Content Based Routing with Elvin4

Bill Segall, David Arnold, Julian Boot, Michael Henderson and Ted Phelps
CRC for Enterprise Distributed Systems Technology (DSTC)
The University of Queensland, St Lucia, 4072, Australia
Phone: +61 (7) 3365 4310 Fax: +61 (7) 3365 4311

elvin@dstc.edu.au

Abstract

Building on experience with a general-purpose notification service, we describe the design and implementation of a second-generation content-based messaging system. Elvin4 *includes a novel security framework, internationalisation, a powerful subscription language, and a modular pluggable protocol stack.*

We discuss its evolution from previous versions, differences from related work, and describe the transition in underlying ideology from notification service to content-based routing and the effect this has had upon the design.

1. Introduction

Mechanisms such as RPC, message queues and multicast can all be termed *directed* communication models: the destination of a message is specified at the time it is sent. The destination can be made more transparent through the use of a name (and name server) or a group identifier (in the case of multicast), but it remains the sender's responsibility to direct the message.

This requirement causes difficulty in situations where the sender does not know the destination, when it is constantly changing, or when the number of recipients varies. A common solution is to introduce an explicit agent or proxy at a known address to which the sender always delivers the message. This agent then handles the message distribution.

Elvin3 [SA97] implemented an alternative mechanism. It provided a means of *content-based addressing*, sending simple structured messages and allowing receivers to use a subscription language to select messages of interest, with a mostly-transparent router process taking the place of the explicit third party.

Over three years of deployment, this model of network programming has been proven simple, flexible, and performant over a range of application areas both within our own organisation and by external clients. However, deployment has also highlighted the need for additional features and exposed some flaws in the protocol design that we have tried to overcome in developing the next version, Elvin4.

In the next section we describe Elvin3 with particular focus on its limitations. We introduce some related work, examine some of the applications in which it has been used, and identify strengths and weaknesses leading to the design goals for Elvin4.

We then discuss the Elvin4 protocol and its implementation, before reflecting upon the transition in our ideology between a notification service and a content-based routing infrastructure and finally discuss our plans for future work in this area.

2. Elvin3

Elvin3 was an attempt to demonstrate that content-based addressing was a viable model for distributed inter-process communication. It was deliberately simple, particularly at the programming interface and did not attempt to provide a full set of features, omitting for example any security mechanism.

2.1. Protocol

The Elvin3 protocol used a long-lived connection between client programs and the routing daemon. It was based on TCP/IP, with a custom marshalling layer supporting six packet types (see figure 1). All packets were sent asynchronously; the connection was assumed to be reliable and there were no acknowledgement or response packets.

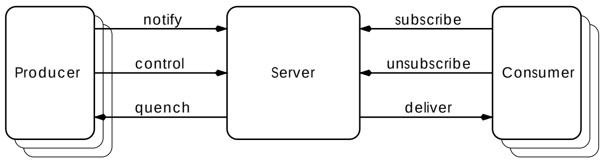


Figure 1: Elvin3 protocol architecture.

The absence of acknowledgements required that all packets were semantically verified within the client library. This was especially onerous for subscription requests: the client library had to parse the supplied expression and this parser added substantially to the size and complexity of the library code. Frustrating differences between regular expression libraries in different operating systems and languages also caused problems with the server rejecting expressions accepted by the client.

The marshalling layer itself was a source of problems: it had several outright bugs and caused significant loss of floating point precision. Whilst easily fixed, compatibility with deployed clients quickly became more important than correctness. These problems were the consequence of the failure to use an existing, standard, marshalling protocol.

2.2. Quenching

An important feature of the Elvin3 design was the introduction of *quenching*. Quenching provides a mechanism for clients to determine whether subscribers are interested in their messages, and enables clients to reduce (and thus, quench) the message traffic by sending information matching only current subscriptions.

What quickly became apparent was that while the idea was successful, the implementation did not scale to the number of subscribers we were supporting. After being enabled (by the *control* packet, see figure 1), whenever a subscription was changed, and at most once every ten seconds, a quench packet was sent to all clients with quenching enabled. However, the quench packet contained a complete copy of the subscription database, and the transmission time of the packets in some cases exceeded the minimum period between updates.

In addition, the subscription database was sent as a boolean expression in string format. This proved very unwieldy for programmers: it had to be parsed to extract the required information, and this made the task of writing a well-behaved, high-volume client unreasonably difficult.

2.3. Implementation

The protocol implementation consisted of a C client library and server, both written for a modern Unix environment. Solaris, Linux, OSF/1, HP/UX, AIX and Ultrix were all used at various stages, but the requirement for a POSIX threads implementation made deployment problematic for *BSD and some older machines. Similarly, the use of thread cancellation prevented a port to Windows NT.

Client libraries were provided for a variety of languages. Some were native implementations of the protocol, and in particular, the Java mapping was pure Java from its first incarnation. The Python mapping started out using the C library under a thin wrapper, but was later rewritten entirely in Python.

Supported languages included Java, Python, TCL, VisualWorks Smalltalk, and Emacs LISP with external work on Allegro LISP and Lambda MOO also undertaken. Notable for its absence here is PERL. We had several attempts at writing a direct PERL mapping, but the use of threaded callbacks in the C API made it difficult, and PERL users were constrained to using command-line utilities to send and receive messages.

2.4. Administration

Deployment of Elvin3 was complicated by the conflicting requirements of our funding bodies and general users. We supplied the system as a bundle of all components, together with their dependencies, which made for easier installation on a raw machine, but tended to cause conflicts with the average well-stocked /usr/local. The integration was pervasive, and unbundling the Elvin components proved too difficult.

Once installed, few problems occurred for typical sites with the exception of clients locating the server. While Unix sites tend to share filesystems, making a central configuration file relatively simple, Windows machines are typically installed with private copies of applications. We opted to rely on the DNS, using a well-known host name to locate the default Elvin server machine for a network. This proved quite difficult for many sites, where the addition of a CNAME record for a protocol the DNS administrators had never heard of took some negotiation.

Finally, the server's reporting and control interface was constrained by the lack of security for messaging. We relied instead on the security mechanisms of the host machine, using signals and log files for server management. This made remote administration increasingly difficult as we deployed more servers across our network. Additionally, the available statistics were only minimally useful and made no provision for service metering, capacity management or QoS monitoring.

2.5. Summary

Despite all these issues, it is important to state that Elvin3 had a number of significant strengths.

Most importantly, the fundamental concept of content-based addressing has proven successful. Elvin3 has demonstrated that the concept is useful in a range of application areas, is feasible to implement with adequate performance, and is quickly comprehensible to programmers of varying skill levels.

The programming APIs are simple and together with the wide range of supported languages, this has minimised the programmer's learning curve. As a result people were likely to use Elvin when they wanted to write something quickly, or integrate existing components.

Alongside this local development and observation, we have observed a wider trend towards undirected communications in particular, and messaging in general. The commercial adoption of publish/subscribe and its benefits for application architecture have been matched by research interest in notification services and novel addressing mechanisms.

3. Related Work

Elvin provides a means of transmitting unacknowledged messages between distributed processes. It is unlike RPC (and remote method invocation), stream protocols like TCP and multicast protocols ranging from raw IP multicast through the various mechanisms for reliable group communication. Due to its use of content-based addressing and the requirement that messages are only delivered to connected clients, it is also distinct from messages queuing and similar store and forward systems.

3.1. Notification Services

Elvin3 shares most features with what are often called notification services. This section introduces a selection of similar services, and describes them briefly. We pay particular attention to the features that differ from those of Elvin.

3.1.1. Keryx Notification Service

Keryx [Low97] is a Java notification service, designed by HP Labs in Bristol. It uses an elegant on-the-wire *transfer syntax* called Self-describing Data Representation (SDR) to describe both messages and subscriptions. Subscriptions are SDR expressions conforming to a restricted grammar, the Default Filtering Language (DFL). DFL predicates are comprised of type tests of SDR elements, arithmetic operations, list operations on compound values and boolean combinators on subexpressions.

Keryx messages are structured as name-value pairs in SDR. The values range from simple data types to lists and nested name-value maps. The underlying transport protocol is TCP-based and quite simple. It is also possible to send SDR messages over alternative transports, although the available implementation does not include this feature.

3.1.2. CORBA Notification Service

After specifying a channel-based event service [OMG95], the OMG developed an extended specification, the CORBA Notification Service [CNS99] to provide filtered channels. A CNS message object can be one of three types: a CORBA Any, a statically typed CORBA object, or a Structured Event comprised of a type, some filterable name-value pairs, and a non-filterable payload. Subscribers connect to a channel, and may register a filter for message object's types, Any value, and the name-value pairs.

CNS provides a means to federate channels into a routing network for events and to specify various qualities of service on a channel, such as persistence or reliability. All communications within CNS are based on CORBA method invocations.

3.1.3. Gryphon

Developed at IBM's TJ Watson Labs, Gryphon [ASSAC99, BCMNSS99, BKSSST99] is an ambitious system that maps a subscription database to a network of underlying brokers that distribute the messages. The subscription evaluation includes security and administrative filtering attributes and is being extended to provide additional services, such as storage and forwarding, within the broker network.

3.1.4. XmlBlaster

An open source development, XmlBlaster [XmlBlaster] uses an XML syntax to describe messages consisting of a filterable header, an opaque body, and a system control section. Filters, in the form of XPath [XPath] expressions are evaluated over the header to select messages for delivery to subscribers.

XmlBlaster also includes a message queuing system within the same framework, with the message control section indicating whether it is directed to a set of destinations, or published for access by subscription.

3.2. Other Systems

A range of other systems, both notification-style and more general message-oriented middleware (MOM), have been developed. They range from large-scale enterprise application platforms [TSS95, Tal-SS, IBM-MQ], to desktop buses for inter-application coordination [OPSS93, Sun93]. While not considered here, we have investigated a wide range of these during the development process.

4. Applications

While content-based addressing enables a new class of applications, the rate of change in programmer mindset from the more traditional IPC/RPC/messaging has meant that many early programs do not fully utilise the power of the mechanism. As an example, our most widely used application, Tickertape, is basically a channel-based application, built using a content-based infrastructure. However, as programmers' experience has matured, we have begun to see the unique abilities of content-based addressing being utilised.

This section introduces a selection of Elvin3 applications, and describes their use of the system.

4.1. Tickertape

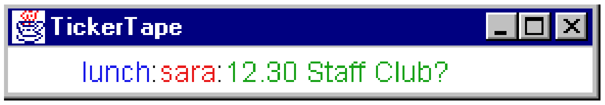
Tickertape [FPSK98, PFKS98] is a lightweight, tailorable desktop tool that provides an interface to transient information via a single-line horizontally scrolling message window (like a stockmarket ticker). It is used as a filter tool for public information sources (Usenet news, CNN and ABC stories, netcomics etc), as a chat tool, and is localised with various information sources (CVS, RCS, web hits, downloads, email notification).

It uses Elvin messages of the form:

to	lunch
message	12:30 Staff Club?
from	sara

Figure 2: An example message.

which are displayed like this in the scroller:


TickerTape

lunch:sara:12.30 Staff Club?

Figure 3: Tickertape Scroller.

The Tickertape client has the ability to subscribe to groups by name and filter the messages for a group by the content of the other fields.

Tickertape became extremely popular, in part because Elvin3's simple APIs made it easy to make local information easily available. This was not without some unfortunate side-effects (tickerspam!) and led to a need for us to address access control for communications between ourselves (private groups) as well as for personal instrumentation (such as email subject notification via Tickertape).

4.2. Awareness *biffs*

One of the early uses of Elvin was small awareness applications of a generic class that, for obvious reasons, we called *biffs*. Originally used by *xpilot* gamers to inform each other who was playing, it was quickly repurposed as *coffeebiff* and has proven quite popular as a social awareness tool.

The interface provides a coffee icon on which to click, a one line scrolling list of current coffee drinkers, and a count of how many people are drinking coffee. Besides keeping our sociologists busy, it has been most interesting from a distributed systems algorithms perspective, leading to investigations into state maintenance and sharing without central repositories that have applications beyond Coffeebiff's lighthearted domain.

4.3. System monitoring

Filewatcher is a daemon that generates Elvin messages when a monitored filesystem is changed. Without operating system support (now available on Linux and Windows), monitoring a complete filesystem is too CPU and I/O intensive, but Elvin's *quench* facility allows the filewatcher to monitor only those files for which a subscription exists.

While initially quite successful, the filewatcher became less useful due to the inadequacies of Elvin3 quench mechanism. This experience contributed to the complete restructure of quenching in Elvin4.

Similar utilities were written for web and ftp server logs. Whenever a new record was written to the log, it was parsed, and the information emitted as an Elvin event. After our experience with the filewatcher, the log watchers did not attempt to quench the traffic.

EDDIE [TM98, Mil99] is a large, general-purpose, systems monitoring tool originally written by staff at connect.com.au for internal use. It's now freely available[1], and optionally uses Elvin to distribute notifications of abnormal conditions to systems staff via Tickertape or a GSM SMS gateway.

[1] http://www.codefx.com.au/eddie/

4.4. Scoop

Scoop is a small program that dumps an entire Usenet news feed into Elvin for subsequent subscription by clients. Whilst it is an obvious case for quenching, we have been using it to drive some volume testing, emitting up to 4Gb of data in some 120K messages per day. The service has proven extremely useful and has made reading Usenet usage more focused and productive. It is intended to make this a quenching client for Elvin4.

4.5. Breeze

Breeze [Breeze] is an event-driven workflow engine written at DSTC. It uses the Elvin3 Java API and message transport to build an asynchronous RPC mechanism. This allows the workflow engine to control heterogeneous components, and to transparently support visualisation of the workflow state.

A number of additional applications have been developed using the Elvin infrastructure, both within DSTC and by other organisations. Those mentioned here are typical of their character, or represent (in the case of Tickertape) the focal point for an 'ecology' of smaller applications sending or receiving messages used by other contributors.

5. Analysis

Each of the messaging systems has similar functionality to Elvin3, and yet all differ substantially in the details of their implementation. In comparing these systems, we focus on two core properties: the addressing model, and the support for message persistence, and related issues of reliability.

5.1. Addressability

The basic focus of our work has been the means of addressing messages. Usually, a message's destination is specified entirely by the sender. This applies to everything from physical mail and telephone calls, to message queuing systems.

Alternatively, the sender can share the role of message selection by specifying a partial address using a channel identifier or by providing addressable metadata independent of the message body, and allowing the receiver to further refine the traffic stream. This is similar to Usenet News, for example, where the sender directs messages to a group, where the receiver then performs further filtering on the basis of sender and subject, before reading the content.

Finally, the sender can specify no destination at all, leaving the selection of messages completely up to the receiver. Analogies for this mechanism are somewhat stretched, but perhaps it could be seen as being like a search engine, where every web page in existence (when the crawler last crawled) is addressable by its content.

One of the benefits of using partially-directed or undirected communication is the reduction in coupling between the communicating parties [ASBBLK99]. As distributed systems become larger, and more subject to piecemeal extension and evolution, closely coupled interfaces become difficult to maintain. Undirected communication reverses the nature of addressing from producers *push*ing messages to consumers *pull*ing them. Of course, while it is possible for a receiver to select messages solely on the basis of their originating address, this reversal would make little difference to system design. Receivers would now have to locate senders, and the arity and identity of communicating parties is still fixed; the only thing that would change is the direction of traffic flow.

Channel, subject and content-based addressing schemes attempt to reduce the coupling between the parties in a communication by removing the specification of the involved parties' identity. Channel-based services require the producer to nominate a channel from which subscribers may receive their messages. This is the least flexible scheme; the direction of messages to a channel restricts their visibility, effectively partitioning the address space, and coupling the parties via the channel identifier.

Subject-based services split messages into an addressable subject, and an opaque body. Subscribers may select messages using filter expressions on the subject. This is a popular scheme, implemented by well-established commercial products such as TIBCO Rendezvous [TSS95]. Such systems sit part-way between channel-based and content-based schemes in their flexibility: their address space is global (all messages are equally visible), but the addressability is limited to a single field, and the burden on the sender is to maximise the exposure of information possibly useful for selecting a message within a single, often textual, field.

XmlBlaster uses an extended form of subject-based addressing, splitting messages into an addressable header, and an opaque message body. The entire message is an XML document, but the application of consumer's XPath subscription expressions is restricted to the header sub-section. The rationale behind this distinction would appear to be that the body section is delivered to the application, but the header section is metadata added to the body purely for the purposes of routing. This distinction is artificial, and constraining: a change in the basis for routing decisions could require that additional information from the body be made available in the header, and because of the separation, this will require that the source program(s) be modified.

CNS has a similar mechanism: some components of the message objects cannot be addressed by the consumer's filters, while Elvin3, Keryx and Gryphon provide complete addressability of the message's content. Such *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 application that has benefited most from this absence of coupling is Tickertape. The initial message format was defined by the subscription of the receiver (the scroller). Over time, features were added to the scroller, for example, to accept MIME attachments and to replace scrolling messages with new contents, but the addition of these fields to the message definition did not mean that the existing senders or receivers stopped working: the original version of Tickertape still functions (without the new features) after over 3 years of enhancement to the basic protocol.

Additionally, the use of content-based addressing has meant that despite the basic Tickertape GUI model being channel-based, some users have opted to search the message text for keywords of interest to them, by registering a suitable subscription. Another collects and publishes a list of known groups using the quench facility to obtain a copy of the registered subscriptions, and parsing for those matching the Tickertape definition.

Similarly, the use of Elvin in Breeze enables multiple, independent components to receive the messages from the workflow engine indicating changes in state, and to perform various functions, from initiating the next step in the workflow, to driving a visualisation of its progress.

5.2. Persistence and Reliability

In discussing notification services, Ramdunny [RDR98] defines a *pure* notification service as one where: "the server is entirely separate from the datastore". This separation of function between message routing and message persistence becomes our second criterion for comparison. Message routers, such as Keryx or Elvin3, deliver messages to connected clients with a matching subscription. Subscriptions are not retained while the client is disconnected, nor do matching messages accumulate waiting for the client's session to be re-established.

CNS, XmlBlaster and Gryphon all provide some level of support for queuing messages for a disconnected client. This suggests the notion of reliability: the server guarantees that a message received will be stored until the client can collect it, and this in turn leads to qualities of service for messages. Using QoS, messages may prioritise themselves in the queue, expire if not retrieved by a given time, be replaced by later messages containing updated information, etc.

Naturally, this additional functionality makes the implementation of the service considerably more complex. The necessity for persistent storage and its associated overhead make for a sharp distinction in the observed performance of those services that function strictly as routers, and those that provide a 'reliable' service.

This does not mean that content-based addressing cannot be combined with a mechanism for persistence, but argues for separation of the functionality.

Some form of persistence has been a common request from users of Elvin. Coffeebiff is an exception to this rule, where the current state is the important property, and historical information irrelevant. However, none of the applications have yet required a type of persistence that needed additional features in the routing daemon for its implementation.

Reliability is a more complex issue, clouded by the fact that quantitative measures appear not to be as satisfactory as a more general qualitative statement. The major issue is that of federated message routers, where the sender successfully delivers a message to an initial router from which it is then forwarded, potentially across huge numbers of linked systems.

In this scenario, 'reliable' delivery would mean taking a global snapshot of the combined subscription databases of all routers at a single point in time, and ensuring that those clients whose subscriptions matched the message acknowledged its delivery. This is obviously not scalable.

Content-based addressing encourages decoupling between senders and receivers, and this proves to be the antithesis of guaranteed reliability. Retaining Elvin3's *best effort* client semantics, together with a more rigorous approach for inter-router forwarding seem to be the best approach for further development.

6. Elvin4

Even before the issues that arose during deployment and application development were known, we had identified a range of features that could not be included in Elvin3 for various reasons, and from this arose the prospect of a next major revision. Throughout its deployment, feature requests were usually answered with "wait for Elvin4". Of course, there came a time when we actually had to write it, and in doing so, select the features to be included in the new version.

In this section we discuss the design goals for Elvin4, with reference to the identified problems with Elvin3, ideas from other messaging systems and our experience with applications using a content-based addressing system as discussed in the previous section.

6.1. Design Goals

The basic design goals for Elvin4 were

• better protocol design and implementation
• some additional message data types
• some changes to the subscription language
• internationalisation
• a security mechanism
• usable and efficient quenching
• automatic server discovery
• scalability to more clients, and beyond a single server

Each of these goals is discussed in the context of the implementation below.

6.2. Implementation

After some analysis and much discussion, we decided that rewriting the system from the ground up was required. We retained the use of C as the implementation language, after serious consideration of Java. It was felt that C was both portable to more platforms and sufficiently better suited for the task of writing networking software to overcome the lack of garbage collection and standard library support available in Java.

Development has proceeded on both Unix and Windows NT roughly in parallel, and despite some difficulties in mapping concepts to the different mechanisms provides by the two systems, the code is relatively clean.

6.2.1. Protocols

The Elvin3 protocol was TCP-based with a custom string-oriented marshalling. It was not modular; replacement of the protocol meant rewriting large sections of the server. For Elvin4, we decided that a modular approach was worthwhile, despite the loss of performance inherent in such an approach. This would also allow us to support multiple protocols satisfying different requirements.

Consequently, Elvin4 supports an abstract protocol stack comprised of three layers: marshalling, security and transport (see figure 4). Each layer of this stack may have multiple concrete implementations available

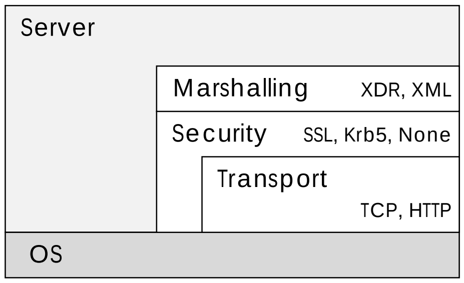


Figure 4: The Elvin4 abstract protocol stack.

within a single server. Interfaces between the layers are well-defined, and in particular, the security layer forms a complete encapsulation of the transport interface, to support protocol implementations, such as OpenSSL, where this is required.

We currently provide concrete marshalling layers for XDR [RFC1832], XML [XML98] and are working towards a serial protocol for embedded and handheld devices. Security modules for SSL [SSL96] and Kerberos 5 [NT94] are being developed, together with a 'none' module which simply passes messages to the transport module, for which we have TCP, UDP and HTTP implementations.

This modularity has also led to the need to describe the protocol stack used for a given server endpoint. We have adopted a URL format, encapsulating the stack description, location data for the endpoint (i.e, a host and port), and other server properties (see figure 5).

Message and quench delivery packets remain asynchronous and together with message emission, unacknowledged. The remainder of the protocol is now acknowledged, supporting connection management, registration of subscriptions and quench requests, and configuration of security keys. A final abstract packet type supports extremely simple producers, using unconnected message emission, for example over a UDP transport.

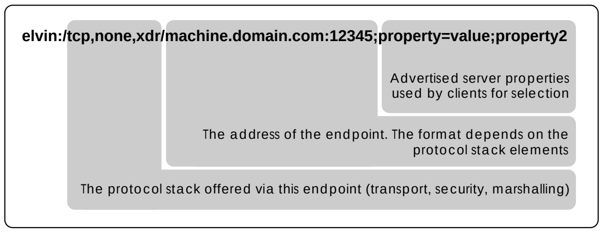

elvin:/tcp,none,xdr/machine.domain.com:12345;property=value;property2
Advertised server properties used by clients for selection
The address of the endpoint. The format depends on the protocol stack elements
The protocol stack offered via this endpoint (transport, security, marshalling)

Figure 5: Elvin4 URL example.

The adoption of XDR as the default marshalling standard, and the change to using RPC-style interactions for the ’administrative’ protocol functions has overcome the protocol-level problems from Elvin3. The modular protocol architecture also means that problems identified in future can be deployed while retaining compatibility with existing clients.

6.2.2. Datatypes and Subscriptions

As for Elvin3, Elvin4 messages consist of a set of named values. In addition to Elvin3’s supported data types of integer, string, and floating point, Elvin4 provides 64 bit integers and an opaque type for binary data such as images or compiled code.

In addition to a simple equality test, Elvin3 provided a POSIX Extended Regular Expression (ERE) matching operator for strings. After observing its use, we have introduced some simpler string matching routines: *begins-with*, *ends-with* and *contains*, with optional case insensitivity. While retaining the ERE operator, these tests cover a large proportion of our observed usage, and are both easier for the programmer and can be effectively optimised within the router.

A second extension was to support simple arithmetic within the subscription evaluation engine. Some Elvin3 applications were required to match a large number of messages, and perform secondary filtering which could be quite simply done by the server. Integer and floating point arithmetic and integer bitwise operations are now available within the subscription language.

6.2.3. Internationalisation

A large amount of effort has been made to support internationalisation with Elvin4. The message string data type is now UTF-8 [Unicode] encoded so that international characters can be represented. This has required additional subscription operators to support normalisation and comparison strengths, however the language remains very similar to Elvin3 for simple cases.

Additionally, the abstract protocol error packet contains a message code and list of arguments supporting the use of message catalogs in the client. The error packet also includes the message string in the server’s native language, to aid debugging and provide a common-case default for simpler clients.

6.2.4. Security

One of the major challenges for Elvin4 was to create a mechanism for authorising message delivery. The difficulty is that this authorisation forms a coupling between the message producer and its consumers, and it is the absence of coupling that is a key benefit of content-based addressing.

Elvin4 controls access to messages through the use of one-way keys. Producers may supply a set of raw keys that are transformed by a one-way hash function within the server, prior to matching. To receive a message containing keys, a subscriber must supply one or more keys matching the *transformed* keys from the producer.

While delivered messages are annotated to indicate that they matched a subscription key, subscribers may also elect to receive only secured messages. Both producer and subscriber key sets may be associated with the server connection to avoid resending the key set with each operation.

The distribution of the keys to both producers and consumers is not managed within Elvin. Applications and individual sites may utilise a variety of mechanisms, ranging through shared filesystems, directory services or even smartcards, to share keys.

Given that the possession of a key enables access to protected message traffic, the transmission of keys between the clients and server must be encrypted to ensure security. As described above, the Elvin4 protocol stack supports the use of a security layer to perform this function.

A more complete description of the Elvin4 security mechanism, its limitations and their possible solutions, is described in a forthcoming paper.

6.2.5. Quenching

Following our experience with quenching in Elvin3, it was obvious that this was an area requiring significant work. There were two basic problems with the existing mechanism

- each quench update contained the entire subscription database
- quench information was supplied as a raw string

To address the first issue, Elvin4 allows clients to specify a filter over the subscription database, constraining quench updates to those subscriptions capable of matching messages produced by the client. The filter is expressed as a list of message attribute names that must be present in the subscription. Clients may register, modify or remove quench filters in a similar way to subscriptions.

Updates to the subscription database cause quench packets to be sent to clients with matching filters. These updates describe additions, modifications and removals from the server's subscription database. The use of quench filters, and the notification of relevant changes in the subscription base, rather than delivering a complete copy, both serve to vastly reduce the processor and network overhead of using quenching.

In addition, the change to sending updated fragments of the subscription database has included the use of an abstract syntax tree format rather than a raw string. This saves the client from requiring a parser, and allows the server to simply forward the relevant portions of its internal state to the quenching client.

The implementation of quenching requires seven additional packets, and a substantial increase in the complexity of marshalling to support the abstract syntax tree components. To support lightweight clients such as handheld or embedded devices quenching is an optional feature in Elvin4.

As an extension of the mechanism, it is possible to automate the quenching process by gathering the attribute names from emitted messages and building a suitable quench filter. The client library can then discard messages for which there are known to be no subscribers without additional code in the application. This *auto-quench* facility can be enabled via a single library call.

6.2.6. Automatic Server discovery

Configuring clients to connect to an appropriate server was one of the major administrative issues with Elvin3 deployment. In an attempt to overcome this, Elvin4 clients may use a multicast query to locate a server. Multiple servers can be provisioned within a multicast domain, with different protocol stacks and configured *scopes*. Unconfigured clients will connect to a compatible server configured with the default scope, while more sophisticated clients may specify particular servers by using non-default scope names or a direct URL. Scopes are not especially useful with a single server, but are intended to provide a mechanism for redundancy and automating failover between clustered servers in a future release.

6.2.7. Scalability

The Elvin3 server demonstrated that content-based addressing could be fast. However, its architecture was limited to supporting a few thousand clients, and fewer on platforms without a lightweight thread model.

Elvin4 remains an extremely fast content-based router, and has removed the dependency on lightweight threads, however there are application domains for which supporting tens or a hundred of thousand clients is required. In these cases, it is necessary to farm out subscription evaluation to multiple servers in a tightly-coupled local area federation. Elvin4 contains a mechanism for handover of client connections to facilitate load sharing in such an environment

The remaining research challenge is to address scalability to wide-area networks, and to provide an internet-scaled Elvin service.

7. Ideology

Elvin started as a lightweight notification service but is now viewed by its developers as a content-based routing service. This change in philosophy for what is essentially the same service is two-fold.

Firstly, our work on scalability has changed our view of Elvin from being a server to that of a service. Connections are now made not to a particular server but to the service itself. Within the DSTC environment, multiple sites are seamlessly connected together using wide-area links so that we can use the service at multiple locations as though it were a single entity.

Secondly, this transition in philosophy is due to thinking about the addressing model of notification services. Traditional communication paradigms rely on source routed messaging - that is that the sender of the information specifies where the information is to be delivered to. Source routing is the traditional model for RPC and even in multicast communication the sender is specifying the address. The inherent limitations on source addressed messaging have generally been mitigated by the use of indirection (e.g name->address resolution, or trader services).

By contrast, Elvin messages are routed not by the sender (which simply emits an unaddressed, structured message) but instead by the recipient. It is the consumer's subscription expression that defines and changes the routing of the messages and hence Elvin is best viewed as a Content Based Routing (CBR) service.

8. Future Work

We have described the interaction protocol between the Elvin4 router daemon and the client libraries. Currently, this is effectively limited to a LAN environment by both performance and protocol design. Our next step is to work towards wide-area scalability continuing towards an Internet-scaled content-based routing infrastructure.

Additionally, we have many more concrete implementations of the abstract protocol to implement and if Elvin is to be truly ubiquitous we need to extend the number of language bindings beyond C, Java, Perl, Python, and Emacs LISP.

Much work remains to make server configuration more possible and easier. Additionally it is necessary to be easily able to manage and control local area federations and the wide-area links between administrative boundaries. Another area requiring attention is key management if the Elvin security model is going to be readily accessible to users.

Finally, there are many more interesting clients to be written, particularly now that usable quenching is available.

Availability

Elvin is available in both source and binary form under a not-for-commercial-use license. Full documentation, FAQs, additional software and the download itself can be found on the Elvin homepage

http://elvin.dstc.edu.au/

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 and the United States Defense Advanced Research Projects Agency under grants F30602-96-2-0264 and F30603-94-C-0161 (both administered by the US Air Force through Rome Laboratories). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

References

ASBBLK99.
David Arnold, Bill Segall, Julian Boot, Andy Bond, Melfyn Lloyd, and Simon Kaplan, "Discourse with disposable computers: How and why you'll talk to your tomatoes," pp. 9-21 in *Proceedings of the Workshop on Embedded Systems (ES99)*, USENIX, Cambridge, Massachusetts, USA (March 1999).

ASSAC99.
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).

BCMNSS99.
Guruduth Banavar, Tuschar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E Strom, and Daniel C Sturman, "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems," *Proceedings International Conference on Distributed Computing Systems*, (1999).

BKSSST99.
Guruduth Banavar, Marc Kaplan, Kelly Shaw, Robert E Strom, Daniel C Sturman, and Wei Tao, "Information Flow Based Event Distribution Middleware," *Proceedings Middleware Workshop at the Internation Conference on Distributed Computing Systems*, (1999).

Breeze.
Michael Lawley, *Breeze: workflow with ease*, http://www.dstc.edu.au/Research/Projects/Krow-Wolf/breeze.html 1999.

CNS99.
Object Management Group, "Notification Service: Joint Revised Submission," OMG TC Document telecom/99-07-01 (July 1999).

FPSK98.
Geraldine Fitzpatrick, Sara Parsowith, Bill Segall, and Simon Kaplan, "Tickertape: Notification and Communication in a Single Line," in *Proceedings Asia Pacific Computer Human Interaction*, , Shonan Village Center, Hayama-machi, Kanagawa, Japan (15-17 July 1998).

IBM-MQ.
MQ-Series, http://www.ibm.com/software/ts/mqseries/ 2000.

Low97.
Colin Low, "Integrating Communication Services," *IEEE Communications* **35**(June 1997).

Mil99.
Chris Miles, "Monitoring, Messaging and Notification," *Proceedings 7th SAGE-AU Annual Conference*, (5-9 July 1999).

NT94.
B Clifford Neuman and Theodore Ts'o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications* **32**(9) pp. 33-38 (September 1994).

OMG95.
Object Management Group, "Common Object Services Specification," OMG TC Document 95-3-31 (March 1995).

OPSS93.
Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen, "The Information Bus: an architecture for extensible distributed systems," *ACM SIGOPS Operating Systems Review* **27**(5) pp. 58-68 (December 1993).

PFKS98.
Sara Parsowith, Geraldine Fitzpatrick, Simon Kaplan, and Bill Segall, "Tickertape: Awareness in a Single Line," in *Proceedings SIGCHI Conference on Human Factors in Computing Systems*, ACM, Los Angeles, CA (18-23 April 1998).

RDR98.
Ramduny, Dix, and Tom Rodden, "Exploring the Design Space for Notification Servers," *Proceedings*

Computer Supported Cooperative Work (CSCW'98), pp. 227-235 ACM, (1998).

RFC1832.
R. Srinivasan, "RFC1832 - XDR: External Data Representation Standard," *IETF Network Working Group*, (August 1995).

SA97.
Bill Segall and David Arnold, "Elvin has left the building: A publish/subscribe notification service with quenching," *Proceedings AUUG Technical Conference (AUUG'97)*, pp. 243-255 (September 1997).

SSL96.
Alan O Freier, Philip Karlton, and Paul C Kocher, "The SSL Protocol, Version 3.0," INTERNET-DRAFT, work in progress (18 November 1996). http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt

Sun93.
Astrid Julienne, (ed)., *The ToolTalk Service: An Inter-Operability Solution*, Prentice Hall (February 1993). ISBN 0-13-088717-X

Tal-SS.
Talarian, *Talarian SmartSockets*, http://www.talarian.com/products/smartsockets/smartsockets.pdf 1998.

TM98.
Rod Telford and Chris Miles, "EDDIE (Essential Distributed Diagnostic and Information Engine)," *Proceedings 6th SAGE-AU Annual Conference*, (6-10 July 1998).

TSS95.
Teknekron Software Systems, *Rendezvous Software Bus Programmer's Guide*. 1995.

Unicode.
The Unicode Consortium, *The Unicode Standard, Version 2.0*, Addison-Wesley Developers Press (February 1997). Second Printing

XML98.
Tim Bray, Jean Paoli, and C.M. Sperberg-McQueen, (eds), *Extensible Markup Language (XML) 1.0*, W3C Recommendation (10 February 1998). http://www.w3.org/TR/REC-xml

XmlBlaster.
Marcel Ruff, *White Paper xmlBlaster: Message Oriented Middleware (MOM)*, http://www.xmlblaster.org/xmlBlaster/doc/whitepaper/whitepaper.html 2000.

XPath.
James Clark and Steve DeRose, (eds), *XML Path Language (XPath) 1.0*, W3C Recommendation (16 November 1999). http://www.w3.org/TR/xpath