Making LDAP Active with the LTAP Gateway: Case Study in Providing Telecom Integration and Enhanced Services

Robert Arlein, Juliana Freire, Narain Gehani, Daniel Lieuwen, and Joann Ordille
Bell Labs/Lucent, 700 Mountain Ave, Murray Hill, NJ 07974
{rma,juliana,nhg,lieuwen,joann}@research.bell-labs.com

Abstract

LDAP (Lightweight Directory Access Protocol) directories are being rapidly deployed on the Web. They are currently used to store data like white pages information, user profiles, and network device descriptions. These directories offer a number of advantages over current database technology in that they provide better support for heterogeneity and scalability. However, they lack some basic database functionality (e.g., triggers, transactions) that is crucial for Directory Enabled Networking (DEN) tasks like provisioning network services, allocating resources, reporting, managing end-to-end security, and offering mobile users customized features that follow them. In order to address these limitations while keeping the simplicity and performance features of LDAP directories, unbundled and portable solutions are needed.

In this paper we discuss LDAP limitations we faced while building an LDAP meta-directory that integrates data from legacy telecom systems, and how LTAP (Lightweight Trigger Access Process), a portable gateway that adds active functionality to LDAP directories, overcomes these limitations.

KEY WORDS: active databases, data integration, MetaComm, directories, Directory Enabled Networking (DEN), Lightweight Directory Access Protocol (LDAP), PBX, Lightweight Trigger Access Process (LTAP), portable triggers, telecom

1 Introduction and Motivation

LDAP (Lightweight Directory Access Protocol) directories [15, 31] are being rapidly deployed on the Web. They are currently used to store data like white pages information, user profiles, and network device descriptions. Compared to current databases, LDAP directories have better support for handling scale and heterogeneity [17]. However, their lack of support for standard database functionality such as triggers and concurrency control hampers their use in more sophisticated tasks, such as Directory Enabled Networking (DEN) services.

DEN is an enabling technology for a wide variety of tasks, including reporting, managing end-to-end security, and offering mobile users customized features that follow them [14]. While this technology is not limited to LDAP directories or to any particular standard, it is frequently associated with the effort by equipment and software vendors to standardize LDAP schemas to support Directory Enabled Networking. The DEN standardization proposes the use of a single logical source for needed information and encourages storage of data in a standard format to enable interoperability and Value Added Reseller innovation.

As part of Lucent's DEN initiative, we built the MetaComm system. MetaComm integrates data from multiple devices into a meta-directory, allowing user information to be modified through the directory using the LDAP protocol as well as directly through two legacy devices: a Definity® PBX and a voice messaging system. In order to prevent data inconsistencies, updates to any of the systems must be reflected appropriately in all the other systems. For example, when a new employee is hired and her information is added to the directory, changes to a variety of devices are made automatically. By providing a simpler, unified interface to data stored in telecom devices, MetaComm greatly simplifies access to this data, reduces the need to manually re-enter data in multiple devices, and also reduces data inconsistencies across repositories.

In order to provide such functionality, MetaComm required a directory server with triggers so that it can identify relevant directory updates in a timely manner. Without triggers, MetaComm would need to use polling to discover updates — increasing the resources required for update identification while reducing the timeliness of the discovery. However, even though a large amount of work has been done in the area of standardizing the LDAP schemas that will be used to provide these services, other crucial areas like support for triggers have yet to be standardized[1]

Our challenge was then to come up with a solution that is portable (*i.e.,* works with any LDAP server) and efficient. Existing approaches to add trigger functionality to database and directory servers require either:

- modification to system internals (*e.g.,* adding either a full-fledged trigger system or associating plug-ins with directory commands),
- knowledge of proprietary log formats, or
- periodic polling of database state.[2]

In MetaComm we used LTAP (Lightweight Trigger Access Process)[3], a gateway that adds active functionality to *any* LDAP server. LTAP does not require the use of any of the above methods — it adds active functionality to LDAP without requiring any proprietary extensions to the protocol, a key advantage. Thus, LTAP can be used to improve interoperability and portability. Another advantage of LTAP is that by unbundling triggers, users that do not need trigger functionality need not pay for any added overheads. For instance, a read-only replica of MetaComm's LDAP directory does not need to be front-ended by LTAP.

In what follows we describe our experiences in using LDAP as a meta-directory, its limitations and our approach to overcome them. The paper is organized as follows. Section 2 contains an overview of the LDAP protocol. Section 3 describes the overall architecture of MetaComm. Section 4 describes the LTAP gateway and its functionality by way of an extended example based on trigger used in MetaComm. Section 5 summarizes our experiences in building MetaComm, and the implications of our gateway approach for adding needed but missing functionality to LDAP. These experiences are key lessons learned in this paper. They will be valuable to those building similar systems. Section 6 contains related work, and conclusions and future work are in Section 7.

2 LDAP (Lightweight Directory Access Protocol) Overview

LDAP is a widely deployed directory access protocol with implementations by a large number of vendors (see reference [16] for a partial list). LDAP can be thought of as a very simple database query and update protocol. From a database perspective, LDAP has some beneficial properties, for example, it deals

[1]The IETF LDAP Extension Group, ldapext, has on-going work to standardize LDAP triggers [20].

[2]The proposals on persistent search at the ldapext group will supply continuous polling of database state efficiently. Once available, persistent search will provide an attractive platform for implementing triggers that fire after an update has been executed. However, persistent search has limitations, for example, it cannot be used to build triggers that fire before an update is executed. .

[3]LTAP can be downloaded from `http://ltap.bell-labs.com`

```
objectclass person
    oid 2.5.6.6
    superior top
    requires
        sn,
        cn
    allows
        description,
        seeAlso,
        telephoneNumber,
        userPassword
```

Figure 1: Definition of the X.500 person objectclass

well with heterogeneity and allows highly distributed data management while keeping data conceptually unified [6].

LDAP servers typically fix a schema at start up time. Each *directory entry* (analogous to a tuple in a relational database) has at least one type — called an objectclass — associated with it. An entry can have multiple types, and types can be added to or deleted from the entry at run time. An example of an objectclass is the X.500 person class in Figure 1. Sub-typing is provided, for instance, person inherits from the top class. Note that certain fields must be specified for each person (*i.e.,* sn — surname, cn — common name), while others are optional (*e.g.,* description, telephoneNumber). All attributes are strings that are "weakly typed". For instance, a telephoneNumber is a special kind of string, a tel.[4] One can assign any value to a telephoneNumber — in other words, type checking is not performed for assignments. However, when the system compares two tel values, it compares them in the expected way. For instance, "908-582-5809" and "9085825809" will be considered equal. Similarly, common name (cn) is a case insensitive string (cis), thus, when it is compared with another string, case is ignored. In addition, attributes are multi-valued, for example, a person may have multiple telephoneNumbers.

Directory entries are stored in a tree or forest. Figure 2 is an example of a typical tree, simplified to remove all but one attribute from each entry. Each entry in the tree is identified by a *Distinguished Name* (DN) which is a path from the root of the tree to the entry itself. The DN is produced by concatenating the *Relative Distinguished Name* (RDN) of each entry on the path. The RDN for an entry is set at creation time and consists of an attribute name/value pair — or in more complicated cases, a collection of these pairs connected with various operators. The RDN of an entry must be unique among the children of a particular parent entry. When using Figure 2 in examples, we assume that the attribute/value pairs in the figure (*e.g.,* "o=Lucent" and "cn=John Doe") are the RDNs. Thus, for example, the DN for John Doe is "cn=John Doe, o=Marketing, o=Lucent". Note the leaf-to-root order is the reverse of that for the representation of a UNIX file or a URL.

The only update commands are to create or delete a single leaf node or to modify a single node. There are two kinds of modification commands. The first can modify any fields except those appearing in the RDN. The second, ModifyRDN, cannot change any attribute/value pairs except those appearing in the RDN. Furthermore, while individual update commands are atomic, one cannot group several update commands into a larger atomic unit (*i.e.,* a transaction). For instance, one cannot atomically change a person's name and telephone number if the name is part of the person's RDN but the telephone number is not.

In addition to providing limited update capabilities, LDAP provides query facilities based on the structure of the tree. All queries must specify the DN of a base entry, and search starts at that entry.

[4]The typing of the fields is specified separately in the schema — all attributes with the same name will have the same type.

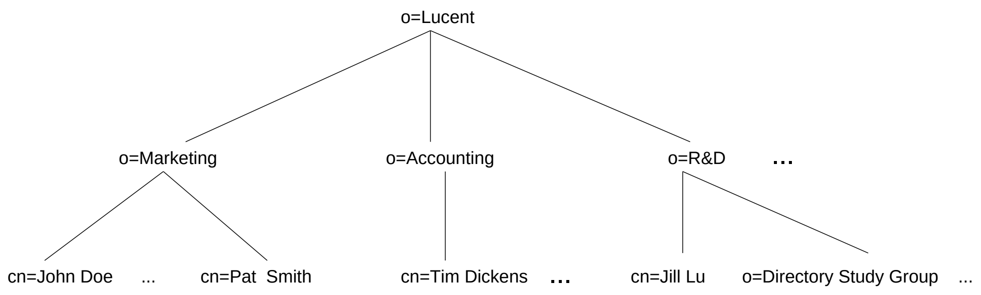

Figure 2: Sample LDAP Tree

There are three query types which differ in their *scope*:

- BASE: queries with a BASE scope examine only the entry specified by the DN. For instance, a BASE scope query against the DN "o=Lucent" would examine only the Lucent record. Examined records are further constrained to pass a *filter*, as we discuss below.
- ONELEVEL: queries with a ONELEVEL scope examine only the entries that are the immediate children of the entry with the specified DN. For instance, a ONELEVEL scope query against "o=Lucent" would examine the entries for Marketing, Accounting, R&D, and any other entries at that level in the tree.
- SUBTREE: queries with a SUBTREE scope examine the entries in the subtree rooted at the specified DN. For example, a SUBTREE query against "o=Lucent" would examine the entries in the tree of Figure 2 (*i.e.*, the entries for Lucent, Marketing, Accounting, R&D, John Doe, etc.).

A user can constrain which of the examined entries the query will return by specifying a *filter*, which is similar to an SQL WHERE clause. For instance, in the SUBTREE query example, the filter

```
(&(objectclass=person)(cn=J*))
```

eliminates the various organizations (*e.g.*, Lucent as a whole, and its divisions) for not being person objects, and eliminates people whose names do not begin with "J" (*e.g.*, Pat Smith). Filters are not as general as a WHERE, they are limited to examining the fields of a single entry. Users can further constrain which attributes are returned using a feature analogous to an SQL SELECT clause.

3 MetaComm: A Telecom Meta-Directory

A great deal of corporate data is buried in network devices such as PBXs, messaging/email platforms, and data networking equipment, where it is hard to access and modify. Typically, the data is only available to the device itself for its internal purposes and it must be administered using either a proprietary interface or a standard protocol against a proprietary schema. In addition, since some of this data is needed in multiple devices as well as in applications like corporate directories and provisioning systems, it is often re-entered manually and partially replicated in different devices in quite different formats. As a result, data is not only hard to access, but there is also a great deal of expense involved in performing updates and the risk of inconsistencies across devices.

The MetaComm system is part of Lucent's DEN initiative. MetaComm integrates data from multiple devices into a meta-directory, allowing user information to be modified through a directory using the LDAP protocol as well as directly through two legacy devices. In order to prevent data inconsistencies, updates to any of the systems must be reflected appropriately in all the other systems. In this scenario,

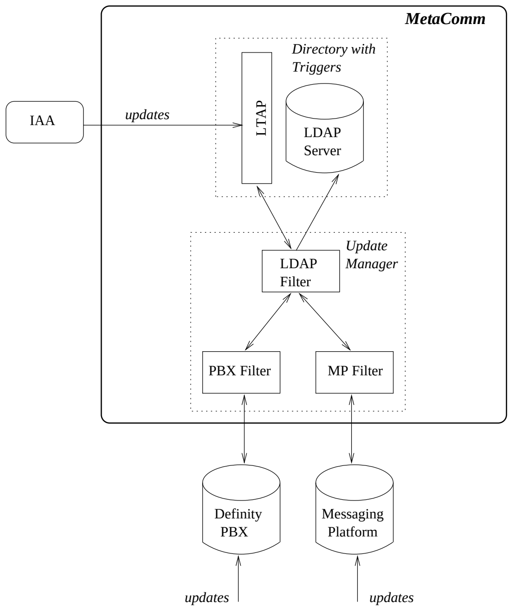

Figure 3: MetaComm Architecture

maintaining consistency poses many challenges, because data integration is performed across several systems with no triggers and with extremely weak typing and transactional support. LTAP is used as infrastructure to solve these problems, providing features like locking and catching LDAP updates that are used by the filters to provide consistency.

In what follows we give a brief overview of the system, how it uses LTAP triggers, and the applications it enables. For a detailed description of MetaComm the reader is referred to [9].

3.1 MetaComm: Architecture and Features

The first prototype of MetaComm integrates user data from two legacy devices: a Definity® PBX and a messaging platform. As shown in Figure 3, our implementation allows user data to be modified in two ways. First, the data can be modified through an LDAP directory which materializes the data from legacy devices. Second, users can continue to modify the telecommunication devices directly through existing proprietary interfaces. Offering multiple paths to modify parameters in crucial telecommunication devices preserves the experience base of administrators of the devices, and it increases the overall reliability and availability of the system since updates can still be made directly to the device even if the directory becomes inaccessible.

Figure 3 shows the various components of MetaComm. The Update Manager (UM) is the central component of the system — it ensures that the data in the devices and in the LDAP server are consistent. Note that consistency is not just a matter of applying the same update to each data repository in a global transaction. Because the repositories lack most basic transaction facilities, MetaComm uses other techniques to ensure that the repositories converge to the same values after some delay (*e.g.*, [28], [8]). These techniques include re-applying updates on certain repositories to ensure the serializability of updates, and recovery from catastrophic communication or storage errors through re-synchronization of the repositories.

Maintaining the consistency of the repositories also requires that the semantics of the data is properly reflected in each repository. A filter or wrapper is associated with each repository. In MetaComm there are three such filters, the PBX filter, the Messaging Platform (MP) filter and the LDAP filter, depicted in Figure 3. Each filter has a protocol converter for communicating with its associated repository and a mapper for translating update commands to/from the schema of the repository. The schema translation and integration of the mapper are realized through *lexpress* [26]. lexpress uses semantic characteristics of the data to simplify data integration. In particular, lexpress uses data dependencies to propagate data wherever it is needed in the global or device schema, and partitioning constraints to translate schema updates correctly and route them to the proper repositories.

Each repository in the system, the legacy devices and the LDAP directory server, must notify the UM when a change occurs. The LTAP module adds active functionality to the LDAP server and notifies the UM of changes to data in the LDAP directory. The main thread of the UM responds to update and synchronization requests by propagating update commands to the appropriate filters. The mapper component in the filter further analyzes the request to ensure that updates are properly forwarded to the associated data repository.

Also shown in Figure 3 is the Integrated Administration Application (IAA), which provides a single point of administration for the telecom devices. It is worth pointing out that any LDAP tool (*e.g.,* an LDAP enabled Web browser) can contact LTAP to administer the telecom devices.

3.2 LTAP: Triggers and Locking

MetaComm uses LTAP as a gateway for its LDAP server. LTAP accepts LDAP messages and returns standard LDAP replies — to the outside world, it looks exactly like an LDAP server. However, it does not perform storage, updates, or queries. LTAP parses each request to determine whether the request is relevant to its triggers. If a request is not relevant it is simply forwarded to the LDAP server, otherwise LTAP performs the required trigger processing actions.

The integrated schema of MetaComm extends the standard X.500 class that describes people with auxiliary classes that represent device specific information. The materialized view of the integrated information is stored in an LDAP server. LTAP triggers are attached to portions of the schema that represent data shared among the devices, ensuring that MetaComm is notified whenever *relevant* data changes.

Besides triggers, MetaComm uses the locking capabilities provided by LTAP (see Section 4.3). As depicted in Figure 3, an update request originating at a device is translated into an LDAP command (to update the global schema of the LDAP server by the appropriate filter) and forwarded to LTAP by the LDAP filter. Because the PBX, MP and the LDAP server lack locking capabilities, the translated updates are sent to LTAP first, so that the corresponding entry is locked before the update is forwarded to the UM. Locking and the possible re-application of the update to the devices ensure proper serialization order.[5]

A more detailed description of LTAP, its functionality and how it is used in MetaComm, is given in Section 4.

3.3 New Applications Enabled by MetaComm

MetaComm allows modification of PBX/messaging settings through any LDAP tool (there are a variety of GUI interfaces to LDAP directories). In our project, we were able to use a commercial LDAP product to easily generate an intuitive Web interface.

Using MetaComm administration, an authorized user/program can easily redirect a telephone extension to a port in another room. Traditionally, this could only be done by a highly trained PBX admin-

[5]It is worth pointing out that the device and the directory may temporarily diverge, but they are guaranteed to converge to a common value based on the order in which they are applied by the UM.

istrator. Consequently, for applications like hoteling,[6] PBX information is not updated when a different person takes possession of a workspace and its telecom resources. Thus, the hoteling employee's calls interact poorly with caller id, leading to callers being identified with uninformative descriptions like "Hotel 1". Once the mapping from extension to port can easily be changed through LDAP, the same tools that assign workspaces can also fix the mapping. In addition, LDAP browsing tools can be used to find out whether a person has any messages, something that previously required either a proprietary tool on a PC, making a phone call, or looking at the light on a phone. We are currently developing such services.

3.4 MetaComm Status

MetaComm was included in a demo at InterOp [25]. The portability advantages of LTAP were demonstrated when preparing the demo since we were able to change directory brands easily.

Lucent has announced a product that will use the MetaComm technology (called Directory Synchronization Technology in the press release) to control Definity® PBXs through an LDAP directory. The technology is currently being transitioned and hardened for commercial use.

4 The LTAP Gateway

LTAP uses the standard *Event-Condition-Action* (*ECA*) model [7]. Triggers are specified to fire when a directory entry is *affected*, that is, read or written by an operation. Users specify an Event to monitor, a Condition to check, and an Action to perform if the event occurs and the condition holds. Users specify these by instantiating either a C++ or Java Trigger object and then registering it with the system. Each specified event monitors the progress of an LDAP operation, for example an event can be set to trigger before an LDAP modify or after an LDAP delete. The condition must be satisfied by the entry affected by the operation. For example, the condition might specify that the affected entry must be of a particular type and must involve the modification of certain attributes.

An Action is implemented by sending a message to a *trigger action server* (TAS) with information about the trigger and the operation that fired the trigger so that the TAS can handle the action. Users must provide the TASs for their applications by adding three functions to an LTAP-provided program shell (see Section 4.4 for more details).

Rather than go into detail on the syntax, we give an example to illustrate the basic trigger features. Full details can be found in the LTAP user manual [1]. The example is based on the triggers used by MetaComm (described in Section 3). LTAP notifies MetaComm whenever a person entry is added, modified, or deleted. We illustrate the creation and execution of a trigger for modification alerts — similar features are required for adds and deletes.

4.1 Trigger Creation

Trigger creation involves defining a trigger action and its trigger action server (TAS), and defining a trigger. A trigger action is specified as follows:

Action actionPersonMod(*ActionMachine*, *ActionPort*);

where *ActionMachine* is a string specifying a machine to contact and *ActionPort* is an integer specifying a port number. These two parameters specify a TAS. Having specified the action, MetaComm next constructs the Trigger in Figure 4. The parameters have the following meanings:

- The first parameter, *e.g.*, "triggerPersonMod", specifies the name of the trigger. It is the key for the trigger and allows looking up the trigger for modification/deletion.

[6] In hoteling, employees who frequently are outside the office share a pool of workspaces. They reserve a workspace when they need to come to the office.

```
Trigger triggerPersonMod(
        ”triggerPersonMod”,  /* (1) name of trigger */
        OpModify,            /* (2) Modify causes trigger to fire */
        Before,              /* (3) Trigger fires and returns before Modify executed*/
        actionPersonMod,     /* (4) Execute action specified by actionPersonMod*/
        ScopeType,           /* (5) Trigger based on type of the entry*/
        ”person”);           /* (6) Trigger placed on person entries */
```

Figure 4: Trigger Definition to Monitor Reads of Jill Lu’s Directory Entry

- The second parameter, *e.g.,* OpModify, specifies the LDAP operation, in this case a modify, causing the trigger to fire. The second and third parameters are combined to define the event.
- The third parameter, *e.g.,* Before, specifies when the trigger firing will take place in relationship to the LDAP operation. In this example, the action will be executed Before the modify is attempted. This parameter could also specify that the trigger should fire After the LDAP operation has successfully completed or after the operation has Failed.
- The fourth parameter, *e.g.,* actionPersonMod, specifies the action to execute after the event has occurred if the condition holds.
- The fifth parameter, *e.g.,* ScopeType, specifies an LDAP search scope. The search scope can be a portion of the naming tree, specified by ScopeBase, ScopeOneLevel, or ScopeSubtree, or objects of a particular type, specified by ScopeType. The trigger event is monitored at all entries that would be returned by a LDAP search with scope specified by the fifth parameter and with the DN or objectclass specified by the sixth.
- The sixth parameter, *e.g.,* “person”, is either a DN — if the fifth parameter is ScopeBase, ScopeOneLevel, or ScopeSubtree — or an objectclass — if the fifth parameter is ScopeType. Given our combination of fifth and sixth parameter, this trigger will fire before any modifications of person entries.

The example in Figure 4 illustrates only the mandatory parts of the trigger, the event and the action. An LTAP condition is optional and contains two parts: a list of fields to monitor, triggerMonitoredFields, and an LDAP filter. If the trigger specifies triggerMonitoredFields, the trigger action will not be executed unless at least one field in the monitored list is *involved* in the LDAP command that caused the firing. If triggerMonitoredFields is not explicitly set, this is equivalent to specifying a list containing all possible fields that could be modified. MetaComm constructs the triggers to monitor fields mentioned in its mapping filters. Filters are used to convert between the LDAP representation of these fields and that of the devices controlled through MetaComm — see Section 3 for more details.

In addition to preventing triggers from firing when uninteresting fields are modified, triggerMonitoredFields serves a security function — LTAP only fires a trigger if the trigger writer is authorized to read at least one of the monitored fields that are involved in the operation (see [22] for details).

The second part of the LTAP condition, the LDAP filter, constrains the trigger to fire only when the object satisfies the filter, for example when the object has a particular value for an attribute. The trigger fires whenever a search requested by the trigger writer and rooted at the specified DN, scope, and filter would return the entry. To continue the example, suppose MetaComm only cares about modifications to a person’s entry if the telephoneNumber field is set. Then, MetaComm would add the following line:

```
triggerPersonMod.triggerFilter = “(telephoneNumber=*)”;
```

This could be used as an optimization if system rules specify that a person created with no telecom access will not get PBX/messaging services added later.

TASs may specify triggerFieldsOfInterest, a list of fields that LTAP needs to include in a trigger notification message. LTAP also provides TASs the ability to access the values of fields pre-update (NeedsBefore), post-update (NeedsAfter) or both (NeedsBoth), or specify neither (NeedsNeither) when no values are needed.

The TAS for MetaComm does not specify a value for triggerFieldsOfInterest since it is interested in all of them, and all fields is the default. However, MetaComm needs information about how the fields are modified and what their pre-update values are in order to properly update the controlled devices. As it needs both the pre- and post-update values, it adds the following line:

```
triggerPersonMod.triggerNeeds = NeedsBoth;
```

Having constructed the Trigger, MetaComm installs the trigger into the LTAP gateway by including the following line:

```
ldapConnection.createTrigger(triggerPersonMod);
```

where ldapConnection is a handle created when MetaComm's Update Manager connected to LTAP.

4.2 Trigger Processing

The following steps (illustrated Figure 5) are performed in basic trigger processing. We do not consider time outs in this discussion. The user manual describes the trigger facilities for failure cases [1].

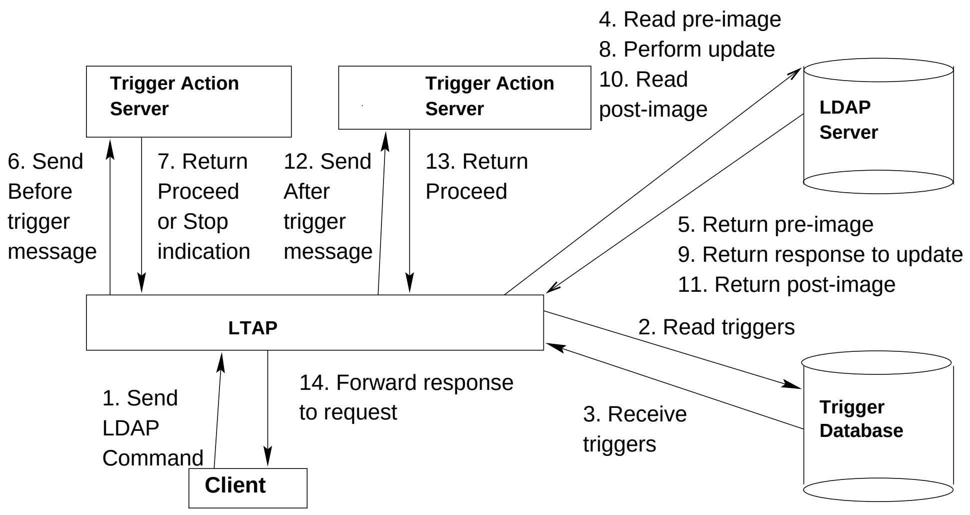

Figure 5: Interactions during Trigger Processing

1 LTAP receives a LDAP command C from a user U for entry E.

2/3 LTAP reads trigger information from the trigger database for E. Note that one instantiation of LTAP caches the trigger database in its entirety in main memory within the LTAP process.

4/5 LTAP reads pre-update information for E (if required by some trigger).

6/7 LTAP processes each matching Before trigger, passing relevant information to the corresponding TAS — in practice, there may be several TASs contacted, but for simplicity only one appears in Figure 5. (A similar comment applies to the possible existence of several TASs for After triggers.)

Note that since the Before trigger creator is required to have super-user status, no run-time authorization is needed. The reasons for this requirement are detailed in our section on security in our previous work [22].

TASs for Before triggers return one of three values to LTAP after completing:

- Proceed: instructs LTAP to execute the command C received in step 1.
- StopFailure: instructs LTAP not to execute C and to return failure to U.
- StopSuccess: instructs LTAP not to execute C but to return success to U. An example use for StopSuccess is to allow updating through views. The original command is not executable against the underlying repository, so sending it there would fail. However, the TAS can rewrite the update and issue the rewritten update itself. If the rewritten update succeeds, the TAS returns StopSuccess. The use of StopSuccess in MetaComm is discussed in Section 5.

14 If any TAS returns StopFailure or StopSuccess in step 7, then return failure to U if a least one StopFailure was received, else return success.

8/9 Execute the LDAP command received in step 1.

10/11 For each After trigger, read the after update information for E as the trigger creator and use this information to authorize the trigger.

12/13 Execute any authorized After triggers.

14 Return status of LDAP command received in step 9.

It should be noted that LDAP command failure will cause the following change — rather than proceeding from step 9 to step 10, LTAP executes any Failed triggers that are authorized to execute and returns the result that was received during step 9 to U.

4.3 Locking

Write locking is required to protect the integrity of the pre- and post-images. Updates to a directory entry while trigger processing on that entry is in progress must be prevented. Thus, when a LDAP update command arrives at LTAP, *i.e.,* step 1 in Figure 5, a write lock is acquired. The lock is held until all the After triggers have been processed, *i.e.,* step 13 in Figure 5. Read requests ignore locks. A write to a single entry is atomic, so each read will see internally consistent entries, *i.e.,* LTAP gives the standard LDAP guarantees for reads. MetaComm, on the other hand, depends on LTAP's locking facilities to maintain consistency between the directory and telecom devices.

Certain users cannot afford to have any operations blocked due to locking. To support such users, LTAP allows locking to be turned off in a configuration file.

4.4 Trigger Action Server Support

In the version of LTAP used in MetaComm, a trigger action contacts a specified port of a TAS. The TAS then executes some code in response to this contact. A TAS returns Proceed, StopSuccess, or StopFailure in response to a Before trigger request.[7] For After/Failed triggers, a TAS simply returns a Proceed as a "done" indication. The LTAP release provides a simple TAS program that can easily be

[7]MetaComm's TAS, the Update Manager, only uses Before triggers, and it never returns Proceed.

tailored by defining three functions of the TriggerFields class as described below. The user then compiles the TAS and starts it on the machine specified in the trigger.

When the TAS is contacted, a number of items are included in the message. For instance, authorized fields that appeared in the triggerFieldsOfInterest of the Trigger associated with this invocation of the TAS are sent if requested. Each field sent is packaged in an AttributeValue. For each field, an AttributeValue stores its name, attributeName; what happened to it, changeType; and its pre-update (post-update) value, beforeValue (afterValue). Note that the pre- update (post-update) value will only be sent if NeedsBefore (NeedsAfter) or NeedsBoth was specified by the trigger creator. Also, beforeValue and afterValue are arrays of values because fields may be multi-valued in LDAP. The following is a more detailed description of what values are sent if triggerFieldsOfInterest was specified:

- If the affecting action was an add, then the post-update value will be sent. (It is illegal to specify an OpAdd trigger with NeedsBefore or NeedsBoth.)

- If the affecting action was a delete, then the pre-update value will be sent. (It is illegal to specify an OpDelete trigger with NeedsAfter or NeedsBoth.)

- If the affecting action was a modify, then its action on any given attribute could be to add a previously non-existent attribute, to delete an existing attribute, or to modify an attribute's contents. In the attribute delete (add) case, only the pre-update (post-update) value will be sent — and that only if NeedsBefore (NeedsAfter) or NeedsBoth was specified. In the attribute modify case, both will be sent if NeedsBoth was specified while the pre-update (post-update) value will be sent if NeedsBefore (NeedsAfter) was specified.

Note that it is illegal to specify a trigger with a non-empty triggerFieldsOfInterest and NeedsNeither.

In addition to the pre- and post-update values in the affected entry, the following fields are sent in the message to the TAS:

- actionDN is the Distinguished Name of the affected directory entry that caused the trigger to fire.

- triggerName, triggerWhen, triggerOp, triggerArg, triggerScope, and triggerTypeOrDN contain the values of the members with the same names specified when creating the trigger that just fired. They correspond to the six parameters to the Trigger constructor.

- bindAuthorization is a structure containing the login name and password of the user whose action caused the trigger to fire. This information is only filled in for Before triggers (which can only be written by super-users), and only from LTAP servers that have been configured at start up time to deliver this information. This information is provided in those cases to allow query/update rewrite by TASs while still preserving the security authorizations of the requesters — the TAS pretends to be the requester while executing the rewritten command. Note that sending this information can be turned off in the LTAP configuration file.

The TAS instantiator's task with regard to processing trigger action service requests is to write three member functions of TriggerFields: triggerBeforeAction, triggerFailAction, and triggerAfterAction. The provided server code parses each incoming service request. If the parsed request has a triggerWhen value of Before, the server calls the triggerBeforeAction member function and returns the ActionStatus returned by that call to LTAP. If triggerWhen is After (Failed), the server calls triggerAfterAction (triggerFailAction) and returns Proceed to LTAP after the call completes. Since the value is ignored by LTAP for After and Failed triggers, the server code picks Proceed arbitrarily to send back for the purpose of synchronization with the LTAP gateway.

A TAS that handles the trigger triggerPersonMod of Figure 4 can be constructed by defining the triggerBeforeAction as in Figure 6.

```
ActionStatus TriggerFields::triggerBeforeAction() {
   Reformat Before/After image into lexpress' internal format;
  Use reformatted images to create commands to update PBX, messaging platform, and LDAP directory;
   If all goes well during updating, return StopSuccess;
   Otherwise, log error and return StopFailure;
}
```

Figure 6: TAS Code for the Before trigger, triggerPersonMod (part of the MetaComm's Update Manager)

5 Experiences

In this section we discuss the rationale for various past and future extensions to LTAP based on our experiences in building MetaComm. We describe tricks we developed for exploiting LTAP features, and useful interactions between LTAP and lexpress [26]. We conclude the section with a discussion on how to use the LTAP gateway model to provide advanced database functionality to directories.

5.1 Extensions to LTAP

Persistent Connections and Synchronization One useful feature that MetaComm provides is synchronization. Synchronization is useful for initially loading data from a device into the global integrated directory, as well as ensuring consistency across devices after catastrophic failures occur. In order to provide the synchronization facility, MetaComm must guarantee that after a synchronization request is processed, the LDAP server, the device being synchronized, and other devices that share data with the LDAP server or device are consistent. Even though synchronization requests can be thought of a sequence of individual updates, the set of updates should be applied in isolation, *i.e.,* other updates should not be allowed concurrently. This required modifications to LTAP:

- In its original implementation, LTAP only allowed a single update per connection from LTAP to a TAS (*e.g.,* UM), but to differentiate synchronization requests from individual updates, persistent connections were added which allow a sequence of updates.
- In order to guarantee that synchronization requests are executed in isolation, all updates must be disallowed while a synchronization request is being processed. To support this, a new *quiesce* facility was added to LTAP.

Monitored Fields When LTAP was originally designed, only the triggerMonitoredFields construct was provided — this construct combined the functionality of triggerMonitoredFields and triggerFieldsOfInterest in the current system. However, one of our users insisted that they be separated to reduce the amount of work needed to process a trigger. For the application in questions, a notification that a field had changed was sufficient — for instance, if a person's name and telephone number have changed, the application only needs to know the name of the person but nothing more. Thus, the distinction between monitored fields and fields that must be shipped to a TAS was made.

This distinction was important for MetaComm as well, but for a different reason. MetaComm only needs to know if name and telecomm-related fields are about to change. However, when it must handle the change by rewriting the update, it cannot lose any parts of the update. For instance, if the IAA asks to modify the sn, telephoneNumber, and description field of a person, the UM will be alerted because of the first two fields. When it rewrites the LDAP command, it will also need to know the new value for description, so that the semantics of the update are preserved.[8] Given that new auxiliary classes can be added to an existing object at any time, it was not practical for MetaComm to explicitly list all the fields that it would like to have shipped, so it asks that all fields belonging to a added/deleted/modified entry be shipped.

TAS return values While developing MetaComm, we discovered the need to support more return values from a trigger action server (*e.g.*, the Update Manager). Besides Proceed and StopFailure, we also needed StopSuccess. The two original values are adequate if the server only needs to approve or reject the operation. However, the MetaComm implementation could be simplified if it updated both the devices and the directory, so that it could deal with failure of LDAP commands more easily. Otherwise, if it had to deal with a Failed trigger, it would have had to keep around state from the device updates until it was certain that the LDAP command had gone through — leading to garbage collection problems.

5.2 Exploiting LTAP Features

Handling Anomalies and the Need to Rewrite Updates The decision regarding the TAS return values led to unanticipated benefits. First, in certain cases, the original LDAP update that triggered the event can only be approved by MetaComm after the underlying devices are changed. This is because it cannot be determined whether the devices will accept the update without actually attempting the update. Some operations that are syntactically and semantically acceptable will be rejected because of arriving when crucial resources are unavailable at the device. When these operations fail, the LDAP update command that triggered them must be rejected (StopFailure is returned). Second, MetaComm was enabled to rewrite LDAP updates and apply them itself which is very useful when the original LDAP command needs to be rewritten. For instance, if Jill Lu's sn is changed to "Lu-Smith", then her cn should be changed to "Jill Lu-Smith". LDAP does not enforce such dependencies, but a simple operation rewrite in the UM by lexpress can. Rewrites are also useful because certain operations on the devices return information that needs to be reflected in the directory. For instance, when an entry for a new mailbox is created on the messaging platform (MP), an id is generated by MP. Since this id is needed by the LDAP directory, the MP passes the id to the UM which rewrites the original LDAP command to set the corresponding field in the LDAP directory. The UM then sends the rewritten LDAP command to the LDAP directory directly. If it succeeds, the UM returns StopSuccess. Otherwise, it returns StopFailure.

Reduced Numbers of Triggers When originally designing LTAP, we anticipated that users would set triggers on the type highest in the lattice whose modification was of interest, but that only fields belonging to that type would be sent to the TAS. This turned out not to be practical, since some application need to differentiate fields to be monitored and fields to be included in a trigger notification (see Section 5.1). While developing MetaComm, we noticed that we could take advantage of the fact that the only updates we were interested were updates to person entries. The update might be to fields belonging to an auxiliary class that was hung off a person, but the objectclass of the modified entry would include person as one of its types. By placing the trigger high in the class hierarchy and taking advantage of LTAP's willingness to allow the triggers to specify any field whatsoever (including those of person subclasses) as being of interest and to be monitored, MetaComm was able to get by using only three triggers — one for modify, add, and delete of a person. A single *any-update* trigger would handle all the alerting needs for the part of the system described in this paper. The extensions to MetaComm that are on-going will probably require only one trigger of the *any-update* sort for each telecomm-related class.

[8] In addition to the fields mentioned explicitly, it will need other fields like cn that will implicitly be changed by the change to sn. See Section 5.2.

The form of all of MetaComm's triggers was extremely similar to that of triggerPersonMod in Section 4. All the triggers are interested in monitoring the same set of fields and in having all fields sent to the UM. The differences were quite uniform. The second parameter to the Trigger constructor is different for each, specifying either add, delete, or modify. Also, the triggerNeeds fields specifies NeedsAfter (NeedsBefore, NeedsBoth) for the trigger for adds (deletes, modifies). As a result of this experience, we realized the need to add an *any-update* trigger type in a future version of LTAP.

Using Pre-images We initially decided to receive pre-images, as well as post-images, in LTAP notifications, so we would always have access to the keys for updating the devices. The keys for updating the devices are often in other attributes than the distinguished name for the LDAP object. When the key changes, both the old and the new value of the key are needed to update the target device. For instance, a key modify may require moving a user from one PBX to another. In practice, the pre-image was even more useful. lexpress uses pre-images during constraint processing to determine if the object being modified resides on the target device. It compares pre-images and post-images to determine what changes have occurred, and only produces updates for target attributes that must change. In addition, lexpress must at times use the values of source attributes that have not changed in creating new values for target attributes. lexpress creates a composite target attribute from multiple source attributes. The composite attribute changes when at least one of its contributing source attributes change. When creating a new composite attribute value, lexpress requires values for all of the source attributes, whether they have changed or not, so these unchanged fields must be sent.

5.3 Offering other Extended LDAP Functionality through Gateways

The use of a gateway to add needed but unstandardized functionality to an LDAP directory (or any repository that speaks a standard protocol) was validated by the experience of LTAP in MetaComm. Such gateways offer portability advantages. For instance, while preparing the demo for InterOp [25], we were able to change directory brands easily — had we relied on proprietary trigger facilities, likely the port would have been much harder. Another advantage of the gateway approach is that it only makes collections of applications that need a given piece of functionality pay for the existence of the functionality — simpler application suites can go directly to an un-encapsulated directory.

The same gateway technique could also be used for replication or materialized views. Stronger security techniques could also be built into a gateway than are present in the directory, provided one can ensure that there is no direct path to the directory except through the gateway and a handful of trusted servers. It would be attractive to use a similar idea to provide sagas [11]. For example, a simple saga mechanism could be built by creating special messages to the gateway to indicate transaction begin and end and to treat each intervening request as a step in the saga, and by keeping information about what needs to be undone if any operation returns StopFailure. Support for more general transaction models would be useful, but this may not be possible when the underlying repositories do not have two-phase commit support.

6 Related Work

Details of the LTAP gateway are given in [22]. In this paper, we focus on the use of LTAP in the MetaComm system — its benefits and extensions/changes that were needed. The MetaComm system and its relation to previous work (*e.g.,* data integration [21, 10], sagas [11], data warehouses and views [2, 4, 27, 13]) is described in [9]. Unlike [9], this paper focuses on our experiences in using LTAP to build MetaComm, the features of LTAP that we exploited in novel ways, and our rationale for various changes and extensions to LTAP.

Netscape's directory plug-ins are the work most closely related to LTAP as their plug-ins can be associated with directory operations [24]. Their plug-ins and our triggers have similar structures — both associate an action with the time immediately before or after the performance of a LDAP command. Their plug-ins run as function calls in the directory server itself. This has performance advantages, but causes reliability, portability, and availability problems than LTAP does not have [22]. Both Microsoft and Cisco plan to add triggering capabilities to Active Directory for DEN, but the capabilities do not exist yet. Novell does not appear to have triggering capabilities associated with their LDAP products. Given that MetaComm needed to be supported across all LDAP servers, LTAP was the only viable choice.

Triggers are a well-known technique for databases. A user specifies an event (or pattern of events) of interest that the system is to monitor, a condition to check (which may always be true), and an action to take. When the event is detected and the condition holds true, the system performs the action. The novelty of our LTAP work is our support for Failed triggers, necessary to cope with the weak transactions of LDAP, and the use of a proxy to provide trigger support. Previous techniques for adding triggers detected events either by adding event detection into the core database engine, *e.g.*, [3, 32, 23, 29], and/or by using pre-processors on database code to signal events, *e.g.*, [23, 5].

C^2*offein* is a system to add ECA triggers in a CORBA environment [19]. They get their events from wrappers around data sources. Rather than write special data wrappers for each data source, LTAP takes advantage of the standard vocabulary of LDAP and only requires one "wrapper" to support arbitrary LDAP data sources.

Two recent proposals for LDAP triggers [12, 30] suggested using persistent queries where the query stays active finding adds, deletes, and modifies to the LDAP directory that match the search condition. A persistent query, thus, corresponds to a trigger. This mechanism has scalability problems that LTAP does not have — it must maintain one open connection per persistent query. In LTAP, only triggers that are currently being processed require an open connection. Persistent search, once standardized, could be used to build a tool providing general After triggers in a scalable way — in other words, a tool providing a subset of LTAP functionality. Such a tool would eliminate the scalability concerns, since only a single connection to the tool would be required. However, Before and Failed triggers could not be supported by such a tool, since the tool would only be alerted after a successful modification had already taken place. Furthermore, such a tool could not provide pre-update values unless it keeps its own copy of the directory. These limitations would make it harder to write an application like MetaComm against the tool than against LTAP.

Other work to extend LDAP to take advantage of database technology aims at adding a more expressive, non-procedural query language to LDAP [17]. We share their aim of moving useful techniques from databases into directories, but deal with triggers rather than queries. The architecture of the LTAP library version is very similar to that used to add more flexible list management to LDAP [18]. In their work, queries to LDAP go through a "list location and expansion" client library before being sent to the LDAP directory. In the LTAP library approach, triggers are provided by having LDAP queries and updates go through a library that does trigger processing as well as passing commands on to the LDAP directory. Also, while the LTAP gateway interface is quite different from their approach, their code could be converted to use a gateway approach as well.

7 Conclusions and Future Work

Triggers and other basic database functionalities are needed by LDAP servers to support applications such as Directory Enabled Networking tasks. The lack of standard trigger facilities is a significant barrier to progress as users begin using LDAP for tasks like provisioning network services, allocating resources, reporting, managing end-to-end security, and offering mobile users customized features that follow them. Current LDAP servers either lack trigger facilities or provide proprietary triggers with significant limitations. In this paper, we described the use of LTAP, a gateway which provides a portable way to add full-fledged triggers to LDAP servers, in the telecom integration project MetaComm [9]. LTAP's alerting and locking facilities provide valuable glue for combining the project's directory, a Definity® PBX, and a messaging platform into an integrated package. The resulting platform is able to support powerful, directory-enabled telecom applications such as the ones described in Section 3.3.

We also described our experiences gained in using LTAP in MetaComm, both where its functionality could be used in unanticipated ways to solve problems and where it needed to be extended. These experiences are key lessons learned in this paper. They will be valuable to those building similar systems. Based on our experiences, gateway technology appears to be a good choice for providing replication, security, and saga facilities to directories in a portable way.

We would like to incorporate LTAP into other networking/telecommunications projects to find new enhancements to the system. We are currently extending LTAP's functionality and integrating it with other telecom products. MetaComm technology is currently being transitioned and hardened for commercial use.

ACKNOWLEDGMENTS: Thanks to Arun Netravali for suggesting we explore the DEN area. Thanks to Qian Ye who helped build MetaComm and test LTAP. Thanks for valuable comments from Mike Holder, Gavin Michael, Shamim Naqvi, Hector Urroz and G. A. Venkatesh. In addition, we would like to thank our colleagues on the MetaComm project who have not already been mentioned: Lalit Garg, Julian Orbach, and Luke Tucker. Andy Liao provided us with a Java LDAP parser which allowed us to replace the C LDAP parser from the University of Michigan that we had been using.

References

[1] R. Arlein, N. Gehani, and D. Lieuwen. LTAP trigger gateway for LDAP directories. `http://ltap.bell-labs.com/LTAPTM.doc`.

[2] J.A. Blakeley, P.-A. Larson, and F.W. Tompa. Efficiently updating materialized views. In *Proc. SIGMOD*, pages 61–71, 1986.

[3] A. P. Buchmann, J. Zimmermann, J. A. Blakeley, and D. L. Wells. Building an integrated active OODBMS: Requirements, architecture, and design decisions. In *Proc. Data Eng.*, pages 117–128, March 1995.

[4] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *Proc. VLDB*, pages 577–589, 1991.

[5] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proc. VLDB*, pages 606–617, August 1994.

[6] S. Cluet, Olga Kapitskaia, and D. Srivastava. Using LDAP directory caches. In *Proc. PODS*, 1999.

[7] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ladin, D. McCarthy, A. Rosenthal, and S. Sarin. The HiPAC project: Combining active databases and timing constraints. *SIGMOD Record*, 17(1):51–70, March 1988.

[8] A. Demers, D. Greene, A. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. ACM Symp. on the Principles of Distr. Computing*, pages 1–12, August 1987.

[9] J. Freire, D. Lieuwen, J. Ordille, L. Garg, M. Holder, H. Urroz, G. Michael, J. Orbach, L. Tucker, Q. Ye, and R. Arlein. MetaComm: A meta-directory for telecommunications. In *Submitted for publication*, 1999.

[10] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[11] H. Garcia-Molina and K. Salem. Sagas. In *Proc. SIGMOD*, 1987.

[12] G. Good, T. Howes, and R. Weltman. Persistent search: A simple LDAP change notification mechanism. `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-psearch-01.txt`.

[13] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *Proc. SIGMOD*, pages 328–339, 1995.

[14] Directory Enabled Networking Ad Hoc Working Group. `http://murchiso.com/den/`.

[15] T. Howes and M. Smith. *LDAP: Programming Directory-enabled Applications with Lightweight Directory Access Protocol*. Macmillan Technical Publishing, 1997.

[16] Innosoft. Innosoft's LDAP world implementation survey. `http://www.critical-angle.com/dir/lisurvey.html`.

[17] H. Jagadish, L. Lakshmanan, T. Milo, D. Srivastava, and D. Vista. Querying network directories. In *Proc. SIGMOD*, 1999.

[18] H. V. Jagadish, M. Jones, D. Srivastava, and D. Vista. Flexible list management in a directory. In *Proc. CIKM*, 1998.

[19] A. Koschel and R. Kramer. Configurable event triggered services for CORBA-based systems. In *Proc. International Enterprise Distributed Object Computing Workshop*, November 1998.

[20] `http://www.ietf.org/html.charters/ldapext-charter.html`.

[21] A. Y. Levy, A. Rajaraman, and J.J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. VLDB*, pages 251–262, 1996.

[22] D. Lieuwen, R. Arlein, and N. Gehani. The LTAP trigger gateway for LDAP directories. In *Submitted for publication*, 1999.

[23] D. Lieuwen, N. Gehani, and R. Arlein. The Ode active database: Trigger semantics and implementation. In *Proc. Data Eng.*, pages 412–420, February–March 1996.

[24] `http://developer.netscape.com/docs/manuals/directory/plugin/index.htm`.

[25] Lucent. `http://www.lucent.com/press/0599/990503.nsc.html`.

[26] J. Ordille. The lexpress system. Technical report, Bell Laboratories - Lucent Technologies, 1999.

[27] X. Qian and G. Wiederhold. Incremental recomputation of active relational expressions. *IEEE Trans. on Knowledge and Data Eng.*, 3(3):337–341, September 1991.

[28] L. Seligman and L. Kerschberg. A mediator for approximate consistency: Supporting "good enough" materialized views. *Journal of Intelligent Inf. Sys.*, 8:203–225, 1997.

[29] M. Stonebraker, E. Hanson, and C. Hong. The design of the Postgres rules system. In *Proc. Data Eng.*, pages 365–374, 1987.

[30] M. Wahl. LDAPv3 triggered search control. `ftp://ftp.isi.edu/internet-drafts/draft-ietf-ldapext-trigger-01.txt`.

[31] M. Wahl, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3), December 1997. `http://www3.innosoft.com/ldapworld/rfc2251.txt`.

[32] J. Widom, R. J. Cochrane, , and B. Lindsay. Implementing set-oriented production rules as an extension to Starburst. In *Proc. VLDB*, pages 275–285, September 1991.