# A Distributed Alerting Service for Open Digital Library Software

Annika Hinze
Department of Computer Science
University of Waikato
New Zealand
a.hinze@cs.waikato.ac.nz

George Buchanan
UCL Interaction Centre
University College London
United Kingdom
g.buchanan@cs.ucl.ac.uk

## Abstract

*Alerting for Digital Libraries (DL) is an important and useful feature for the library users. To date, two independent services and a few publisher-hosted proprietary services have been developed. Here, we address the problem of integrating alerting as functionality into open source software for distributed digital libraries. DL software is one application out of many that constitute so-called meta-software: software where its installation determines the properties of the actual running system (here: the Digital Library system). For this type of application, existing alerting solutions are insufficient; new ways have to be found for supporting a fragmented network of distributed digital library servers. We propose the design and usage of a distributed Directory Service. This paper also introduces our hybrid approach using two networks and a combination of different distributed routing strategies for event filtering.*

## 1. Introduction

Users of modern digital libraries (DL), such as Greenstone [21], can keep themselves up-to-date by searching and browsing their favourite collections, or more conveniently by resorting to an alerting service. It has been shown in the context of the projects Hermes [10] and Dias [13] that an alerting service which integrates information from a wide variety of providers relieves users from the tedious and cumbersome task of searching and browsing. For open DL software, it is a promising approach to directly integrate the alerting functionality into the Digital Library software. The most successful open source software for creating digital library systems is Greenstone.[1] Greenstone is a meta-software to build digital libraries (and their collections).

Here, we address the problem of integrating alerting as functionality into open source software for distributed digital libraries. The implementation of an alerting service that supports notifications about Greenstone collections and documents proved a challenging task.

The Greenstone network of DL servers is only loosely coupled and subject to changes. The network may contain cycles and/or consist of islands of arbitrarily connected/disconnected servers and collections. Users want homogenous access to their profiles and collections at varying network nodes. This situation created new challenges and opportunities that are neither addressed by existing digital libraries (and their alerting services), independent DL alerting services such as Hermes and Dias, nor existing event notification systems (as we will show in Section 2). In particular, we addressed the following research problems:

1. Fragmented and dynamic network: 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. Cyclic network: The service faces the problem of possible infinite loops and duplicates of event messages.
3. Unified single access point: Users interacting with Greenstone servers want to be notified about changes in collections residing at different hosts 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.
4. 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.
5. Alerting as a fluent extension of searching and browsing: The collections built using the Greenstone software may support a variety of retrieval functionalities.

[1] Evidence can be found in the support of Greenstone by UNESCO and the international DL community [8].



Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'05)
1545-0678/05 $20.00 © 2005 IEEE

In order to allow for homogenous access, the alerting service should not create new retrieval paradigms but extend the existing ones of a collection.

6. Heterogenous content: Digital library collections contain differing metadata formats, variable content types (e.g. audio, visual, text) and organization (classification schemas). Thus, each individual installation of a meta-software such as Greenstone is likely to have little in common with any other given installation. This makes the prediction of the required filter algorithms a difficult task.

This paper describes the design and implementation of a distributed alerting service for digital libraries, supporting distributed and federated collections managed by a distributed DL software, namely the Greenstone software. We introduce our design of a network structure and communication protocol for distributed alerting as well as a methodology for profiles as continuous queries using the built-in collection-based retrieval functionalities. The remainder of the paper has the following structure: Section 2 shows that it is not possible to simply copy related approaches for the scenario encountered here. Section 3 describes implementation details of the distributed Greenstone software that are important for the design of the alerting service. Section 4 introduces our design for the distributed alerting service for accessing federated and distributed collections. Section 5 describes the profile language and local filter methods. Section 6 briefly gives implementation details for the Greenstone Alerting Service. To conclude, we summarize the lessons learned in the design and implementation as well as the contributions of this paper.

## 2. Related Work

This section presents the results of our analysis of related approaches pertinent in the context of the proposed application for digital libraries.

### 2.1. Alerting and Digital Libraries

*Individual alerting services* are offered by publishing houses (such as Springer Link Alert[2], Elsevier Contents Direct[3], or ACM Table-of-Contents Alerts[4]) and secondary publishers (e.g., SwetsScan[5]) and citation services (e.g, ISI services[6]). These are solitary (centralised) services that neither cooperate with other services nor do they openly support independent digital libraries.

[2]available via http://springerlink.metapress.com/
[3]http://www.contentsdirect.elsevier.com/
[4]available via http://portal.acm.org
[5]available via http://www.swetsblackwell.com/
[6]available via http://isiknowledge.com/wos

A few *document-centered systems* have been proposed which could be used in this context: One of the earliest systems developed was SIFT [22], a centralised tool for wide-area information dissemination, that is now commercially operated as InReference. Support for open heterogeneous document collections is found in Hermes [10], CQ [15] and Dias [13]. Hermes is a centralised system that cannot support distributed user access or collections. CQ supports keyword based queries and focusses on query routing in an acyclic distributed environment. The keyword approach is too limiting in our context; the technical limitations of the routing in tree-based networks are discussed in Section 2.2. Dias is a distributed system that extends the architecture of Siena [5] into a P2P system. The data model of Dias is based on free text and the profile language supports a boolean model with proximity operators. Thus, Dias supports a predefined document structure focussing on textual documents. We see this approach as too limiting for an open digital library supporting collections of arbitrary document types (e.g., music, pictures, text documents). We discuss the technical aspects of Dias' implementation in Section 2.2.

### 2.2. Distributed ENS

Current distributed Event Notification Services are mostly designed on a *fixed acyclic overlay network* topology [2, 5]. Profile flooding or event flooding are used to distribute information in the network (e.g., in Rudbes [23] and JEDI [7]). Computation of coverings and mergings between profiles are used to reduce the traffic load and the memory usage (e.g., in Rebecca [16] and A-mediAS [12]). In general, network reconfigurations are not allowed and link failures or congestions in the inner nodes are neglected. For our scenarios, a fixed acyclic overlay network topology cannot be used since the Greenstone (GS) network is neither fixed nor acyclic. Profile flooding could lead to orphan profiles residing on disconnected GS servers sending spurious notifications regarding long-cancelled profiles. Scribe [18] and Hermes [17] use rendezvous nodes, which act as meeting points for subscriptions and events. A rendezvous node may become a bottleneck in the network [2]. Similarly, node or link failures may lead to erroneous system behaviour creating both false positives and negatives. Spurious notifications may be created by disconnected servers for cancelled profiles; occurring events might not be filtered due to the unreachability of their designated rendezvous nodes.

Few approaches work towards more *general network topologies* of arbitrary graphs [6, 11]. Carzaniga [6] proposes a combined broadcast and content-based (CBCB) routing scheme: a content-based layer superimposed over a traditional broadcast layer. Each node maintains its own broadcast tree which is required to be symmetric in the sense that each pair of nodes does have an intersection in



Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'05)
1545-0678/05 $20.00 © 2005 IEEE

their corresponding trees. This approach allows more general networks but restricts itself to only use acyclic structures, resulting in the problems discussed above. A two-level approach [4] partitions the problem space (i.e., which servers are responsible for filtering which events) into zones and creates a routing tree for each zone (global level). In addition, servers are organized into server cliques based on their network proximity; each server is responsible for certain zones (local level). Again, this acyclic structure is susceptible to failures and congestion. Moreover, the topology is created statically by a central instance and does not adapt to network changes or link failures. Halletal [11] eliminates cycles using sequence numbers; the routing adapts distance vectors known from traditional routing protocols. Each node stores and, possibly, filters all profiles. This only scales to a small number of profiles and leads to the mentioned problems of orphan profiles.

Recent work uses flexible *peer-to-peer network* structures. Koubarakis [14] see the nodes of the distributed ENS as super-peers and the subscribed clients as peers. A spanning tree is constructed for the current physical network structure of super-peers. The network is semi-dynamic: If the physical network structure changes, the spanning tree is simply recalculated. This approach uses P2P protocols, but the underlying network structure is similar to a tree of nodes; peers are assumed not to be orphaned. Many other approaches use Distributed Hash Tables (DHTs) implemented over structured P2P networks. In [20], several dissemination trees are constructed (one per peer), forming an overall graph. Profiles are stored locally and the whole Peer-to-Peer network is flooded with the events. For performance reasons, notification delivery is best-effort: when peers fail, there is no guarantee that notifications are delivered. A P2P network is not sufficient for our scenario, since peers are required to maintain the network. DHT topologies require the cooperation of the involved peers, which cannot be postulated in the open DL context. Merging of independent networks is not arranged for and not possible without considerable reconstruction.

So far, distributed architectures are either based on fixed acyclic networks which use flooding or rendezvous nodes; or they depend on groups of dissemination trees, which are difficult to adapt to the expected network changes. For our given application field of DLs, we conclude the following design considerations: Fixed architectures of nodes lack the necessary flexibility; profile flooding and rendezvous nodes are not sufficient for filtering because both false positives and negatives can occur.

## 3. Greenstone

The Greenstone digital library software provides an open source software that empowers users to build their own digital libraries. Thus, Greenstone is a meta-software to build digital libraries (and their collections), but is not a digital library itself.

A typical Greenstone digital library installation may be structured as shown in Figure 1. A Greenstone software installation is referred to as *Greenstone server*; the computer on which the server runs is the *Greenstone host*. Each host can manage several *collections*. A collection consists of a configuration file and a number of documents (e.g., articles, music files) which form a data set (shown as squares in Figure 1). A user can gain access to the collections offered by a host via a receptionist (hatched circles in Figure 1). A receptionist can give access to several Greenstone hosts ($Receptionist\ I$ has access to $Hamilton$ and $London$, $II$ only to $London$). The receptionist in cooperation with the hosts present the user with a single access point to the collections offered by several hosts where the underlying storage and distribution structure is transparent to the user.

Collections are federated, i.e., they may reside on different hosts but have single uniform access points. They can also be distributed, i.e., a single collection can consist of data sets on different hosts. Collections can contain sub-collections: a complex collection is identified by its entry collection. For example, collection $Hamilton.D$ consists of the data set $d$ and the sub-collection $London.E$ with its data set $e$. $London.E$ is also regarded as an independent collection by host $London$ (indicated by the line from the server to the collection). Collection $London.G$ is only accessible as sub-collection of $London.F$ and not as independent collection; i.e., $London.G$ is a private collection. Collections with empty data sets but sub-collections with data sets are called virtual collections (such as $Hamilton.C$).

In all cases, the users are not aware of the internal structure of the collections. They perceive a collection as a homogenous structure with a single entry point to all data sets identified by the entry collection name. The distribution of sub-collections is also transparent to the users: a sub-collection is presented as being part of a collection regardless of the actual location of the data.

In order to understand the design of the alerting service, it is necessary to know details about the implementation of the data access to distributed collections. Users access collection data via a receptionist's interface. The receptionist talks to the respective servers via the SOAP-based Greenstone protocol (indicated by the dashed line in Figure 1). The receptionist issues a request for collection data to the Greenstone server which hosts the collection that the user wants to access. For example, to allow a user access to collection $Hamilton.D$, the receptionist issues a request towards the server on host $Hamilton$. 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



Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'05)
1545-0678/05 $20.00 © 2005 **IEEE**

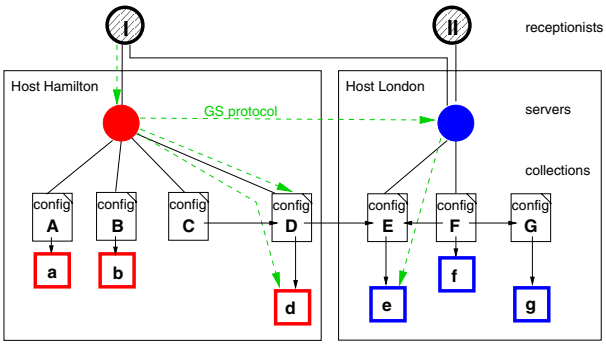


**Figure 1. Distributed Greenstone collections**

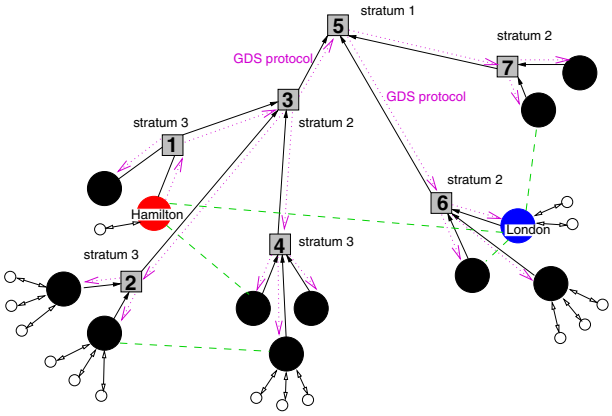


**Figure 2. Alerting using the GDS**

data set $d$. The server also learns about the existence of sub-collection $E$ on host $London$. $Hamilton$ 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 $e$. $London$ responds with 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. Further details about distributed Greenstone can be found in [1, 3].

## 4. Overall Design of GS A lerting

As argued in Section 2, it is not possible to use the GS network for distributed alerting since it is too fragmented. P2P techniques are also not suitable since that would require active interest and support of the servers (and the server administrators) for sustaining the P2P network; this cannot be expected for social and political reasons.

Instead of an overlay network based on the DL server nodes as brokers, we propose to use a mechanism adopted from distributed directory services (e.g., DNS): Auxiliary network nodes that form a maintenance network in addition to the existing scattered physical networks of DL broker nodes. Section 4.1 describes the design and usage of the Greenstone Directory Service (GDS). For alerting about distributed collections, both the directory service and the Greenstone network have to be used. The distributed alerting is described in Section 4.2.

### 4.1. Greenstone Directory Service

The conceptual organization of the Greenstone Directory Service (GDS) is depicted in Figure 2. The original network of DL hosts (black circles) is shown in dashed lines; client are shown as white circles. Collections are represented by their hosts. Events occurring within a collection on a particular Greenstone server need to be forwarded to the other servers where user subscriptions may be stored. Currently, only one Greenstone server runs on each Greenstone host. The figure shows a realistic scenario of Greenstone servers running on disconnected hosts. Most Greenstone servers are solitary installations. If a server holds distributed subcollections, it holds a direct reference to one or more other Greenstone servers (see description of communication using the Greenstone protocol in Section 3).

These sub-collection references are depicted in Figure 2 with dashed lines. We see, for example, the connection between the $Hamilton$ server and the $London$ server that has been introduced in Section 3. Each server is registered at exactly one service installation in the distributed Greenstone Directory Service (GDS). Each service installation is depicted as a shaded square with an identifying number. Each GDS installation resides on a certain stratum. A GDS primary server on stratum 1, is a computer equipped with GDS software. A GDS primary server has access to all Greenstone servers within the network following the tree towards the leaves. Other computers on stratum 2 or higher, equipped with GDS software, have similar access to all Greenstone servers in their sub-trees. They have to query their parent servers to obtain access to other branches of the tree. The GDS offers a 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 DL server forwards the message to its GDS server. The message is distributed upwards within the tree and downwards to all tree leaves.



Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'05)
1545-0678/05 $20.00 © 2005 **IEEE**

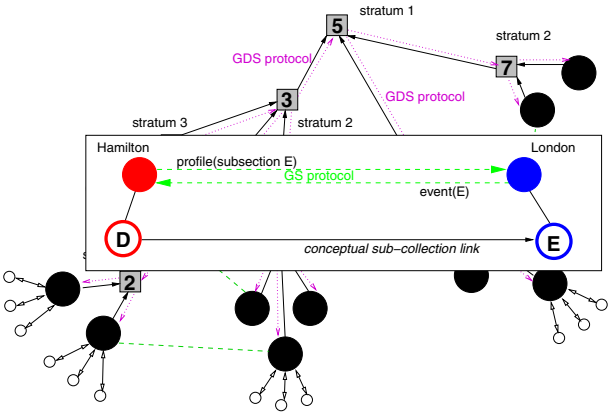


**Figure 3. Alerting using GS Network**

### 4.2. Hybrid Alerting in Greenstone

We follow a hybrid approach for distributed alerting in Greenstone: (1) for notifications about federated collections we use event flooding over the Greenstone Directory Service (broadcast), and (2) for notifications about distributed collections, we use profile forwarding using the Greenstone network (unicast or multicast).

**Federated Collections:** Users submit their profiles to a certain Greenstone server. The profiles reside at the server to which the user submitted the profile. When a collection is rebuilt, event messages are created by the collection's server. The event messages are flooded to all servers via the GDS network. Each server filters the incoming events according to the local profiles and notifies its clients accordingly. This technique is inspired by *event flooding* as proposed by Carzaniga [5]. Its implementation significantly differs from their design as here we use the Greenstone Directory Service as the communication network and not the network formed by Greenstone servers. The reasons for this design have been discussed in Section 1 (e.g., potentially disconnected network fragments and dangling profiles).

Figure 2 shows a communication example (following the dotted lines): Clients connect to Greenstone servers and define their profiles for the alerting service. The clients and their profiles are depicted as small circles connected to the servers. We assume a new collection is built at the $Hamilton$ server. Subsequently, an event message is created by $Hamilton$ announcing the documents in the new collection. The message is broadcast to all DL servers: forwarded upwards in the GDS tree and downwards towards the leaves using the GDS protocol (indicated in dotted arrows). Consider the clients connected to the $London$ server: Their profile is stored locally at the server. As soon as the event message arrives at the server, it is filtered and notifications are sent to interested clients.

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

**Distributed Collections:** If a sub-collection is rebuilt that resides on a different server than its super-collection, the server where the super-collection resides is not aware of the rebuilt sub-collection. Returning to the network shown in Figure 2, server $Hamilton$ includes a collection on the $London$ server.

In Figure 3 we see a close-up of this connection; $Hamilton$ holds a collection $D$ that includes a sub-collection $E$ hosted on the $London$ server. The connection between these two collections is shown as a solid arrow from left to right in the enlarged area at the center of the figure.

Server $Hamilton$ is not immediately aware of the rebuilt $London.E$ sub-collection and cannot, therefore, issue the event message. On the other hand, the server where the sub-collection resides on ($London$) is not aware of the collection $London.E$ being part of another collection. It can therefore only issue event messages regarding the (sub)collection as an independent public collection or as part of a local collection – i.e. changes in $London.E$ will not be visible as changes in $Hamilton.D$. Virtual collections, which hold no content of their own, but simply aggregate a number of sub-collections that actually contain content pose further problems with ensuring that event messages are actually sent when an event occurs. For example, when a sub-collection is a private sub-collection (not visible in its own right) to a virtual collection. Without appropriate strategies to forward event messages from the sub-collection to its virtual parent, no event message would by issued be either server.

Even if a message is forwarded from $London$ through the GDS network identified above, using an event message from $London$ as a notification for the server $Hamilton$ (in order to trigger a message by $Hamilton$) is not possible: Server $Hamilton$ might not recognize the newly built sub-collection because London could (as in this case) identify it by a different name.

To address this problem, an auxiliary profile is forwarded from $Hamilton$ to $London$, identifying $London.E$ as a sub-collection of $Hamilton.D$. When an event occurs within $London.E$, the forwarded auxiliary profile matches the event locally on $London$, and an event is forwarded over the Greenstone network to $Hamilton.D$. The $Hamilton$ server then broadcasts the event using the mechanisms described in the 'Federated collections' section above. For the broadcasting of the event, the originating collection is transformed from $London.E$ to $Hamilton.D$, so subsequent event forwarding will be consistent with the event having



Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'05)
1545-0678/05 $20.00 © 2005 **IEEE**

originated in the super-collection on $Hamilton$.

## 5. Profiles and Filtering Details

Greenstone is an open software also in the sense that the designer of each collection determines the collection's retrieval functionality (typically searching and browsing on various attributes and formats). For ease of use, the Greenstone alerting service mirrors this functionality. Each profile is a Boolean combination of a number of attribute-value pairs (on macro level). The term 'values' is used here in a broader sense: Values might be sub-queries (micro-level) such as: (1) a list of IDs, e.g., for hosts and documents; (2) wildcards; or (3) filter queries. The macro profile is evaluated using a variant of the equality-preferred algorithm [9].

On the micro level, it is impossible to anticipate all formats and configurations that Greenstone may support. We exploit Greenstone's openness by combining index-based filter strategies with the system's own retrieval functionalities. For alerting, search queries can be used as profile queries and document browsing is extended by a "watch this"-button which triggers an identity-centered observation. More details about the profile language and the filter implementation can be found in [19].

## 6. Routing Details

The alerting service for Greenstone 3 has been implemented in Java. The Greenstone Directory Service uses XML messaging over SOAP to provide a tree of interconnected auxiliary GDS servers. Messages are delivered using "best effort". The GDS servers implement an asynchronous and anonymous communication network that forwards messages from Greenstone servers to other Greenstone servers without the servers having to be aware of the identity of the recipient. The GDS supports broadcasting and multicasting as well as a Naming Service similar to DNS in order to allow for point-to-point communication. Further details about the implementation can be found in [3].

## 7. Discussion

*Effectiveness of our approach:* It has been argued that using profile forwarding for auxiliary profiles might lead to dangling profiles in case of network separation. In the case of the fragmented nature of the Greenstone network, we reduced the scope for dangling profiles through our general adoption of event forwarding. The case of auxiliary profiles is different and needs consideration of the nature of the (fragmented) Greenstone network. One has to note that the forwarded auxiliary profile is for a client that is a Greenstone server holding a super-collection, not a user. Each forwarded collection profile is itself unique; it exists on only one server (that hosting the sub-collection), and refers only to one other host (of the super-collection). Due to this simple constraint, when a subscription is cancelled (by a sub-collection being removed from the super-collection), ensuring deletion is straightforward.

Dangling auxiliary profiles would mean that a sub-collection is still sending alerts to the user even though the auxiliary profile has been cancelled (but not removed due to network separation). We argue that this case cannot happen or would not cause harm. A dangling auxiliary profile could have one of three reasons: (1) removed or relocated super-collection, (2) removed or relocated sub-collection, and (3) severed network connection between the two hosts. A removed or relocated super-collection (without removal of the sub-collection reference and the auxiliary profile) would be in conflict with the Greenstone collection management. Similarly, a removed sub-collection without the super-collection being informed would lead to GS conflicts. This leaves only the third case to be considered: If the network connection between super-collection and sub-collection is lost, the auxiliary profile would trigger notifications towards the super-collection only (which cannot be reached). As soon as the network connection is re-established, any deletion or update of the auxiliary profile triggered by the super-collection can be performed without users being aware of the problem. Similarly, notifications about the update of the sub-collection would be delayed until the network connection is reestablished. Then, the information can be sent to the super-collection and then distributed to the users.

*Generalization of our approach:* DL software systems such as Greenstone are meta-software where the installation determines many features of the actual run-time system. For DL meta-software, a number of general properties exist that allow one to reason about generalization of our approach to other DL meta-softwares. DL systems use protocols that provide a common set of functions (e.g. search, browse, document retrieval). Existing research in the DL community thus provides reasoning over DL systems as a whole. DL systems have also been viewed as particular forms of hypertext, database and information repositories.

At a higher level of abstraction, beyond digital libraries themselves, we may consider the problems of alerting in the case of meta-software in general. In our case, we present a model for meta-software that will also hold for other meta-software that shares the properties listed in the Introduction. Examples are meta-software for creating tailored content management systems and software for creating online learning systems. However, an open point for future research is the development of the currently immature understanding of which types or forms of meta-software may be identified.



Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'05)
1545-0678/05 $20.00 © 2005 **IEEE**

## 8. Conclusions

This paper proposed and described the design and implementation of a distributed alerting service for a distributed open source digital library software. We have shown that existing techniques cannot be simply applied to this new scenario. We proposed the use of a Greenstone Directory Service in addition to the dynamic and fragmented Greenstone Network. The alerting service combines the use of both communication networks. We support the coexistence of both paradigms of querying (implemented as search or browse) and filtering. The contributions of this paper are:

1. Design and implementation of a Greenstone Directory service for the support of distributed alerting over loosely coupled digital library servers
2. Hybrid alerting communication (routing) and filtering supporting distributed and federated collections, i.e., collections in which the documents are distributed over several hosts, federated by different DL servers.

In addition, we support alerting as continuous browsing and searching in digital collections. We evaluated the performance of the alerting service: the filtering acts as an additional step in the build process of a collection extending the overall process insignificantly. As future work, we will thoroughly evaluate the scalability of the alerting using both the GDS and the GS network; so far, initial tests have been promising. We also plan to integrate a smooth transformation of Greenstone search queries into profiles and vice versa in order to bring the user experience of the alerting service even closer to typical Greenstone interactions.

**Acknowledgements** We thank Sven Bittner for his valuable help in analyzing distributed Event Notification Services. We thank Andrea Schweer for her efforts in implementing the Alerting Service for Greenstone.

## References

[1] D. Bainbridge, G. Buchanan, J. McPherson, S. Jones, A. Mahoui, and I. Witten. Greenstone: a platform for distributed DL applications. In *Proc. of the ECDL*, 2001.

[2] S. Bittner and A. Hinze. Classification and analysis of distributed event filtering algorithms. In *Proceedings of the OTM: CoopIS, DOA, and ODBASE*, 2004.

[3] G. Buchanan and A. Hinze. A Distributed Directory Service for Greenstone. Technical Report 01/2005, CS Department, University of Waikato, New Zealand, Jan. 2005.

[4] F. Cao and J. P. Singh. Efficient event routing in content-based publish/subscribe service network. In *Proceedings of INFOCOM*, 2004.

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

[6] A. Carzaniga, M. Rutherford, and A. Wolf. A routing scheme for content-based networking. Technical report, Department of CS, University of Colorado, June 2003.

[7] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. *IEEE Trans. Softw. Eng.*, 27(9):827–850, 2001.

[8] Energy and Resources Institute, New Deli. Report on ICDL 2004, Feb. 2004. available online at http://www.teriin.org/events/icdl/icdl2004report.pdf.

[9] F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Efficient matching for content-based publish/subscribe systems. Technical report, INRIA, 2000.

[10] D. Faensen, L. Faulstich, H. Schweppe, A. Hinze, and A. Steidinger. Hermes – a notification service for digital libraries. In *Proc. of the JCSL, Roanoke, USA*, June 2001.

[11] C. P. Hall, A. Carzaniga, J. Rose, and A. L. Wolf. A content-based networking protocol for sensor networks. Technical Report CU-CS-979-04, Department of CS, University of Colorado, August 2004.

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

[13] M. Koubarakis, T. Koutris, C. Tryfonopoulos, and P. Raftopoulou. Information alert in distributed digital libraries: The models, languages, and architecture of dias. In *Proceedings of the ECDL*. Springer-Verlag, 2002.

[14] M. Koubarakis, C. Tryfonopoulos, S. Idreos, and Y. Drougas. Selective information dissemination in P2P networks: problems and solutions. *SIGMOD Rec.*, 32(3):71–76, 2003.

[15] L. Liu. Query routing in large-scale digital library systems. In *Proceedings of the ICDE*, March 1999.

[16] G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Technische Universität Darmstadt, Sept. 2002.

[17] P. Pietzuch and J. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proc. of the 22nd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, Vienna, Austria, July 2002.

[18] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In *Proc. of the Workshop on Networked Group Communications*, London, UK, Nov. 2001.

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

[20] W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proc. of the 2nd Workshop on Distributed event-based systems (DEBS '03)*, 2003.

[21] I. H. Witten, C. Nevill-Manning, R. McNab, and S. J. Cunningham. A public library based on full-text retrieval. *Commun. ACM*, 41(4):71–75, 1998.

[22] T. W. Yan and H. García-Molina. SIFT - a tool for wide-area information dissemination. In *Proc. of the Usenix'95*, Jan. 1995.

[23] A. Yazici and C. Sener. RUBDES: A Rule Based Distributed Event System. In *Proc. of the ISCIS - Computer and Information Sciences, Antalya, Turkey*, Nov. 2003.



Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'05)
1545-0678/05 $20.00 © 2005 **IEEE**