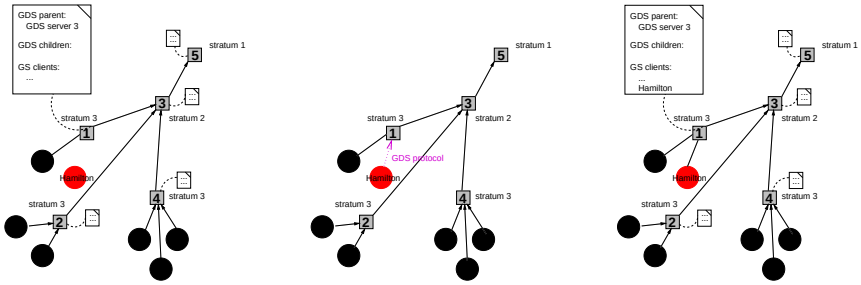
Each GDS server can perform the following actions:

1. Registering of other servers (on the next lower stratum)
2. Registering at another server (on the next higher stratum)
3. Register any number of GS servers
4. Unregister any of the above.
5. Messaging
6. Identify connected GS servers to the network

The registration of or at other servers (Actions 1 + 2) is in principle similar to the registration of Greenstone servers with the GDS (using the same protocol with a differing message content). Here, we focus on the description of the registration of Greenstone servers at the network (Action 3) and the un-registration of Greenstone clients from the GDS (Action 4). These actions as well as the messaging and the client identification are described in detail in the next paragraphs.

Register a GS server with the GDS (Action 3) Figure 9 demonstrates the registration of a Greenstone server at the GDS network. For clarity, we show only the link list of $GDSserver1$; the other lists are iconized. In this example, the new Greenstone server is $Hamilton$ (see Figure 9(a)). Server $Hamilton$ sends its registration message to $GDSserver1$ (see Figure 9(b)). The server stores the identity of the $Hamilton$ server in its local list of Greenstone servers, linking $Hamilton$ into the GDS network (see Figure 9(c)). Note that a Greenstone server should only register with one GDS server. GDS servers may cache some or all responses forwarded through them, and some or all registrations held by other GDS servers on lower strata beneath them.

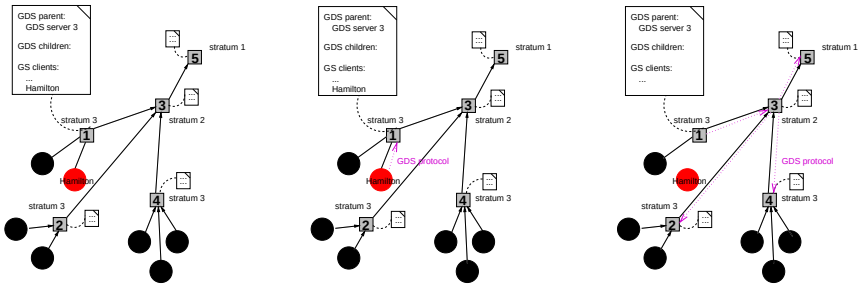
[4]Currently, we support an open GSD network for all servers. Several strategies are considered for future work, e.g., initial connection only to the receptionist and all siblings under this receptionist, and explicit exclusion from or inclusion into the GSD



(a) Step 1 (b) Step 2 (c) Step 3

Figure 9: Registering (adding) a GS server at the GDS

Unregister a GS server with the GDS (Action 4) Similarly, Figure 10 shows the same $Hamilton$ server un-registering itself from the GDS network (see Figure 10(a)). A message is sent to the GDS server (see Figure 10(b)). In this case, the GDS server not only removes the registration from $Hamilton$, but also passes a message through the rest of the network cancelling this registration. This strategy avoids any problems should the registration of $Hamilton$ have been cached anywhere else (see Figure 10(c)).



(a) Step 1 (b) Step 2 (c) Step 3

Figure 10: Un-registering (deleting) a GS server from the GDS

Messaging (Action 5) Every message is sent in an XML format across TCP/IP using SOAP. A Greenstone server that wishes to send a message across the GDS network initially sends the message to the GDS server at which it is registered. The GDS server then takes control of how the message is sent across the GDS network. With only one exception (which is described subsequently), messages sent into the GDS network do not receive a response or acknowledgement. Messages within the GDS system may be sent in one of three ways: Broadcast, point-to-point, multicast. At present, point-to-

point and multicast messaging are not implemented is not implemented since they are not required in the current design of the alerting service.

For broadcast messages, each message will be forwarded to all Greenstone servers on the GDS network, and from these to their registered Greenstone servers. The principle is illustrated in Figure 11. The GDS server that first receives the message then broadcasts it to all other Greenstone servers registered with it, GDS servers registered with it, and the GDS server on the next higher stratum. Further GDS servers repeat this pattern, forming a complete span of the GDS tree. If a given Greenstone server is not presently available or active, any messages forwarded to it are simply dropped by its GDS server – message sending is only 'best-effort'.

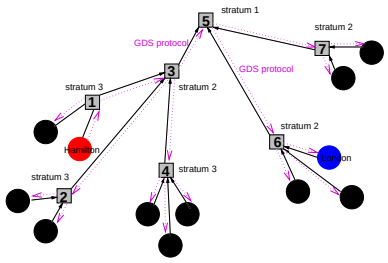


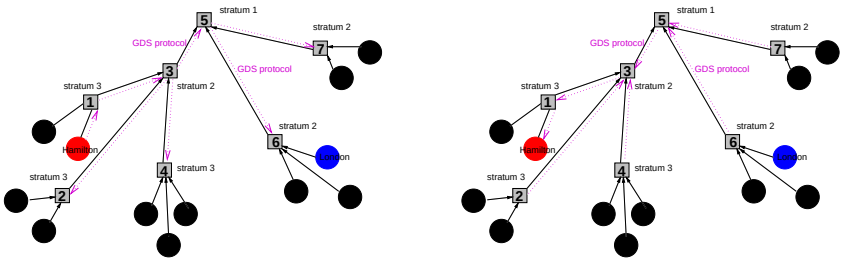
Figure 11: Broadcasting a message

Call for participants in the GDS (Action 6) The one exception to the convention of non-acknowledged messages is when a call is sent to the GDS network to return the names of all Greenstone servers in the network. In this case, the originating server is sending a query to the network, and expects a response. The call is distributed within the GDS network; Figure 12(a) shows the communication for the case that no caching is implemented.

A second exceptional property of this message is that no other Greenstone servers receive a message (see Figure 12(b)), i.e., call and response are answered by the GDS network without interacting with the Greenstone servers. As response, each GDS server lists the servers registered at it. The response to this message will be received in separate parts by the originating Greenstone server – i.e. the server requesting a list of all servers.

3.2 Alerting for Solitary Collections

Users submit their profiles to a certain Greenstone server. The profiles reside at the server that the user submitted the profile to. Event messages are created by servers; they have to me filtered according to the user profiles. After filtering the local profiles, the events are flooded to all servers using the GDS network. For event flooding of the network, we use the broadcast facility of the GDS. This technique is inspired by the *event flooding* as proposed by Carzaniga [2]. Its implementation significantly differs



(a) Step 1 (b) Step 2

Figure 12: Call for participants in the GDS

from their design as here we use the Greenstone Directory Service as communication network and not the network formed by Greenstone servers. The reasons for this design have been discussed in Section 1 (potentially disconnected network fragments and dangling profiles). Each server filters the incoming events according to the local profiles and notifies its clients accordingly.

Figure 13 shows a communication example. Greenstone servers have clients connected that define profiles for the alerting service. The clients and their profiles are depicted as small circles connected to the servers. Let's assume a new collection is formed at the $Hamilton$ server. Subsequently, an event message is created by $Hamilton$ server announcing the documents in the new collection. The message is forwarded upwards in the GDS tree and downwards towards the leaves using the GDS protocol (indicated in dotted arrows). We consider the client connected to the $London$ server: Their profile is stored at the $London$ server. As soon as the event message arrives at the server, it is filtered and a notification is send to the client.

In the case of solitary collections, i.e., without distributed sub-collections, the server where the collection resides on issues the event message. The task is more challenging once the collection also supports sub-collection on other servers.

3.3 Alerting for Distributed Sub-Collections

If a sub-collection that resides on a different server than the collection is rebuild, the server where the collection resides on is not aware of the rebuilt sub-collection. For illustration, see in Figure 14 the sub-collection of $Hamilton.D$ with $London.E$. Therefore, it cannot issue the event message. The server where the sub-collection resides on is not ware of the (sub-) collection being part of another collection. It can therefore only issue event messages regarding the (sub-) collection as an independent public collection or part of a local collection – this case has been addressed in the previous paragraph. Using this event message as a notification for the other server is not possible, because the server might not be aware of the collection this (sub-)collection belongs to (in case of different entry collections). The problem is even greater is the sub-collection is a private sub-collection to a virtual collection. No event message would be issued in this

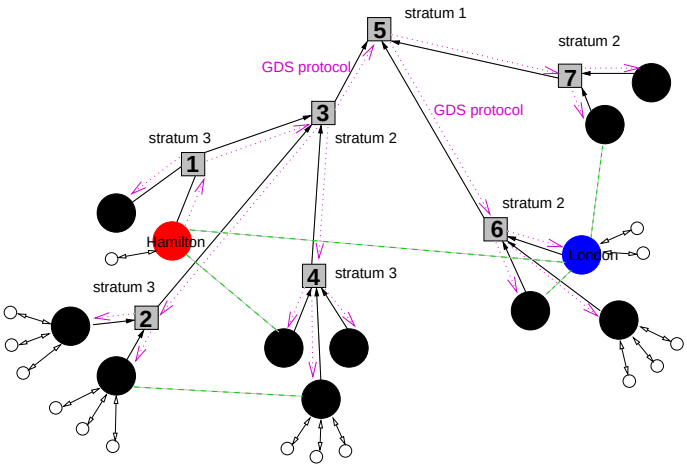


Figure 13: Concept of Alerting about collections on single hosts using the Greenstone Directory Service

case.

We propose therefore the following hybrid design: In addition to client profiles, the alerting service supports internal server profiles that observe changes in sub-collections. The profiles are handled differently to the user profiles: they use the *profile forwarding* approach. Server profiles are forwarded directly to the servers providing the sub-collection. This communication uses the Greenstone server network and the GS protocol. Event messages are filtered close to the event publishers and the servers acting as profile clients are notified directly without using the GDS. Considering our example in Figure 14, we assume again that collection D on server $Hamilton$ has a sub-collection E on server $London$ (indicated by the conceptual link). $Hamilton$ registers a profile at $London$, using the GS protocol communication network. On event in $London.E$, $London$ filters the event against the profiles and notifies $Hamilton$ about the event. After $Hamilton$ learns about the event, it can announce the event in $London.E$ as event from collection $Hamilton.D$ using the GDS network as described before.

Thus, in a second step the events in sub-collections are announced as events within the entry collection using the mechanism already described in the previous paragraph.

4 Summary & Future Work

This paper describes the technical details of distributed alerting in Greenstone: (1) the methods to access federated and distributed collection in Greenstone (using the Greenstone network), (2) the design and implementation of the Greenstone Directory Service to connect the dynamic and fragmented network parts, and (3) distributed alerting using

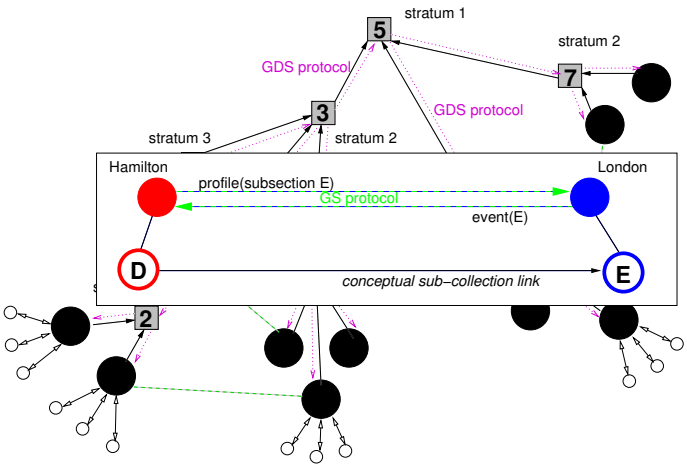


Figure 14: Concept of Alerting about distributed collections using the Greenstone Directory Service

both the Greenstone network and the Greenstone Directory Service.

We follow a hybrid approach to implementing the alerting service over open and distributed Greenstone networks by:

1. combining communications in two different networks, i.e., using the Greenstone network and the Greenstone Directory Service
2. using two communication protocols, the SOAP-based Greenstone protocol and the XML-based GDS portocol
3. using a combination of two versions for identifying communication partners: point to point communication with directly connected GS server (hosting distributed collections) and anonymous broadcast with the rest of the network connected by the GDS (hosting federated collections).
4. using a combination of two different (opposite) filtering strategies (event flooding in the GDS and profile forwarding in the GS network) for federated and distributed collections

As future work, we plan to thoroughly evaluate the scalability of the alerting using both the GDS and the GS network; so far, initial tests have been promising. Different caching strategies for link information in the GDS will be evaluated. We also plan to integrate a smooth transformation of Greenstone search queries into profiles and vice versa in order to integrate the user experience of alerting service even more with typical Greenstone interactions. MOre details about the profile language and the implementation of the local alerting functionality can be found in [9].

References

[1] David Bainbridge and Ian H. Witten. Greenstone digital library software: current research. In *JCDL '04: Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries*, pages 416–416. ACM Press, 2004.

[2] A. Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, Milano, Italy, December 1998.

[3] A. Hinze. *A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service*. PhD thesis, Freie Universität Berlin, July 2003.

[4] David K. Hsiao. Federated databases and systems: part i — a tutorial on their data sharing. *The VLDB Journal*, 1(1):127–180, 1992.

[5] M. Frans Kaashoek, Andrew S. Tanenbaum, and Kees Verstoep. Using group communication to implement a fault-tolerant directory service. In *International Conference on Distributed Computing Systems*, pages 130–139, 1993.

[6] David L. Mills. Internet time synchronization: The network time protocol. In *Zhonghua Yang and T. Anthony Marsland (Eds.), Global States and Time in Distributed Systems, IEEE Computer Society Press*. 1994.

[7] Luc Moreau. Distributed directory service and message routing for mobile agents. *Sci. Comput. Program.*, 39(2-3):249–272, 2001.

[8] B. Clifford Neuman. The prospero file system: A global file system based on the virtual system model. *Computing Systems*, 5(4):407–432, 1992.

[9] A. Schweer. Alerting in greenstone 3. Master's thesis, University of Dortmund, Germany, 2005.

[10] Ian H. Witten and David Bainbridge. *How to Build a Digital Library*. Elsevier Science Inc., 2002.

[11] Ian H. Witten, Stefan J. Boddie, David Bainbridge, and Rodger J. McNab. Greenstone: a comprehensive open-source digital library software system. In *DL '00: Proceedings of the fifth ACM conference on Digital libraries*, pages 113–121. ACM Press, 2000.