An Integrated Alerting Service for Open Digital Libraries: Design and Implementation

Annika Hinze, Andrea Schweer, George Buchanan

University of Waikato, New Zealand
a.hinze@cs.waikato.ac.nz
University of Dortmund, Germany
andrea.schweer@uni-dortmund.de
University College London, United Kingdom
g.buchanan@ucl.ac.uk

Abstract. Alerting services can provide a valuable support for information seeking in Digital Libraries (DL). Several systems have been proposed. Most of them have serious drawbacks, such as limited expressiveness, limited coverage, and poor support of federated and distributed collections.
In this paper, we present the detailed design and implementation for a comprehensive alerting service for digital libraries. We demonstrate typical user interactions with the system. Our alerting service is open to other event sources, supports a rich variety of event types, and works on distributed as well as on federated DL collections.

1 Introduction

For several years now, there has been increasing demand for a comprehensive alerting services for Digital Libraries [7, 9]. The alerting services supported by proprietary digital libraries, such as Springer Link alert or the ACM digital library TOC alert, are widely known; unfortunately, they have restricted coverage and very limited expressiveness for the subscriptions (e.g., Table-of-Contents (TOC) of a specified journal). These proprietary digital library systems can be contrasted against open digital library systems that provide extendible and flexible tools – supporting varied file formats, custom services and connection between separate libraries.

Recently, existing open digital library systems have started to incorporate alerting functionality. Most of them offer only a restricted focus (e.g., stored search on metadata). Alerting services in Digital Libraries (DL) could inform users about new collections or changes in the classification scheme of a library as well as about new or changed documents. Users express their interest in these *events* through *subscriptions*. The service filters all events and notifies users accordingly. An 'open' alerting service should be accessible to a wide variety of digital library systems and extend the flexible principles of open digital libraries briefly introduced above – e.g., by supporting a flexible model of which events may occur, consistently handle distributed and federated libraries and provide event feeds to differing alerting services. However, to provide a full range of the events that may occur in a specific digital library, the system must also be *integrated* with the DL – as explained in [5].

This paper introduces the first distributed alerting service that offers sophisticated user subscriptions over a wide range of sources without the limitations of its predecessors: The service supports a wide range of event types using content-based filtering; it is open to external event sources, and notifies users via email, web page or an RSS feed. We present a detailed requirements analysis, our alerting system in use, details of the service design and results of an evaluation. A companion paper [5] describes some of the librarianship difficulties highlighted by the work reported in this paper, omitting the technical content presented here.

The body of this paper commences with the results of our requirements analysis. This section is followed by a demonstration of our Greenstone alerting service in use (Section 3). We subsequently discuss in turn its architecture (Section 4) and detailed design (Section 5). In Section 6, we outline the user-centered and technical requirements that influenced the design of our service. Related work is discussed in Section 7. Our paper concludes with an outlook to future research.

2 Requirements

In the course of designing the Greenstone alerting service, we conducted a number of studies to identify the requirements for the system. One of the key challenges was identifying the different types of events and notifications that a comprehensive service should supply. This part of the design was driven by the results of two studies: a user survey and a claims analysis of the intended design. The latter was based on use cases identified in collaboration with the developers of Greenstone. From these sources, we also developed the user-centered functional requirements for the alerting service. The technical requirements were drawn from an analysis of existing digital libraries. Table 1 summaries the requirements we have identified. We now discuss the requirements in detail, ordered according to their sources.

General considerations: A number of general requirements can be drawn from experience with previous alerting services: Users must be able to find items of interest (F1). They must be able to create new and edit or delete existing subscriptions (F2). The providers must be able to publish descriptions of the events they offer (F3), and they must be able to send event messages to the alerting service (F4). Finally, users must be able to view their notifications (F5).

User studies: The requirements that were identified based on the results of the user studies are: A major concern was that the service could notify users about too many irrelevant events (false positives). It is especially remarkable because no option in the questionnaire corresponded to this problem. The users concerns are taken into account as the requirements that subscriptions should be as fine-grained and as similar to conventional Greenstone usage as possible. In addition

Functional Requirements	Technical Requirements
F1. find items	T1. provide content-based notifications
F2. add, edit, delete subscriptions	T2. provide customizable notifications
F3. publish event descriptions	T3. support all Greenstone event types
F4. publish event messages	T4. use familiar metaphors for user interface
F5. view notifications	T5. support different kinds of event sources
	T6. event sources on several abstraction levels
	T7. support flexibility of Greenstone set-ups
	T8. support distributed/federated collections

Table 1. Requirements for the Greenstone alerting service

to that, notifications have to be as unobtrusive as possible. Both goals can be reached by providing customizable, content-based notifications (T1 & T2).

DL Scholarship: Based on our experiences with digital libraries, we believe that the alerting service should have the following characteristics to be fully integrated into a digital library: It should provide notifications about a wide selection of events occurring in the digital library (T2 & T3). It should stay consistent with the users conceptual model of the digital library (i.e., creating a subscription and receiving a notification should be similar to using the other services of the digital library) (T4). It should integrate with the infrastructure of the digital library (i.e., seamlessly support distribution and federation of the DL) (T8).

Greenstone specific: A number of requirements were developed to address the special features of the Greenstone digital library system: Greenstone allows for the combination of different internal and external sources for a DL collection. The alerting service should be similarly open and support other event sources in addition to Greenstone DLs (T5). Greenstone is very flexible as to which document formats can be stored and retrieved, the metadata formats that can be used, the collection structure and the service configuration. The alerting service should therefore place as few constraints as possible on the configuration of the collections it can be used for (T6 & T7).

The requirement T8 creates particular challenges for the alerting service in the Greenstone context: To support Greenstone's distributed nature, the alerting service itself has to be distributed to a much higher degree than present in current alerting services in the context of digital libraries. Ideally, users should be able to create one single profile and then transparently add subscriptions for different Greenstone installations to it. In addition to that, it should be possible to subscribe to events from different collections or hosts using one single subscription (for example, notify me about all new collections).

To address requirements T2 and T3, we analyzed Greenstone 3 to identify useful event types. In the context of DLs, events refer to state changes in all objects in the DL software: such as collections, documents, and the software

Event Type	Details
software	ew release, new bug, bug resolved, new patch
host	new host, host deleted
interface	new interface, interface deleted
site	new site, site deleted
collection	new collection, collection deleted, collection rebuilt
document	new document, document deleted, content of document changed, metadata of document changed
part of document	new part of document, part of document deleted, content of part of document changed, metadata of part of document changed
service	new service, service deleted, service-specific event

Table 2. Event Types Identified for Greenstone 3

itself. We identify all objects in this context and list the creation and destruction of each kind of object (where applicable), as well as all ways these objects can change. Table 2 lists all 24 types of events we identified for Greenstone 3.

3 The Alerting Service in Use

In this section, we demonstrate the user side in two typical interactions with the alerting service: first, the initial creation of a subscription (i.e. registering an interest) and, second, the receipt of notifications that match the subscription. For clarity, we will primarily focus on the familiar events of new or changed documents; our full range of events will be introduced in Section 5.

3.1 User Side: Creating Subscriptions

Simple subscriptions: Our user is browsing a digital library collection that frequently has changes and updates made to its content (this often occurs with, e.g., WHO collections). The user has found a document that is of interest to them and that may contain additional information in the future (see Figure 1, left). In order to monitor the evolution of this document, they click on the "Watch This" button circled on the left-hand side of the interface in the figure. This simple gesture registers a subscription for the user that will now send them a notification when a change is made to the document.

To find new documents, our user may turn to classifications in Greenstone: They might find a classification that is interesting and relevant to them (see Figure 1, right). Again, they click on the "Watch This" button, and they will be subsequently notified when a change is made to the content of this classification, e.g., when a new document appears in it, a new sub-classification is added or a member document is removed.

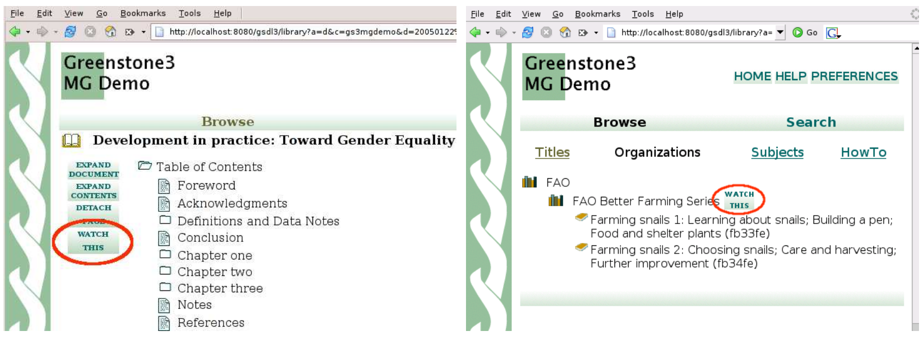

File Edit View Go Bookmarks Tools Help
http://localhost:8080/gsdl3/library?a=d&c=gs3mgdemo&d=20050122
Greenstone3
MG Demo
Browse
Development in practice: Toward Gender Equality
EXPAND DOCUMENT
EXPAND CONTENTS
DETACH
WATCH THIS
Table of Contents
Foreword
Acknowledgments
Definitions and Data Notes
Conclusion
Chapter one
Chapter two
Chapter three
Notes
References
File Edit View Go Bookmarks Tools Help
http://localhost:8080/gsdl3/library?a=
Go
Greenstone3
MG Demo
HOME HELP PREFERENCES
Browse
Search
Titles
Organizations
Subjects
HowTo
FAO
FAO Better Farming Series
WATCH THIS
Farming snails 1: Learning about snails; Building a pen; Food and shelter plants (fb33fe)
Farming snails 2: Choosing snails; Care and harvesting; Further improvement (fb34fe)

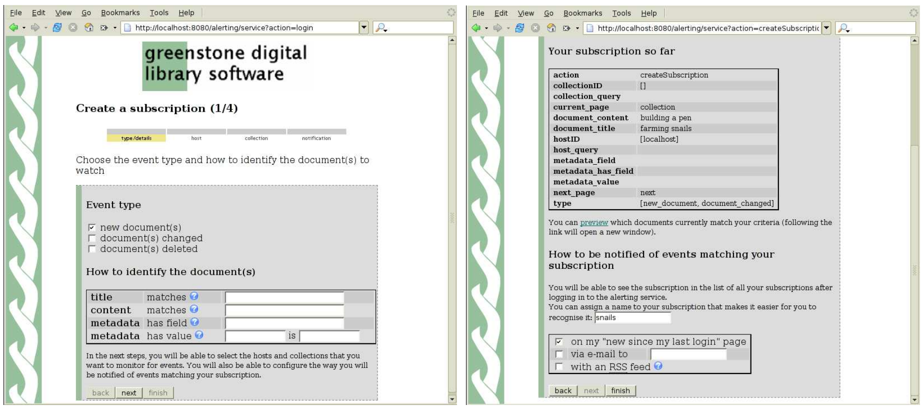
Fig. 1. "Watch this" buttons to create subscriptions shown on an individual document page (left) and a classifier display (right).

In both these cases, the alerting service simply uses this user's default preference for how to notify about the updated information. Simple subscriptions are set up in a browsing interaction style. Obviously, this kind of subscription only works for existing classifications or documents.

Complex subscriptions: Users can also create subscriptions about documents where no classification exists yet, or define sophisticated subscriptions. Here, we use a technique similar to a search query. A subscription is created in four steps. In Figure 2, we can see the first and the fourth step in the process. Firstly, users define the event type and a query that helps identify the involved documents (Step 1, see Figure 2, left). Next, the involved Greenstone hosts (Step 2) and collections (Step 3) have to be defined (by selecting from a list or specifying part of the name). Finally, the means of notification are defined as shown in the screenshot in Figure 2, right. After defining a subscription, notifications about the events may be received by the user; this is described in the following subsection. Users can always edit or delete their subscriptions with immediate effect.

3.2 User Side: Receiving Notifications

Greenstone (GS), being a full-text digital library system, works with periodic explicit rebuilds (re-indexation) initiated by the collection administrator. Thus, the typical pattern of updates to the library are occasional and large-scale rebuilds, rather than frequent changes to individual documents where updates are performed immediately on the receipt of a changed document. Our Greenstone alerting service is triggered by each rebuild; changes might have occurred in the rebuild that subscribed users may be interested in. Any user who has at least one subscription involving the rebuilt collection may be due to receive notifications about changes to the library.


File Edit View Go Bookmarks Tools Help
http://localhost:8080/alerting/service?action=login
greenstone digital library software

Create a subscription (1/4)

type/details
host
collection
notification

Choose the event type and how to identify the document(s) to watch

Event type

[✓] new document(s)
[ ] document(s) changed
[ ] document(s) deleted

How to identify the document(s)

title	matches		
content	matches		
metadata	has field		
metadata	has value		is

In the next steps, you will be able to select the hosts and collections that you want to monitor for events. You will also be able to configure the way you will be notified of events matching your subscription.

back
next
finish
File Edit View Go Bookmarks Tools Help
http://localhost:8080/alerting/service?action=createSubscriptic

Your subscription so far

action	createSubscription
collectionID	[]
collection_query	
current_page	collection
document_content	building a pen
document_title	farming snails
hostID	[localhost]
host_query	
metadata_field	
metadata_has_field	
metadata_value	
next_page	next
type	[new_document, document_changed]

You can preview which documents currently match your criteria (following the link will open a new window).

How to be notified of events matching your subscription

You will be able to see the subscription in the list of all your subscriptions after logging in to the alerting service.
You can assign a name to your subscription that makes it easier for you to recognise it: snails

[✓] on my "new since my last login" page
[ ] via e-mail to
[ ] with an RSS feed

Fig. 2. A user's subscription displayed for editing; Steps 1 (left) and 4 (right) of four.

As discussed before, users can configure the delivery method for notifications in their subscription configuration. At present, we support email and RSS feed notification for personal delivery. An example RSS feed is shown in Figure 3, accessed using the Liferea feed reader (`http://liferea.sourceforge.net`).

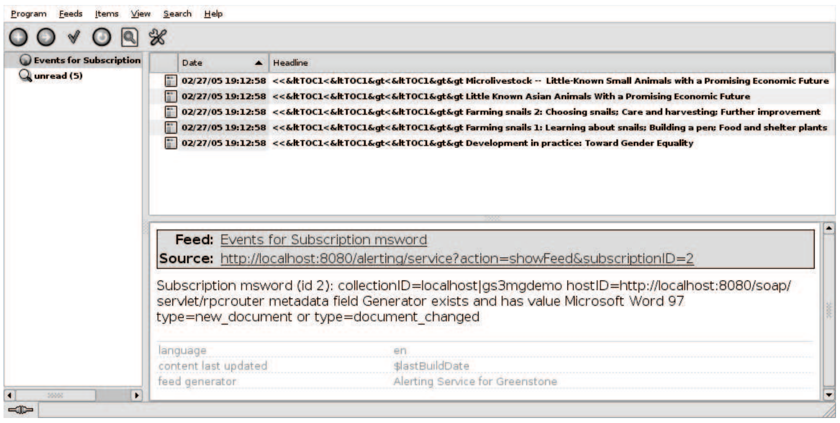
Alerts may also be displayed in the Greenstone library interface in a general or personalized way. The general interface may provide a 'new accession page' and 'new/changed' highlight for particular documents. The personal page may display a user's recent notifications after the user has logged in to Greenstone.

4 Architecture of the Alerting Service

We now discuss the server side of the alerting service, providing an overview of our alerting architecture and its distribution. Detailed descriptions of the components and their design will be given in the following section. We chose the latest generation of the Greenstone DL software as the basis for building our system. Greenstone is a well-established software system for delivering digital library services; the alerting service builds upon Greenstone's modular architecture [2].

General Architecture: The general architecture for the alerting service is shown in Figure 4. Existing DL components in Greenstone are identified in white, whilst the new elements are highlighted in gray. The alerting sequence [8] comprises four steps that can be seen at the bottom of the figure: (1) rebuild and trigger of alerting service, (2) observation of event messages, (3) filtering of event messages according to user subscriptions, and (4) notification.

Note that the *Observer* component in itself is distributed: It consists of a generic observer (within the alerting service in Figure 4) and an event detection plug-in for each event type (shown in Figure 4 as last phase in the built process).



Date	Headline
02/27/05 19:12:58	<<&ltTOC1<&ltTOC1&gt<&ltTOC1&gt&gt Microlivestock -- Little-Known Small Animals with a Promising Economic Future
02/27/05 19:12:58	<<&ltTOC1<&ltTOC1&gt<&ltTOC1&gt&gt Little Known Asian Animals With a Promising Economic Future
02/27/05 19:12:58	<<&ltTOC1<&ltTOC1&gt<&ltTOC1&gt&gt Farming snails 2: Choosing snails; Care and harvesting; Further improvement
02/27/05 19:12:58	<<&ltTOC1<&ltTOC1&gt<&ltTOC1&gt&gt Farming snails 1: Learning about snails; Building a pen; Food and shelter plants
02/27/05 19:12:58	<<&ltTOC1<&ltTOC1&gt<&ltTOC1&gt&gt Development in practice: Toward Gender Equality

Feed: Events for Subscription msword
Source: http://localhost:8080/alerting/service?action=showFeed&subscriptionID=2

Subscription msword (id 2): collectionID=localhost|gs3mgdemo hostID=http://localhost:8080/soap/servlet/rpcrouter metadata field Generator exists and has value Microsoft Word 97 type=new_document or type=document_changed

language	en
content last updated	$lastBuildDate
feed generator	Alerting Service for Greenstone

Fig. 3. A Greenstone RSS feed shown in a feed reader

The alerting sequence is initiated when a collection in the library is (re-)built and changes in the library's indexes and content are identified. Each event is reported to the *Observer* process, which prepares them for processing. The *Filtering* phase then takes each event in turn, and matches it against the filter's own index of subscriptions. If a match is found between a subscription and an event, a notification should be sent to the user who owns the subscription. A match is sent to the *Notifier*, which creates a notification about the event message and sends it to the user according to the user's preferences. The heart of this apparently simple sequence is the *Filtering* phase, which will be discussed in detail in Section 5.

Distributed Architecture: The Greenstone alerting service can be distributed. A distributed architecture for the alerting service is required in three cases: (a) a number of different digital libraries use a common alerting service; (b) a library's content is distributed; or (c) the alerting service is on a separate computer to the DL server. Greenstone itself is designed to be distributed if required [1]. Therefore, Case (c) is simple since any connection in our architecture (see Figure 4) may be between processes on separate machines. Beyond this basic separation, we make a distinction between *federated* collections, Case (a), and *distributed* collections, Case (b). For the forwarding of event messages to distributed or federated Greenstone servers, we have created the *Greenstone Directory Service (GDS)*. The GDS provides a common message-forwarding–infrastructure for all alerting distribution scenarios. Here, we give a brief resume of GDS – for complete details see [4]. The GDS is constructed as an independent network of Directory nodes that together form a tree structure. Each tree of GDS nodes provides a separate community, which can send and receive messages to and from any digital library registered with a node of the GDS tree. GDS messages are transmitted in an XML formwat across TCP/IP connections. Each node re-

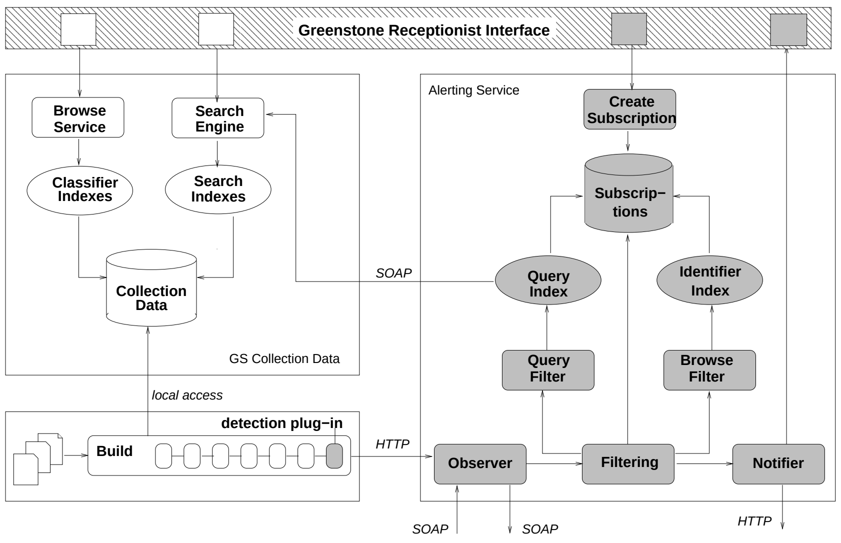


Fig. 4. General architecture of a local Greenstone installation (left) with build process (bottom left) and alerting service (right)

ceives messages from its children, and distributes received messages forwards to all its other children, and upwards through the tree for propagation across the whole GDS network of which the node is a member. Messages are transmitted through the GDS network when a collection is rebuilt – being sent into the GDS server with which the host digital library server is registered. Note that a GDS transmission node does not itself match or filter events against subscriptions – it purely acts as a transmitter in a communication network. The new build process we implemented in Greenstone supports notification of changes made during a collection rebuild through the GDS service, and we have also implemented the receipt and processing of remote collection changes against locally stored subscriptions in a discrete subscription filtering client application. Other digital library systems can readily be extended to support the transmission of change messages into the GDS network – it is not specific to the Greenstone software. Support for EPrints and D-Space – both popular digital library systems which we will meet later in this paper – is anticipated in the near future.

The case of federated collections is shown in Figure 5. Two Greenstone hosts are shown with their Greenstone software and alerting services. The left-hand Greenstone server hosts a collection that the two subscriptions A (left) and B (right) are interested in. When a change occurs in the collection (by rebuilding the collection – see Step △1), the last phase of the build process (i.e., the detection plug-in) triggers the observer (Step △2), which, in turn, forwards the event message to the local filter component and broadcasts it also on the Greenstone Directory Service (Step △5). Locally, the event is filtered according to

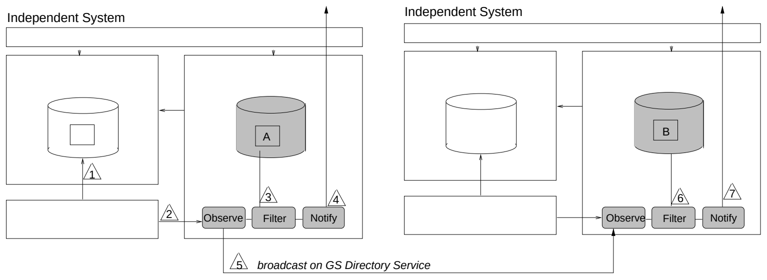
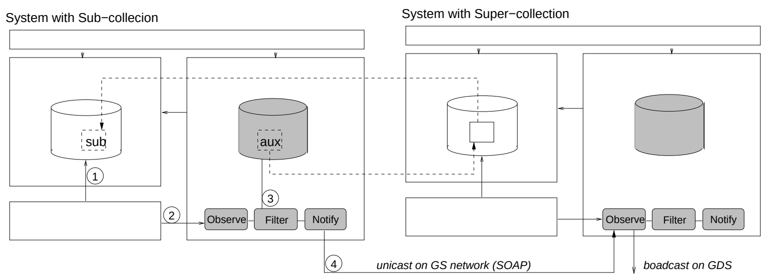


Fig. 5. Federated collections on distributed Greenstone installations. Two independent systems shown (simplified).

subscription A (Step △3) and the respective user is notified (Step △4). For the distributed case, every alerting process on a Greenstone host connected to the Greenstone Directory Service receives the broadcast event message. We now follow the alerting process through the system shown the right hand side of Figure 5: The observer receives the event message and forwards it to the filter. The event is filtered according to subscription B (Step △6) and its user is notified (Step △7). We can see that the subscriptions are kept locally at the alerting server that initially receives them.

The case of distributed sub-collections on a separate server is handled differently (see Figure 6): The super-collection (right) holds a reference to its sub-collection (left), indicated by a dashed line between the GS collection databases. Sub-collections have no knowledge of their super-collections. Consequently, they cannot send out messages referring to the changed collection (the sub-collection might have a different local name); what is needed is an event message that refers to the super-collection. The alerting system creates an auxiliary subscription for the sub-collection (left), which refers to the super-collection, represented by a dashed line. On rebuild of the sub-collection (Step ①), the local observer



System with Sub−collecion
sub
aux
1
2
3
Observe
Filter
Notify
4
unicast on GS network (SOAP)
System with Super−collection
Observe
Filter
Notify
boadcast on GDS

Fig. 6. Distributed collections on distributed Greenstone installations. System with sub-collection (left) and system with super-collection (right).

is triggered (Step ②). The matching auxiliary subscription is identified (Step ③) and a notification is sent to the super-collection's server (Step ④) via the SOAP-based Greenstone network. The Greenstone network only connects servers with sub/super-collections. The observer on the super-collection side receives the message and identifies the respective super-collection. It then sends out an event message for the super-collection over the GS Directory Service. The further process is identical to the event handling in the distributed case as described before (starting at Step $\overset{5}{\triangle}$). For sub/super-collections, subscriptions are forwarded, in opposition to event forwarding in the distributed case.

5 Design Details

This section describes in detail the components of the Greenstone alerting service introduced in the previous section. The design of our alerting service supports 30 different event types, for instance: document events, metadata events, collection events, index events, category events, classifier events, system events, and software events. The event types observe different actions (such as new, changed, deleted items) supporting different ways of identifying the affected items. For conceptual clarity of description, here we focus on new and changed documents. For comprehensive coverage of the service's design, refer to [10].

5.1 Design: Creating Subscriptions

The Greenstone alerting service provides a web interface to create subscriptions. All subscriptions are conjunctions of predicates. Predicates have a type, a field name, and a value. There are four different kinds of predicates: Equality predicates, metadata predicates, substring predicates, and query predicates. Each predicate refers to a single field name. If more than one predicate in a subscription refers to the same field, these are evaluated as disjunctions.

Equality predicates are currently used for identification of hosts, collections, and documents. Substring predicates are currently only used for hosts and collections. Metadata predicates refer to metadata fields and vales. Metadata predicates are satisfied if the document has the specified metadata field and the specified value is equal to that of the metadata field. The field name may be freely defined by the user. Query predicates can be defined for title and full-text of documents; they have free-defined Greenstone queries as values.

5.2 Design: Observing Events

Each collection in Greenstone consists of the documents' full-texts and accompanying metadata. During a rebuild, this information is processed into the standard METS format and stored in a relational database. Classification and content indexes are also generated for the collection. Each collection and each document has three time-stamps. For alerting, the following time-stamps are relevant: `accession` date and last `(re)build` for a collection, and time-stamps for `accession`, `modification` and `(re)indexed` for documents.

When a collection is rebuilt, the collection, new, and changed documents receive the same `(re)built/(re)indexed` time-stamp. New documents are identified by their `accession` time-stamp being identical to their `(re)indexed` time-stamp. To detect changed documents, the `modification` and `(re)indexed` time-stamps are compared. On each rebuild, an alerting sequence is initiated. Greenstone 3 uses incremental rebuilds; the detection plug-in at the end of the build process detects the events of changed or new documents. It then creates event messages containing attribute-value–pairs. The event message holds information about the type of event, the affected document's ID, and the collection ID. The message is sent via HTTP to the observer component of the alerting service.

The generic observer component provides an interface for internal and external event messages: messages from local builds are received through HTTP, whilst events from builds on other hosts are received through SOAP. The event messages are passed on to the filtering component of the alerting service.

5.3 Design: Filtering Event Messages

The filter component tests incoming event messages against the subscriptions stored in the database. We use a hybrid implementation of the equality-preferred–algorithm, which is in turn an extension of the counting algorithm [6]: The counting algorithm tests each predicate in a set of subscriptions only once, counting the number of successful predicate matches for each subscription. A given subscription is matched by the incoming event message if its number of matched predicates equals the total number of its predicates. The equality-preferred–algorithm tests all equality predicates first and then proceeds to further predicates for the subscriptions using the counting algorithm.

We use a hybrid equality-preferred–algorithm in three phases. The filtering component holds a special index of subscriptions, clustering them according to their equality predicates, i.e., all subscriptions with equality predicates regarding the same set of field names are clustered together (see identifier index in Figure 4). In Phase 1, the corresponding clusters for the incoming event are identified. Corresponding clusters are those which refer to fields that are contained in the event message. Then, the equality predicates in these clusters of subscriptions are evaluated (using the clusters' hash indexes on the values). The result is a list of partially matched subscriptions.

In Phases 2 and 3, all other predicates are tested using the counting algorithm. In Phase 2, all non-equality predicates are filtered and a counter is maintained for each affected subscription. Non-equality predicates are metadata predicates, substring predicates, and query predicates. For query predicates, we use collection-inherent DL searches for the stored queries (see query index with access to GS search engine in Figure 4, left). Similarly, for metadata predicates we use Greenstone's metadata retrieve service. The Greenstone alerting service determines the appropriate query service offered by a collection by applying heuristics for typical queries on a field, e.g., a predicate regarding the title field uses a title search in a collection. If no appropriate service is found, the predicate is not satisfied.

In Phase 3, the number of matching non-equality predicates are compared to the total number of non-equality predicates in each partially matched subscription. In addition, all subscriptions without equality predicates are considered. Information about matching subscriptions is written to the database to be used for notifications.

Our design constitutes a novel approach to combine filter and search functionalities; this is necessary because possible search methodologies for future collections may not be known at the time when the subscription is defined.

5.4 Design: Creating Notifications

Notifications can be delivered to the user in a general or personalized way. General delivery uses, e.g., a 'new accessions' page in the Greenstone interface, and 'new' highlights for collections or documents in the interface. We maintain a Greenstone 3 alerting module for presentation of general notifications; the GS3 user interface is created from such modules dynamically at runtime [2]. Each interface module (including the alerting module) communicates with the core DL interface code through SOAP. These general notifications require that recent notifications are recorded in Greenstone's relational database for future use.

Alternatively, the users may receive notifications through a message (email) or RSS feed. Implementation of these notification is straightforward, and require no further explanation. An example has been shown in Figure 3.

6 Evaluation

In this section, we present the results of our evaluation of the alerting service. This section consists of two principal parts: first, the discussion of the compliance of our design with the original requirements introduced earlier in this paper; second, an analysis of the performance of the system in use.

6.1 Evaluation: Design

The requirements for our alerting service were introduced in Section 2. We will first discuss the functional requirements F1 to F5: The service supports the finding of items through a simple addition to the standard DL build process [5]. We have demonstrated the editing of subscriptions (F2), publication and viewing of events (F3–F5) in Section 3.

Research in event-based systems has indicated that users frequently have problems defining effective subscriptions to represent their interests (reflected in requirements F2 & T4, see Table 1). To overcome the syntactic problem of defining even basic queries, we introduced simple interface features such as the "Watch This" button. For sophisticated subscriptions, we provide an advanced query subscription interface. The learning demand on the user is reduced by providing an analog of Greenstone's interactions, mirroring browsing and querying. Further user testing needs to be conducted to refine this approach.

The technical requirements T1 to T8 are also addressed by our design. We use the digital library itself to determine content-based events (T1), and as seen in Section 3 customizable notifications (T2). Our close relationship to the existing DL system readily supports a user interface similar to the familiar DL controls (T4) and a range of events consistent with the capacities of the underlying DL systems (T3). The alerting service's open format for events readily provides open access to a variety of event sources (T5) and levels of abstraction (T6). Our use of subscription forwarding (in the case of distributed libraries) and event forwarding (for federated libraries) ensures consistency where such collections exist (T8) and a variety of DL configurations (T7) – in the latter case supported further by our open event format.

6.2 Evaluation: Performance

We wished to study the performance of the distributed alerting service in practice. There were three separate areas that we wished to evaluate in terms of performance: event detection, filtering of incoming events against locally stored subscriptions, and finally the distribution of events across the Greenstone Directory Service network.

Before reporting our results in detail, it is worth clarifying the rate of change that may be expected in a digital library. The Humanity Development Library (or HDL) is produced by the United Nations and distributed on CD-ROM. The current public distribution contains over 160,000 separate pages, each indexed as a separate document. Of this material, some 50% has been updated or added since the first version seven years ago. In other words, the number of items changed per day is somewhat under 50. This rate of change is typical for the various United Nations collections that are supported by Greenstone, and the HDL is of median size. These UN collections have a higher change rate for existing documents than is typical for most digital libraries. It follows from this sort of volume of change that the challenge for the distributed alerting service comes less from the frequency of changes in each library collection, but rather from the potentially vast distribution/federation, the number of libraries involved, and the number of profiles.

Event Detection: For the first tests of the alerting service, we wished to verify our expectations of the time costs for detecting changes in the digital library – an area where existing data gave little indication of what order of performance one ought to expect.

Detecting a change in a document, or in a classification requires a series of simple database lookups that we expected to be in the order of $O(p)$, where p is the number of document properties (e.g., metadata such as document title – see Table 2). Running a sample build over a collection comprised of a mixture of large plain text, HTML and XML documents (of book size) resulted in an underlying build time of c. 15 seconds per document (mean=15.2s) of which c. 0.2 seconds (mean=0.16s) was spent identifying which, if any, change messages should be forwarded for this document. Clearly, event detection adds only a small cost to the indexation of a document in the library. We believe that this performance can be slightly improved with further optimisation of the build code.

Local filtering: Secondly, the issue of profile matching required study – identifying the process load of processing incoming events against the user profiles stored on a local Greenstone server.

The performance of the counting and equality-preferred algorithms is well known. However, in our implementation we extended the original algorithm, which compared only numbers. We had to support full-text search and string comparison to satisfy the full range of events for Greenstone and other digital libraries. Neither of these forms of comparison have any relationship with the number of subscriptions to be matched, so we should still anticipate overall performance characteristics of the local filtering to be similar to the original algoriths [6]. The primary expectation from the earlier implementation should be for performance to be in the region of $O(s)$ where s is the number of subscriptions (or, more precisely, unique subscription predicates). Fabret et al's algorithm evaluates any predicate (e.g., "title includes Shakespeare") only once, even if the predicate is included in a number of different subscriptions. In our tests, we followed their model of considering the performance of their algorithm on the number of unique predicates, rather than (more complex) subscriptions.

We conducted a small experiment to evaluate the local filtering performance – varying the number of predicates in the profile database from $1,000$ to $100,000$ unique predicates. Predicates were of equal portions of the different available types, created using simulated subscription data. A digital library subscription will typically include three or more predicates (in our simulated test data, an average of four) – e.g., a subscription may list an event type, collection, author name and subject field. At a worst case, $100,000$ predicates would typically represent $25,000$ subscriptions, if every predicate of every subscription were unique. Real life data would certainly include some predicates that appeared in more than one subscription, increasing the number of subscriptions that this would represent.

The time taken to match all predicates against a single event varied linearly from 0.14s (with 1000 predicates) to 0.44s (10,000 predicates). Approximately 0.10s was fixed overhead. These findings are consistent with those of Fabret et al. However, this test does not represent the whole picture – full-text retrieval predicates are much more costly to evaluate than simple string matching, and we wish to undertake further study to distinguish the costs of different styles of predicate. Unfortunately, we have little data at present to identify the relative frequency of particular types of predicate (tests of document metadata, full-text, etc.) under actual use.

However, the results above demonstrate that the matching of events against user subscriptions is scaleable and consistent with known state-of-the-art algorithms.

Unique predicates (subscriptions)	Time Taken (in sec)
1000 (256)	0.142
2500 (640)	0.182
5000 (1280)	0.243
7500 (1920)	0.338
10000 (2560)	0.437
25000 (6400)	0.783
50000 (12800)	1.465
75000 (19200)	1.981
100000 (25600)	2.479

Table 3. Results of local filtering tests for Greenstone Alerting Service

Event Distribution: Given the tree structure of the Greenstone Directory Service, one critical limitation could be the throughput capacity of the single node at the root of the tree, through which every event would have to pass. Therefore, our first point of concern was the nominal capacity in practice for a single node within the GDS to receive and forward messages. Using event forwarding over the GDS network (as described in Section 4), each change in a collection is received only once by a network node, and forwarded once to each immediate child node, and once to its parent node (excepting the case of the root node). Note that GDS nodes do not filter events, but transmit them onwards through the network without further processing. Thus, the primary limitation is in fact the rate of *output* to other nodes.

For testing purposes, the GDS network was run on a small cluster of servers at University College London, consisting of two Apple Mac OS-X computers, two Linux servers and two machines running Windows XP. Three computers ran as GDS servers only, three as Greenstone servers producing Greenstone Alerting messages. Two GDS servers were registered with the third, which acted as the root server. The GDS root server machine was an Apple Mac-OS X G5 computer with 1Gb of main memory. The computers were connected locally through a 10Mbit network, with two (Linux) machines running at a remote site at Middlesex University (UK).

As shown in [3], even such a small network topography can be used to sucessfully test distributed alerting services, as it has the advantage of a real-world test over a simulation.

We expected the time cost of the distribution of events to be small, but limited by the available network bandwidth and the overhead of establishing connections between GDS nodes. We achieved average point-of-origin to remote (off site) recipient transmission times of 2.3 seconds per event, but further testing indicates that the current implementaton could be significantly improved – e.g., at present each message is transmitted as a new, separate connection, and this is clearly wasteful.

7 Related Work

In this section, we review previous work for alerting in digital libraries in proprietary systems, in open DL systems, and in mediator approaches.

Alerting in Proprietary DL Services: Individual alerting services are offered by publishing houses, such as Springer Link Alert (via `http://springerlink.metapress.com`), ACM Table-of-Contents Alerts (via `http://portal.acm.org`), and Elsevier Contents Direct (`http://www.contentsdirect.elsevier.com`). These are solitary, centralized services that neither cooperate with other services nor openly support independent digital libraries. These services provide simple email notifications about new volumes published by the company, rudimentarily tailored to the user's interests. Only coarse-grained selectivity is offered, which may result in readers obtaining a high proportion of notifications of low relevance. Advanced subscriptions, e.g., regarding library organization or classification of documents are not supported. Unlike our Greenstone alerting service, the systems are not open to additional event sources.

Alerting in Generic DL Systems: So far, only two of the popular generic DL systems provide alerting features – D-Space and EPrints. D-Space (`http://www.dspace.org`) is being developed as a reference model for document management systems, and supports the storage and retrieval of electronic documents. Readers using a D-Space server can place a subscription on a specific collection, which then waits for any documents to be added to the collection. No additional constraints can be added to a subscription – the subscriber is simply emailed a notification each day listing all new documents added to the collection. This can be compared to a simple 'watch-this collection' subscription without further filtering in our alerting service.

EPrints (`http://www.eprints.org`) is a simple open source system for providing an internet-accessible document repository. EPrints supports simple subscriptions that alert a reader when a matching document is inserted into the EPrints repository; matches are made against the metadata fields of a document. In the Greenstone alerting service, this can be compared to a basic metadata subscription using a query that is restricted to new documents.

The engineering of the incorporation of the alerting services in EPrints and D-Space has been reported only briefly in the available literature. In contrast to the GS alerting service, their subscription systems are deeply embedded in the DL implementation.

Mediating Alerting Services: Only a few systems have been developed to support open heterogeneous document collections for publish/subscribe features. Here we focus on the two systems that target at DLs: Hermes and Dias.

Hermes [7] is an integrative system that covers heterogeneous services and event sources. Subscription definition focuses on typical queries regarding scientific publications, such as authors, title, or keywords. The service operates independently of any library implementation, using (active) email or (passive) web pages for information access. Typically, such an alerting service would be operated by a scientific library (secondary provider) as a service for its users, notifying about documents provided by primary providers. Unlike our Greenstone alerting service, Hermes only aggregates notifications from different sources and is limited by the underlying types of alerts that it receives. It was this restriction that motivated the work presented in this paper.

Dias [9] adopts the basic ideas of Hermes. Dias is a distributed system based on peer-to-peer communication. The data model of Dias is based on simple free-text documents. Dias's subscriptions support Boolean queries with proximity operators. We see this text-focussed approach as too limited for an open digital library supporting collections of arbitrary document types (e.g., music, pictures, text documents).

Tools such as Hermes and Dias suffer significantly from not being integrated with the digital library. For example, observation of events and access to documents is problematic. In addition, identifying different versions of the same work is particularly difficult without explicit knowledge of the underlying structure of collections or a set of valid document identifiers. Similarly, approaches that rely on poor sources such as the active support from publishers and DLs or on monitoring publishers' web-pages to extract event information cannot hope to lead to sophisticated event notifications.

Summarizing the related work, most existing systems only support the detection of new documents. EPrints additionally supports changed documents. Advanced subscriptions, e.g., regarding indexes or classifications, are not available. Often systems are implemented in a centralized manner. Support for federated or distributed collections could not be found. Only mediating systems support open event sources; unfortunately they receive only poor support and are extremely limited in their access to pertinent and detailed data.

8 Conclusions

This paper proposed the design and implementation of a comprehensive alerting service for digital libraries. We have shown that existing alerting services for DLs have considerable shortcomings: limited types of supported events; limited range of subscription model and notification options; no support for distributed collections; restricted support for federated collections.

To address these limitations, we presented the detailed design of the Greenstone alerting service, describing both stand-alone and networked implementations. The Greenstone alerting service supports a much wider range of event types than previous systems, and supports events in federated and distributed collections. Our alerting service is open to other event sources and providers.

Our design for a comprehensive alerting service can readily be applied to a wide range of DL systems, as it reuses existing DL components, and requires only protocol-level access to the DL to which it is attached. In addition, by using existing DL system functionality, it minimizes both programming and run-time demands of the alerting service. Using well understood principles from event-based systems, efficiency can be achieved without building extensive specific indexes for alerting. We demonstrated the scalability of our approach through a series of test of both the distributed event forwarding system and the subscription filtering component.

We plan to evaluate further performance optimization strategies of the filter algorithm. We also plan to further analyse the scalability of the distributed service under a high load of subscribers, particularly when matching a high proportion of content-text subscriptions. Initial results are promising, but we wish to scrutinise this particular issue further to ensure scalability. Another extension involves a modification of the Greenstone protocol: We plan to develop a structured methodology to determine the appropriate query service to be used for evaluating a given filter predicate.

Acknowledgements This work was supported by the University of Waikato and EPSRC grant (GR/S84798).

References

1. D. Bainbridge, G. Buchanan, J. R. McPherson, S. Jones, A. Mahoui, and I. H. Witten. Greenstone: A platform for distributed digital library applications. In *Proceedings of the ECDL*, Sept. 2001.
2. D. Bainbridge, K. J. Don, G. R. Buchanan, I. H. Witten, S. Jones, M. Jones, and M. I. Barr. Dynamic digital library construction and configuration. In *Proceedings of the ECDL*, Sept. 2004.
3. S. Bittner and A. Hinze. Classification and analysis of distributed event filtering algorithms. In *Proceedings of COOPIS*, October 2004.
4. G. Buchanan and A. Hinze. A distributed directory service for Greenstone. Technical Report 1/2005, Department of Computer Science, University of Waikato, New Zealand, Jan. 2005.
5. G. Buchanan and A. Hinze. A generic alerting service for digital libraries. In *Proceedings of the JCDL*, June 2005.
6. F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Efficient matching for content-based publish/subscribe systems. Technical report, INRIA, France, 2000.
7. D. Faensen, L. Faulstich, H. Schweppe, A. Hinze, and A. Steidinger. Hermes – a notification service for digital libraries. In *Proceedings of the JCDL*, June 2001.
8. A. Hinze and D. Faensen. A Unified Model of Internet Scale Alerting Services. In *Proceedings of the ICSC (Internet Applications.)*, Dec. 1999.
9. 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*, Sept. 2002.
10. A. Schweer. Alerting in Greenstone 3. Master's thesis, University of Dortmund, Germany, May 2005.