Supporting Disconnectedness - Transparent Information Delivery for Mobile and Invisible Computing

Peter Sutton, Rhys Arkins,
School of Computer Science and Electrical Engineering
The University of Queensland
Brisbane QLD 4072 Australia
p.sutton@csee.uq.edu.au, rarkins@dstc.edu.au,

Bill Segall
CRC for Enterprise Distributed Systems Technology (DSTC)
UQ Brisbane QLD 4072 Australia
bill@segall.net

Abstract

As computing devices become ubiquitous and increasingly mobile, it is becoming apparent that the directed peer-to-peer communication model has shortcomings for many forms of distributed interprocess communication. Undirected communication, including content-based messaging, is becoming increasingly common. This paper examines the issues involved in supporting content-based messaging to both mobile devices and users using a combination of connected and mobile (possibly disconnected) devices. These issues include persistence, multi-client shared subscriptions, non-destructive notification receipt, and notification expiry. The discussion is placed in the context of the development of a proxy-server to provide disconnectedness support for the Elvin content-based messaging service.

1. Introduction

Interprocess communication is commonly achieved using directed links between tightly coupled senders and receivers. In this case, the destination of the message must be known at the time of sending - which is difficult when the destination is unknown, is changing, or the number of recipients varies. There is a growing trend towards using loosely-coupled autonomous objects for building large-scale distributed systems to confront these shortcomings of directed communication [1].

A common approach to achieving decoupling of interacting objects is using an event-notification or publish-subscribe design style. In an event-based system, object interactions are modelled as events, which are transmitted in the form of notifications. In event notification services, such as Elvin [2], TIB/Rendezvous [3], Siena [4], Gryphon [11], and OpenQueue [5], consumers can specify which events they are interested in by submitting subscriptions to an event router. Some services, such as OpenQueue, support subject- or topic-based routing, whilst other services, such as Elvin, provide full content-based routing.

The advantage of using an undirected event service instead of other traditional communication mechanisms is that this method decreases the coupling of objects in a distributed environment and removes the need for many static dependencies. While many of today's distributed computing interactions already lend themselves to the undirected model of communication, the new wave of wireless, handheld, mobile, ubiquitous and invisible devices will generate an even greater need for decoupling of distributed objects; indeed, invisible computing's fundamental properties are particularly incompatible with centralised architectures and tightly coupled communication techniques.

With this new wave of distributed systems comes a range of challenges, including: dealing with mobile devices which are often disconnected from the network; delivering information to *users* who have a range of devices on which they may receive information; and scaling interactions for large numbers of devices.

This paper examines the issues involved in supporting undirected communication in distributed systems which include mobile (possibly disconnected) devices and describes how these issues have been resolved in the context of the Elvin content-based messaging service.

The remainder of this paper is organised as follows. Section 2 discusses undirected information delivery for mobile devices and in particular describes content-based messaging and its appropriateness for supporting disconnected devices. Section 3 describes the Elvin content-based messaging service on which this work is based. Section 4 describes issues involved in delivering information in the face of disconnectedness and multiple devices. Section 5 describes the implementation of disconnectedness multi-device support for the Elvin content-based messaging service and discusses the lessons learned. Conclusions are drawn in Section 6.

To be published in Proceedings of 2001 IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01), May 2001.

2. Undirected Information Delivery for Mobile Devices

The "sometimes disconnected" nature of mobile computing devices means that undirected information delivery is necessary, i.e. producers and consumers need to be decoupled so there is no requirement for a direct network connection (e.g. TCP) between them. This section discusses this issue in more detail and also describes content-based messaging and its appropriateness for mobile disconnected devices. Related work in the area is also considered.

2.1. Hardware Constraints Encourage Undirected Communication

The hardware constraints imposed by mobile devices such as personal digital assistants (PDAs) mean that the amount of data transferred and processed should be kept at a minimum. This is the result of limited availability of bandwidth and memory, as well as other factors such as limited power supply and computational ability. Undirected communication is advantageous over directed communication in that filtering can be achieved at an "event server" or router, and intelligent subscriptions can minimise the amount of data transferred.

Undirected communication is well suited to PDAs as these devices are commonly used for collection of data, yet due to the nature of their frequent disconnectedness, the location and number of recipients may be continually changing. This characteristic communication paradigm is an example of where decoupling the producer and consumer of data provides greater flexibility, as the producer can send its data without being concerned with the location or availability of its recipients.

2.2. Decoupled Communication

Decoupled communication can take many forms and names, including publish-subscribe, store-and-forward, event notification, message queuing, subject-based routing and content-based messaging. Sometimes these terms are used interchangeably and there is indeed overlap between many of the concepts. In this work we will classify decoupled communication as one of two types: message queuing or subject/content based messaging.

2.2.1. *Message Queuing.* Message queuing is essentially a form of directed communication that doesn't require a connection. Messages are usually directed to a particular destination and are queued on both the disconnected device and a server, so that when the device is reconnected to the network, messages are transferred in both directions.

2.2.2. *Subject/Content Based Messaging.* Subject or content based messaging[1] decouples the producers from the consumers. Producers *publish* messages but are unaware of the destination. Consumers *subscribe* to particular groups of messages and one-to-many messaging is possible (i.e. a message produced by a source may be passed to many destinations).

There are significant differences between subject and content based messaging. Subject-based addressing allows consumers to subscribe only on the basis of message header information (or *meta-data)*, usually the *subject* or *channel* or some combination of meta-data items[2]. This form of communication lends itself to multicast propagation with multicast groups corresponding to particular channels. A limited amount of coupling remains between producers and consumers - they are coupled on the basis of the channel name [6].

Content-based addressing, on the other hand, allows the consumer to subscribe based on any aspect of the content of a message - not just the subject or other header. This provides greater flexibility to users and application developers as there is less coupling between producers and consumers, for example, consumers aren't forced to take a complete channel feed, they can selectively filter appropriate messages *at the router* (or server).

Content-based schemes have neither restrictions on the visibility of messages nor restrictions on what elements of a message can be used for selection. The major distinction of content-based routers is that message routing is determined by the consumers of the information rather than the producers [6].

2.3. Disconnected Devices

Current mobile devices such as PDAs are characterised by frequent disconnectedness which significantly affects how distributed network communication can occur. This disconnectedness can be due to a number of reasons - out of range, device turned off, or application swapped away if an operating system is single tasking. For example, in the PalmOS operating system, a network application can be connected only when the user has that application open as the focus. It is not possible to set these applications to run in the background - every time they are opened the network

1. This is sometimes called subject or content-based *routing.* This work will consider content-based routing to be at a lower level than content-based messaging, e.g. routing algorithms or network techniques (e.g. mapping to multicast groups) to support such messaging. Content-based messaging would normally be based on content-based routing, but need not be.
2. The subject or channel is also sometimes known as the *topic* or *group*.

connections need to be completely re-established.

Most work in dealing with the disconnectedness of computing devices (usually mobile devices) revolves around issues of *data replication* and *synchronization*. Data replication involves copying data (e.g. a database or part thereof) from some central repository to a mobile device so that the data can be accessed whilst the device is disconnected (e.g.a salesperson on the road). Synchronization is a similar concept but also includes the idea that changes made in either copy of the data need to be propagated (synchronized) to the other. There are many commercial software offerings (e.g. [7-10]) which support synchronization and replication for mobile devices.

Message queuing[3] (described above) is a frequently used communication paradigm for mobile devices as it supports disconnectedness; however, it is typically not undirected.

2.4. User Characteristics

One of the key features of PDA use is that, as opposed to desktop applications, portable devices are used frequently for short periods and often on the move [14]. Content must therefore be selective and minimised so that users need not spend excessive time trying to locate the information they are after. PDAs are not well suited to the storage or display of large amounts of information - they are better suited to the display of data such as news headlines, weather, and stock prices.

Content-based messaging provides an excellent method for filtering data and selecting content of interest, as filtering will be done at the server, not at the PDA. A similar concept is the recent advent of web clipping [13] which is performed to reduce the amount of data in normal web pages.

2.5. Related Work

There are many systems available which support decoupled communication. A representative selection of these are examined in this section.

2.5.1. TIB/Rendezvous. TIBCO's TIB/Rendezvous product [3] is an established messaging middleware with many customer installations worldwide. TIB/Rendezvous decouples producers and consumers by using publish/subscribe; however, it uses subject-based addressing and does not provide full addressability of content.

TIBCO's service is referred to by themselves as "reliable delivery", and this "reliable" message delivery is implemented in the TIB/Rendezvous daemon. The routing daemon retains outbound messages for sixty seconds, and retransmits if clients have intermittent network failure - an insufficient period to support most disconnected devices.

TIBCO also has a feature known as "Certified Delivery" which has a decentralised, stream-oriented, peer-to-peer architecture. Rather than using a queuing mechanism at either the server or a proxy, producers store messages until each consumer has acknowledged receipt. This message delivery, acknowledgement, and retransmission is all done by the client libraries.

2.5.2. Siena. Siena (Scalable Internet Event Notification Architecture) [4] is an example of wide-area event notification content-based routing. Siena allows subscriptions to address all fields of notifications. The emphasis in this work is on scalability - supporting content-based messaging over a wide area network. There is no support for disconnected devices.

2.5.3. OpenQueue. OpenQueue [5] is an open source protocol for publish/subscribe message queuing. While connected to a server, a subscribing client receives published messages in real time. When a client reconnects after an absence, the server sends all messages queued for that clients while it was off-line. Messages are published in "topics" and for each subscriber to a topic the server maintains a queue of messages in that topic. OpenQueue can therefore be considered a cross between subject-based addressing and message queuing.

OpenQueue identifies each client by requiring them to authenticate or "log in" each time they connect. Clients remained subscribed at the server even while they are disconnected, and authenticate themselves upon reconnection.

2.5.4. Gryphon. Gryphon [11] is a distributed message brokering system - it maps a subscription database to a network of underlying brokers that distribute the messages. Gryphon supports content-based subscription and indeed, supplies content-based *routing*. Like Siena, the emphasis in this project is on scalability

2.6. Summary

Content-based messaging has many advantages over other forms of decoupled communication for disconnected devices. These include greater decoupling (no channel names); reduced data flow (more filtering can occur at the "server"); and greater flexibility. However, support for content-based messaging to disconnected devices is, in general, lacking. This work aims to address this issue by adding support for disconnected devices to the Elvin content-based messaging service. The following section describes the Elvin content-based messaging service in more detail and

3. Message queuing can form the basis for synchronization.

section 4 describes some of the issues involved in providing support for disconnectedness.

3. Elvin Content-Based Messaging Service

Elvin [2] began as a publish-subscribe notification service, but has since evolved into a content-based messaging service [12][6]. Elvin consists of:

- An easy to use API, allowing application developers to generate and consume information simply. There are language bindings for C/C++, Java, Python, Smalltalk, Emacs Lisp, and Tcl, which are supported by a number of development tools.
- Dynamic definition of both information formats (messages) and subscriptions. This is a key feature required to allow scalability across organisational boundaries.
- Flexible and dynamic message content delivery defined as the application developer requires. Information is distributed only to the points where it is needed, allowing greater system throughput to be achieved. The importance of bandwidth efficiency over individual throughput is a fundamental design criterion of the Elvin service.
- A simple but powerful subscription language able to express complex constraints on the information routed to applications. Elvin allows all of the information to be used for routing choices - everything behaves like an addressable subject in a more traditional publish/subscribe system.
- Quenching is an unique feature of the Elvin service. It allows producers to receive information about what consumers are expecting of them so that they need only generate the events that are in demand. This is important for some classes of producers where the act of producing the event is expensive.
- A decoupled security model designed to maintain the flexibility of publish/subscribe messaging. Traditional security mechanisms are point-to-point, allowing for authenticated communication between two parties. Elvin provides a flexible security mechanism where producers and consumers can have overlapping key sets that combine to allow multiple-party authorisation. This is used to control the delivery of notifications whilst maintaining the flexibility of loosely-coupled components.

In its basic form, Elvin operates by having a server acting as a notification router between multiple connected clients. Clients can act as producers and/or consumers of events, and the server is responsible for routing notifications of interest to consumers. This has been extended to include “federations” of multiple servers but the concept of routing notifications based on content to interested clients remains the same.

3.1. Elvin Applications and Concepts

Elvin has several established applications which have mostly been implemented for desktop PCs. This section describes two of these applications.

3.1.1. Tickertape. By far the most common Elvin application in terms of deployment numbers is DSTC’s Tickertape [16]. The Tickertape application is a scrolling one-line window providing users with a wide variety of information from many sources, using a minimum of screen space. DSTC also provides a number of producers for Tickertape for retrieving updates of web pages such as Slashdot and various news services. Tickertape is built using Elvin and functions as both an event consumer, receiving awareness information in the form of events, and an event producer, which enables Tickertape to be used as a communication application.

3.1.2. Eddie. Eddie [18] is a tool that collects data from the system it is running on or from nearby systems or services. Eddie can run rules over the collected data and take actions based on the results, making it a very useful system/service monitoring tool. A monitoring front-end such as Eddie could be written for hand-helds so that system administrators can have ready access to network information while working around the office.

3.2. Other Elvin Usage

Elvin is also in use in workflow applications, as a USENET filtering service, in various awareness applications, in education, and as infrastructure for CSCW [6].

4. Content-based Messaging for Mobile Users

There are many issues involved in seamlessly supporting content-based messaging in the face of mobility and disconnectedness. This section enumerates these issues and describes how they have been addressed in this work. Whilst Elvin has been addressed specifically here, these issues apply equally to other content-based messaging services. More advanced implementation issues are described in section 5.

4.1. Persistence

Obviously, in order to support the delivery of events or notifications to disconnected devices, a persistent data repository is required to store the events.

Elvin by design is non-persistent[4] - once clients discon-

4. As are most content-based routing systems.

nect from the Elvin server their subscriptions are no longer active and they miss any arriving notifications until they next connect and resubscribe. For mobile devices this creates a problem as they may be frequently disconnected yet do not wish to miss important notifications.

Rather than modifying (and possibly encumbering) the Elvin service, a prototype Elvin "proxy" has been developed which can store notifications while clients are disconnected.

In the standard Elvin system, there exist two main components: a client and a server. (Multiple clients may exist, supported by a single server or federation of servers.) The proxy model extends this by including proxies which act as normal clients to the server but as a proxy server to clients. Clients then connect directly to an Elvin proxy server rather than connecting to the Elvin server itself.

In the Elvin client/server model, an Elvin client must maintain a connection to its server to keep its subscriptions active. If the connection is closed (or lost), notifications may be lost. The Elvin proxy server works by maintaining a permanent connection to the Elvin server and remaining subscribed on behalf of the clients. Any notifications delivered by the server to the proxy while the clients are disconnected will result in the notifications being stored on the proxy until the clients next connect.

4.2. Subscription Groupings

The proxy must be able to handle multiple clients with separate sets of subscriptions. Therefore, the proxy needs to be able to distinguish these sets from each other, which was not possible using the original Elvin protocol. The concept of a "session" was introduced to refer to a group of subscriptions for a particular client or clients.

4.3. Multiple Receivers

The increasing ubiquity of computing and communication devices means that users can receive information via one or many devices. Mobile users characteristically have some form of PC or laptop which they may use in conjunction with (and sometimes in preference to) more mobile devices such as PDAs and mobile phones. To accommodate this and to allow for routing of messages to *where the user wants them*, users need to be able to access their subscriptions (or some subset of their subscriptions) from their Elvin desktop application or some mobile device. This means that sessions need not be client specific - sessions may span multiple clients or applications.

4.4. Non-destructive Notification Receipt

Users who use more than one device for receiving notifications may wish to receive notifications on one device but keep a copy on the proxy to be downloaded by another device at a later time. An example of this is someone who uses both a laptop and a PDA. They may wish to use their PDA for viewing notifications while mobile, but save a copy of the received notifications on the proxy for downloading and archiving by the laptop at a later time. Semantically this is similar to POP email clients electing to leave messages on the server or remove them.

To prevent multiple notification deliveries to "non-destructive" clients which leave and rejoin a session, the proxy keeps track of clients within a session to which a notification is delivered. Notifications are never delivered more than once to the same client.

4.5. Session Modification

Because a session may be shared amongst multiple devices or applications, if one registers a subscription then the other will not be aware of it. To handle this, the proxy should return a list of all subscriptions whenever a user joins an existing session to ensure that client and proxy state are consistent. Also, if more than one client is connected to the same session, if a client adds, modifies, or removes a subscription then the others should be immediately informed by way of an update.

4.6. Multiple Concurrent Sessions

Depending on what device they are using and the frequency with which they intend to use it, users may elect to use alternately configured sessions at different times. Client libraries and the proxy allow for a user to be connected to multiple sessions concurrently. The proxy will deliver only one copy of notifications which match subscriptions in more than one session to which clients are concurrently connected.

4.7. Notification Storage

Clients may spend long periods of time without connecting to the proxy, which could result in a large number of saved notifications if the proxy was to store them forever. In the case of notifications such as chat messages or stock prices, clients may wish to only receive the last few minutes worth, even if they were disconnected for hours or days. The proxy allows clients to specify a *time-to-live* (TTL) for each subscription, which will be the maximum time which the proxy should keep any notifications matching that subscription. Once a notification reaches its TTL it should be

"expired" by the proxy.

Similar to TTL values, clients may instead prefer to specify the *maximum number of notifications* to keep for a subscription. The TTL and maximum message number values should interact so that client requests are evaluated like "store notifications for up to X hours and store no more than Y of them".

5. Elvin Proxy Implementation Issues

The design and implementation of a new proxy protocol for Elvin raised various challenges to integrate smoothly with existing Elvin concepts. This section describes some of the issues which were overcome in developing the proxy prototype. A brief description of the proxy prototype's architecture is given first, followed by discussions on proxy security and proxy federation. Greater detail may be found in [19].

5.1. Proxy Implementation Architecture

The internal architecture of an Elvin proxy is very much implementation dependent; however, there are several common challenges faced in any implementation which are discussed below.

5.1.1. General Description. The Elvin proxy prototype has been developed using the alpha release Elvin Python language bindings and hence shares many characteristics of other Elvin applications which may use the bindings.

The proxy attempts to establish a connection to an Elvin server using either a supplied URL or server discovery. The proxy then accepts requests from proxy clients. For client requests which can be resolved by the proxy alone, such as a request to join a session, responses are returned immediately. For requests which require interaction with an Elvin server, the proxy's server endpoint is used to send the request in the same manner as a normal Elvin application. In each case the callback function for the server request is set to a function which sends a reply to the client when executed.

Notifications received by the proxy server from the Elvin server are delivered to connected clients whose subscription matches the notification. The proxy server may also store the notification if there are off-line clients which have matching subscriptions.

5.1.2. Delivering and Storing Notifications. When a notification arrives from the server, the proxy first builds a list of all the sessions to which the Subscription IDs belong. The proxy then generates a list of all clients connected to those sessions and delivers a copy to each. If there are any remaining matching subscriptions which do not have a client connected, the notification is then stored to be delivered later. The proxy strips from the packet any Subscription IDs to which the notification has already been delivered. Each of the delivered packets include only the Subscription IDs for that client, while the copy stored on disk by the proxy includes only subscriptions to which the notifications have not yet been delivered.

5.1.3. Expiring Notifications. The combination of time-to-lives and maximum message numbers for subscriptions creates the difficulty of managing stored notifications and ensuring these values are abided by.

Enforcing the *maximum message number* is the easier of the requirements - for each subscription a count is maintained of the number of stored notifications. This value is incremented when new notifications are stored and decremented as notifications expire or are delivered to the client. The proxy can check the number of stored messages for the subscription as new notifications arrive, and remove the oldest one if the maximum has already been reached.

Enforcing *time-to-live* (TTL) expiry of subscriptions provides a greater challenge, as given the granularity of TTLs is one second, the proxy may experience notifications reaching their TTL as often as once a second. If the approach taken was to ensure that notifications were always removed as soon as they reached their TTL, this could produce a large processor overhead due to checks every second.

One possible way of expiring notifications which have reached their TTL would be to ensure that when clients connect and are delivered their saved notifications, those which have exceeded their TTL are deleted and not delivered to the client. This alone would ensure that clients never receive expired notifications, but could result in an excessive amount of disk space usage, as many expired notifications could be stored for clients which connect infrequently.

To complement the method described above and ensure that a more efficient use of disk space occurs, a method to "sweep" expired notifications is also required. Because clients are guaranteed not to be delivered notifications whose TTL has been exceeded, sweeping does not need to occur with the same frequency as notifications expire and can be done at a more sensible period, as we are assured that expired notifications will not be delivered to clients. Our only objective of sweeping is to minimise wasted storage of notifications which have expired and should be deleted. For instance, if notifications are checked every one minute for expiry, this would result in an expired notifications overhead equivalent to the amount of notifications arriving at the proxy per minute, as only one copy of each notification is stored regardless of the number of client matching subscriptions. This approach appears to provide a good compromise between processing overhead (required for each

"sweep") and space overhead of expired notifications which have not been removed. The figure of one minute could be adjusted if it were later found that either the space or processing overhead was too great.

5.2. Security in the Elvin Proxy

Securing proxies is an important issue to be considered before they can be deployed widely. An unsecured proxy could present vulnerabilities such as sessions being maliciously deleted and secret keys being compromised. The methods for ensuring that such a situation does not occur are discussed below.

5.2.1. Session Security Schemes. In considering security for the proxy, three possible security schemes were considered - single session password security, multi-level session password security, and access control list security. This section gives a brief description of the three methods.

Password Protected Sessions

The current Elvin proxy scheme has sessions as its primary objects. Although the concept of separate "clients" exists, clients identifiers are unique only within a session and do not identify a person, device, or application in a global sense. The first option in securing the proxy is to have a single password which protects a session. The password is supplied when the session is created and is required to join or delete the session. Once the password is provided to join a session, users are free to make any modifications to the properties of the session, such as adding, modifying, or deleting subscriptions, quenches, etc. This model assumes that any object who has the session's password is trusted and there is no need to distinguish different levels of access rights for sessions.

Multi-level Passwords for Sessions

This second password model builds on the first but instead defines multiple session passwords which correspond to different levels of access control. Although there exist a large number of operations available (such as joining a session, adding/removing subscriptions, etc.), these can easily be aggregated into two or three levels of security. The bottom level in both cases is a read-only "user level" which would be a password which only gives people or applications the right to join and leave a session, and receive notifications while joined.

For two-level access control, the only other password would be an "admin." password which gives the ability to execute all other operations which might be considered destructive in some way (e.g. Subscription addition/modification/deletion, session deletion, etc.). Under this scheme anyone with the admin. password has the ability to leave the session as a shell, or even take ownership of it by deleting it and then creating a new one in its place with the same SessionID. Obviously in this scheme, anyone with the admin. password would need to be fully trusted.

Three-level access control is similar to the two-level access control except that it includes the concept of ownership of a session by having not only "user" and "admin." passwords but also an "owner" password. The difference in access rights between admin. and owner would be that those without owner access cannot delete the session. While the admin. password would still provide the ability to delete all subscriptions, it would not allow a non-owner of the session to delete the session and recreate it with a new admin. password. This means a session-owner can allow other users to modify their session (perhaps even removing all subscriptions or adding bogus ones) but ownership of the session can not be seized. The owner password would of course have the ability to change the admin. and user passwords.

Access Control Lists

Access control lists (ACLs) could provide a flexible, but complicated, method of securing the proxy. This would introduce the concept of unique "users" to the proxy schema, with each session belonging to a particular user. Owners of sessions could then define access control lists for each session which define the access rights for each user. This could obviously be made more efficient by aggregating operations and also adding users to groups to generate smaller ACLs. This security scheme would result in a significant shift of the proxy away from being session-centric to user-centric.

5.2.2. Merits of Security Schemes. In comparing the various security models, the first comparison must be between the session-password and ACL methodologies. While access control lists undoubtedly provide the most flexibility and advanced configuration possibilities, it is doubtful whether such a complicated model is necessary. This is quite an important factor when it is considered how unlikely it will be that sessions are shared amongst more than one user. If there is a need to share sessions amongst users, then it is likely that the users will be trusted.

The major benefit of the single password scheme is its simplicity, with the primary disadvantage being less flexibility than the other two schemas. Using a multiple password scheme would provide greater flexibility and introduce the concept of "ownership" of sessions into the security scheme. Given that, in the majority of cases, users will be sharing sessions amongst their own devices, it seems unnecessary to bother with the distinction of multiple levels of access for sessions which will mostly be used by a single person.

5.2.3. Integrating with Elvin's Security Keys. The Elvin protocol includes its own security layer for securing subscriptions and notifications which needs to be integrated

into the proxy's client handling.

Existing Key Mechanism

Elvin clients use producer and consumer security keys to provide access control over the Elvin protocol. The security keys may be used by producers to restrict which consumers are authorised to receive notifications and also by consumers to restrict which producers they receive notifications from.

Clients can specify keys in two ways. The first is to have connection-based keys that last for all notifications the client makes or receives while connected. The second is to use specific keys for each notification or subscription which is sent. These keys are added to any connection-based keys to define the security keys for the subscription or notification.

Proxy Key Mechanism

A security challenge faced by using the proxy is that, with the proxy maintaining a single connection to the server, the proxy cannot use any connection-based keys or else suffer possible security breaches between sessions. Proxies need to keep state information for the connection keys of connected clients and only when a proxy-add-subscription request is made should the proxy merge the connection and subscription keys and put both into the add-subscription request which is sent to the server. As clients need to know the existing keys in order to change them, it will also be required to transmit these keys as part of the subscription's details to clients upon joining sessions. This should not be considered a security risk as we are already assuming that, through session passwords, all users of a session may be trusted. The proxy's handling of client keys should be transparent to the proxy clients and require no change in how clients use keys.

5.3. Proxy - Server or Service?

As of version 4, Elvin has been established as a service rather than just a collection of networked servers. With such tools as server discovery and federation, clients now conceptually connect to an Elvin "service" rather than to just a server. Given the Elvin service "cloud", it is important to consider where a proxy server or service should conceptually lie and what interconnection of proxies is possible.

5.3.1. Elvin Federation. Elvin has been developed to allow both local area and wide area federation of Elvin servers. With the aid of server discovery, the coupling of Elvin clients to servers has been decreased and the concept of connecting to a single Elvin service promoted. The Elvin service has been extended beyond that of a single server to a federation of autonomous servers cooperating to route messages to consumers.

Local area federation provides the ability for organisations or related groups to federate servers so as to provide features such as failover to backup servers, load-sharing ability, and the linking of sites into an autonomous network. Wide area federation is envisioned to provide a global communications "backbone" allowing notifications from anywhere in the world to be read by consumers with matching subscriptions anywhere.

5.3.2. Proxy Coupling. Acting as both a server to clients and a client to servers, it is not obvious as to whether an Elvin proxy should be tightly coupled to clients or servers, or loosely coupled to both. On one hand the proxy could be considered as a service itself to complement the Elvin service; while, on the other hand, it could also be considered more of a stand-alone "home" or "docking station" for mobile devices.

The current prototype model of the proxy uses a model that is closely coupled with connected clients. Clients explicitly connect to a proxy and must reconnect to the same proxy each time. While introducing proxy discovery could reduce the need for knowing Elvin proxy addresses, the problem exists that clients could do this only once because after their first connection they'd be required to connect to the same proxy each time. Nevertheless, implementing proxy discovery is certainly a feature which could be added for the benefit of new clients trying to locate a proxy.

5.3.3. Establishing an Elvin Proxy Service. A united proxy *service* could be useful for mobile clients who migrate between networks but wish to have notifications proxied. Ideally, such clients could use proxy discovery at their different sites to locate a local proxy and then resume receiving and sending notifications. Also, by connecting to a local proxy rather than their home proxy, clients may be able to achieve lower latency in their connection depending on the topological separation of the two networks.

The problem to overcome in transforming the proxy into a service is that proxies are highly dependent upon state whereas Elvin servers are stateless. With a federated Elvin service, the server to which an Elvin client connects is not important so long as the server is federated with the required servers. In the case of the proxy, state information must be stored, so it makes distributed federation much more complicated.

One method for providing universal availability of a proxy service is to have session names globally unique and hence identifiable by all proxies; however, this greatly limits the namespace available for naming sessions and adds a large degree of complexity and latency to the previously simple task of allocating session identifiers. Alternatively, session identifiers could remain unique within a proxy only and clients could instead use an additional identifier to identify their home proxy and then locate their session. This leads to the problem that clients now need to be able to ex-

plicitly name their home proxy to locate their session, and if clients are able to do this, they could just connect to their home proxy each time rather than rely on proxy discovery.

While the benefits of decoupling proxies from their clients by establishing a proxy service appear limited due to namespace issues, developing a unified proxy service with proxy discovery could possibly still prove efficient for roaming mobile hosts. Where efficiency could be gained is that notifications sent to an Elvin federation currently need to be relayed through a client's home proxy regardless of the location of the client. If the client spends a considerable proportion of its connected time topologically separated from its home network then it could prove efficient to move details of its session to the closer proxy so that notifications do not take an inefficient path. This would, however, result in a complex handover mechanism which would make it difficult to ensure that no notifications were lost nor duplicated in the process (Elvin does not impose global ordering of notifications so the two proxies could receive notifications with differing interleaving).

6. Conclusions

Distributed computing looks to provide some interesting challenges and opportunities as distributed systems incorporate more mobile semi-connected devices. As network topologies become increasingly dynamic, the challenge of disseminating data to interested parties will push the boundaries of existing communications models. Interprocess communication based upon a content-based messaging paradigm presents itself as an ideal way to decouple autonomous network components to promote flexibility and scalability of mobile devices.

This paper has described the issues involved in supporting content-based messaging on mobile devices and how these issues have been overcome in the implementation of a proxy-server to provide disconnectedness support for the Elvin content-based messaging service.

Acknowledgements

The work reported in this paper has been funded in part by the Cooperative Research Centre Program through the Department of Industry, Science and Resources of the Commonwealth Government of Australia.

References

[1] Houston, Peter. Building Distributed Applications with Message Queueing Middleware. March 1998, Microsoft Corporation.

[2] Elvin website. http://elvin.dstc.edu.au/ Last visited 8/10/00

[3] TIBCO. TIB/Rendezvous Concepts. http://www.rv.tibco.com. Last visited 28/11/00.

[4] Siena website. http://www.cs.colorado.edu/serl/dot/siena.html. Last visited 8/10/00.

[5] OpenQueue Home Page. http://openqueue.sourceforge.net. Last visited 2/10/00.

[6] Bill Segall, David Arnold, Julian Boot, Michael Henderson and Ted Phelps. Content Based Routing with Elvin4. Proceedings AAUG2K, Canberra, Australia, June 2000.

[7] Information Transport Associates Corp., "Database Synchronization and Replication Software for remote and mobile computing," http://www.itacorp.com. Last visited 30/11/00.

[8] Synchrologic Corp., "iMobile Suite," http://www.synchrologic.com/about/about_imobile.html. Last visited 30/11/00.

[9] XcelleNet, Inc., "Afaria," http://www.afaria.com. Last visited 30/11/00.

[10] Extended System, "XTNDConnect RPM: Programmable Middleware for Mobile and Windows Applications," http://www.extendedsystems.com/products/rpm. Last visited 30/11/00.

[11] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tuschar D. Chandra, "Matching Events in a Content-based Subscription System," *Principles of Distributed Computing*, 1999.

[12] Arnold, D., Boot, J., Henderson, M., Phelps, T., Segall, B. Elvin - Content-Addressed Messaging Client Protocol, Proposed Internet Draft, 2000. http://elvin.dstc.edu.au/download/internet-draft.txt. Last visited 28/9/00

[13] Palm.Net Service website. http://www.palm.net/ Last visited 8/10/00.

[14] DSTC M3 Group website. http://www.dstc.edu.au/m3/. Last visited 10/10/00.

[15] Hinze, Annika and Faensen, Daniel. A Unified Model of Internet Scale Alerting Services. In Proceedings of the International Computer Science Conference, Hong Kong, 1999.

[16] Sara Parsowith, Geraldine Fitzpatrick, Simon Kaplan, Bill Segall, Julian Boot. Tickertape: Notification and Communication in a Single Line. Proceedings Asia Pacific Computer Human Interaction 1998, Japan.

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

[18] Rod Telford and Chris Miles. Eddie (Essential Distributed Diagnostic and Information Engine). *Proceedings 6th SAVE-AU Annual Conference*, July 1998.

[19] Arkins, R., *Persistent Elvin For Mobile Devices*, BInfTech honours thesis, The University of Queensland, November 2000.