Content-Based Addressing and Routing: A General Model and its Application

Antonio Carzaniga[†] David S. Rosenblum[‡] Alexander L. Wolf[†]

[†]Dept. of Computer Science
University of Colorado
Boulder, CO 80309, USA
{carzanig,alw}@cs.colorado.edu

[‡]Dept. of Information & Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
dsr@ics.uci.edu

University of Colorado
Department of Computer Science
Technical Report CU-CS-902-00
January 2000

Abstract

The designers of communication networks are being challenged by the emergence of a new class of addressing and routing scheme referred to as *content-based addressing and routing*. This new approach differs from traditional unicast and multicast schemes in that it performs routing based on the data being transported in a message rather than on any specialized addressing and routing information attached to, or otherwise associated with, the message. An example of an application for content-based addressing and routing is an event notification service, which is a general-purpose facility for asynchronously and implicitly conveying information from generators of events to any and all parties expressing interest in those events. In order to implement content-based addressing and routing, we can adopt well-known and successful network architectures and protocols, provided that we understand how to map the core concepts and functionalities of content-based addressing and routing onto this established infrastructure. Toward that end, we have formulated a general, yet powerful model of addressing and routing that allows us to formalize the crucial aspects of content-based addressing and routing in a surprisingly simple manner. Furthermore, it allows us to treat traditional unicast and multicast addressing and routing uniformly as instances of this more general model. This paper presents our model and demonstrates its utility by showing its application to the design of an existing event notification service.

	unicast	**multicast**	**content-based**
intended destination as specified by producer	explicit	explicit	implicit
attribute of consumer used in routing	pre-assigned, unique identity	group identity	expression of interest in content
information flow	directed	directed	emergent

Table 1: A Comparison of Addressing and Routing Mechanisms.

1 Introduction

Traditional addressing and routing mechanisms in networks, both unicast and multicast, are based on the use of explicit and specialized addressing and routing information attached to, or otherwise associated with, messages. The actual data contained within messages, referred to as *content* (or sometimes *payload*), are typically invisible to the transport mechanism and, therefore, are not considered when performing addressing or routing operations.

A rather different approach is to expose the content to the network transport mechanism so that it can influence the addressing and routing of messages. In the extreme, no information other than the content is used; a network that takes this approach is said to use a *content-based* addressing and routing scheme. Under such an approach, producers will generate messages, but with no particular destinations intended. The destinations are determined by consumers expressing interest in the delivery of messages satisfying some arbitrary predicates on the content, independent of the producers of the messages. Thus, addressing and routing are rendered implicit in the combination of expressions of interest and the content of the messages flowing through the network. Moreover, complete responsiblity for determining the eventual destinations for messages is given over to the network infrastructure itself.

Table 1 presents a high-level comparison of unicast, multicast, and content-based addressing and routing mechanisms along three dimensions. First, producers of messages in unicast and multicast mechanisms provide explicit information about intended destinations, while in a content-based mechanism this information is implicit in the content itself. In fact, there may be no destination at all for a message if there is no consumer that has expressed interest in receiving the data. Second, in a unicast mechanism, the attribute of the consumer of a message is a pre-assigned, unique identity within the network. The primary contribution of the multicast mechanism is the concept of the group identity, which provides a level of indirection between message producers and consumers. Producers address messages to the group, and the messages then get routed to any and all consumers that have joined the group. A given consumer can join more than one group, which in a sense gives it multiple identities. In a content-based mechanism, consumers declare their interest in receiving messages having particular content, and it is this expression of interest that is used to route messages through the network. Of course, a given consumer can express multiple kinds of interest and therefore receive multiple kinds of messages. Third, the flow of information between a message producer and a message consumer in unicast and multicast mechanisms can be seen as the result of producers directing messages to selected consumers, given their use of explicit destination specifications and explicit identity attributes. By contrast, the flow of information under a content-based addressing and routing mechanism implicitly emerges from the circumstantial interplay between expressions of interest and any messages that are generated.

One can easily conceive of applications that could take good advantage of content-based addressing and routing. A prominent example is an *event notification service*, which is a general-purpose facility for asynchronous distribution of information in a network. Examples of such systems include Elvin [10], Gryphon [2], iBus [6], JEDI [4], Keryx [14], Siena [3], TIB/Rendezvous [13], as well as implementations of specifications such as the CORBA Event Service [8], the CORBA Notification Service [9], and the Java Distributed Event Specification [12]. Scalable versions of the service are typically implemented as a network of servers that provides clients with *access points* offering a publish/subscribe interface. The clients are of two kinds: *objects of interest*, which are the producers of notifications, and *interested parties*, which are the consumers of notifications; of course, a client can act as both an object of interest and an interested

party. Clients use the access points of their local servers to *advertise* the information about notifications that they generate and to *publish* the advertised notifications. Clients also use the access points to *subscribe* for individual notifications or compound patterns of notifications of interest. In its most generic form, a notification can be viewed as a set of attributes, where each attribute can have a name, a type, and a value. The service is responsible for *selecting* the notifications whose content are of interest to clients and *delivering* those notifications to the clients via the access points.

Event services peform notification selection using two kinds of content selection mechanisms: *filters* and *patterns*. A filter selects event notifications by specifying a set of attributes and constraints on the values of those attributes. For example, a client of the service might be interested in receiving notifications about significant stock price changes for a particular company, and so issues a subscription that includes a filter specifying the company name and a price differential. While a filter is matched against a single notification based on the notification's attribute data, a pattern is matched against two or more notifications based on both their attribute data and on the combination they form. At its most generic, a pattern might correlate events according to any compound relation. For example, a client might be interested in receiving price change notifications for the stock of one company only if the price of a related stock has changed by a certain amount. Rich languages and logics exist that allow one to express event patterns [7].

There are three major variants of an event notification service. In a *channel-based* service (e.g., iBus, the CORBA Event Service, and the Java Distributed Event Specification), notifications are fed into what amounts to a discrete communications pipe. Subscriptions are made by simply identifying the pipe (i.e., channel) from which notifications are expected to flow. Filtering then reduces to channel selection, which for purposes of routing can be treated as a simple equality test on a distinguished field of the content that represents the channel identifier (analogous to a group identifier in multicast routing). In a *subject-based* service (e.g., JEDI and TIB/Rendezvous), a distingushed field of the content, called the "subject", is also assumed to be present, but filtering is performed through more powerful predicates than the simple equality test of channel-based services.[1] The third variant is the *content-based* service (e.g., the CORBA Notification Service, Elvin, Gryphon, Keryx, and Siena), so named because filtering can be performed across the entire content of the notification.

Content-based addressing and routing, while eminently useful, pose significant challenges to the designer of a communciations network.

- Destination addresses are no longer explicit, pre-assigned identifiers (e.g., IP addresses), but instead are dynamically induced by predicates expressed over hypothetical message content.
- Routing tables are maps from predicates to destination addresses of neighbors in the network, rather than from explicit recipient destination addresses to neighbor destination addresses.
- Routing involves applying the predicates found in routing tables to the content of incoming messages and then forwarding messages satisfying the predicates toward the appropriate neighbors in the network.
- Subnet (or hierarchical) routing, which is intended to achieve autonomy and scalability in a communications network, involves the computation of inclusion relationships among predicates. Yet the subnets induced in this way may bear no relationship to the hierarchical structure of the underlying transport network.

These complications require a new approach to the design of addressing and routing mechanisms. We describe such an approach in this paper, which is based on the definition of a general, yet powerful model of addressing and routing. The model allows us to formalize the crucial aspects of content-based addressing and routing in a surprisingly simple manner. Furthermore, it allows us to treat traditional unicast and multicast addressing and routing uniformly as instances of this more general model. A primary benefit of the model is that it leads to a principled method for mapping the core concepts and functionalities of generic

[1] Note that although we treat both channel- and subject-based services as targeting distinguished fields within notifications (a channel identifier and a subject, respectively), in practice, the information they contain can be implemented independently of notifications, but this does not affect our general characterization of them.

addressing and routing schemes onto the established infrastructure of successful network architectures and protocols.

The next section of the paper presents our model. In order to make the presentation concrete, we describe the model in terms of a generic content-based event notification service. We then demonstrate the utility of the model by studying its use in guiding the implementation of an existing content-based addressing and routing mechanism. Related work is discussed in Section 4, and Section 5 concludes the paper.

2 A Model for Content-Based Addressing and Routing

The notion of content-based addressing and routing is generally applicable to a wide range of applications. In this section we use event notification as a representative application domain. As described in Section 1, an event notification service typically realizes a publish/subscribe protocol for its clients. In particular, some clients publish notifications about events that have occurred, and other (but possibly the same) clients subscribe for notifications of interest. In an event notification service then, a notification is the elementary data packet, and it is implicitly *addressed* to all subscribers whose subscriptions *match* the content of the notification. Because there may be multiple such subscribers, the event notification service is thus a multicast service. Of course, its use of content-based addressing and routing makes it fundamentally different from traditional multicast services.

In order to achieve scalability in a wide-area network, the event notification service by necessity must be implemented as a distributed network of servers. It is the responsibility of the event notification service to *route* each notification through the network of servers to all subscribers that registered matching subscriptions, and to keep track of the identity of the subscriber that registered each subscription.

From this informal description of event notification, we can identify a number of important properties. The fact that notifications are delivered based on their content rather than on an explicit destination address adds a level of indirection that provides a great deal of flexibility and expressive power to clients of the service, while also complicating the implementation of the service. Furthermore, the behavior of subscribers is dynamic, since they add and remove subscriptions. Yet this dynamism is transparent to publishers of notifications, since they simply generate content irrespective of its eventual recipients.

As discussed in Section 1, a subscription acts as a *filter* on notifications and is typically expressed as predicates on notification content. But for the purposes of presenting our general model of content-based addressing and routing, we may ignore the details of the notification data content (its type, its structure, etc.), and we may ignore the details of the language used to subscribe for notifications (its expressive power, its predicate set, its pattern operators, etc.). As we discuss in Section 3.7, the design and complexity of the routing algorithms used to implement a particular data content and subscription language design depend quite heavily on such details. But what is important at this point is defining abstractly how one notification relates to another, how subscriptions match notifications, and how these relations influence content-based addressing and routing.

2.1 Content-Based Addresses

Let $\mathcal{N}$ represent the set of all notifications, and let $\mathcal{S}$ represent the set of all subscriptions. Let n represent a particular notification in $\mathcal{N}$. A subscription $s \in \mathcal{S}$ defines a set of matching notifications $\mathcal{N}_s$, such that $\mathcal{N}_s \subseteq \mathcal{N}$. We say that s *matches* or *covers* n (written $s \sqsubset_S^N n$) iff $n \in \mathcal{N}_s$. Note that the determination of whether s matches n is dependent on the particular data model chosen for notifications and the particular language used for registering subscriptions. The subscriber C who registers s is an implicit addressee of n. If C unregisters s, then C is no longer an implicit addressee of n; in Section 2.4.3 we discuss other implications of unregistering.

In addition to relating subscriptions to notifications, we can extend the notion of covering to relate subscriptions to each other. In particular, we say that a subscription s_1 *covers* a subscription s_2 (written $s_1 \sqsubset_S^S s_2$) iff $\mathcal{N}_{s_1} \supseteq \mathcal{N}_{s_2}$. As shown below, covering relationships between subscriptions can be exploited for purposes of content-based routing.

Because a subscription s is associated with the identity of the eventual recipient of each notification n that s matches, s is a *content-based address* of n. More abstractly, we can treat any set of notifications (i.e., any

subset of $\mathcal{N}$) as a content-based address, and in general there are multiple content-based addresses induced by a particular notification n—they are simply all the notification sets (i.e., subsets of $\mathcal{N}$) that contain n. A subscription then is one particular representation of a content-based address; we discuss other possible representations in Section 2.4. For this more abstract treatment of content-based addresses, we can use the notation $x \sqsubset n$ to indicate that a content-based address x covers a notification n. Additionally, in the remainder of this section we drop the superscript and subscript from the operators for the more specific covering relations since they should be clear from the context.

2.2 Routing Tables for Content-Based Routing

A routing table is a *map* $x \mapsto H$ (read x *maps to* H) that maps an address x to a set H of next-hop addresses (with the cardinality of H being one when routing within a unicast service). A next-hop address is the address of the next server on the path toward the address being mapped. More formally, a routing table R_σ for a server σ can be defined as follows:

$$R_\sigma \stackrel{\text{def}}{=} \{x \mapsto H \mid x \in \textit{Addresses} \text{ and } H \subseteq I_\sigma\}$$

where *Addresses* is the universe of routable addresses and I_σ is the set of identities of neighboring servers and local clients of σ. Note that I_σ is a subset of some universe of explicit destination addresses *Dest-Addresses*, and that in traditional routing typically *Addresses* $\equiv$ *Dest-Addresses*. Below we drop the server identity σ to simplify the notation.

In the context of content-based addressing, a routing table has exactly the same function. In its simplest form it maps content-based addresses (such as subscriptions) into a set of explicit addresses of neighbor servers and explicit identities of local subscribers. Using the routing table to route a notification n at a server becomes a two-step process. First, the map is searched to find all mapped subscriptions s that cover n. Second, n is routed to all next-hop addresses mapped by all such s. More formally, let X_n be the set of all addresses in the domain of R that cover n:

$$X_n \stackrel{\text{def}}{=} \{x \mid x \in \textit{dom}(R) \text{ and } x \sqsubset n\}$$

Then n is routed to all servers in H_n, where

$$H_n \stackrel{\text{def}}{=} \bigcup_{x \mapsto H \text{ and } x \in X_n} H$$

2.3 Subnets Induced by Content-Based Routing

Routing tables are central to any routing algorithm, and hence they must have compact representations and must support fast lookup and update operations. The cardinality of the map in a routing table equals the cardinality of the address space, but it is not necessary to materialize every single mapping in the routing table. Instead, the size of the table can be significantly reduced by combining addresses into equivalence classes based on their mapped values.

In practice, if

1. two addresses x_1 and x_2 map to the same next-hop set H, and
2. there exists an address $\overline{x}$ that represents both x_1 and x_2,

then the routing table need only store the mapping $\overline{x} \mapsto H$ instead of the two mappings $x_1 \mapsto H$ and $x_2 \mapsto H$. This simple principle is the generalization of the well-known method of *subnet routing* employed as a basis for scaling traditional routing tables to networks of millions of nodes, and in this case $\overline{x}$ is the *subnet address*. In the case of IP (unicast) routing, the first condition frequently holds because addresses have a hierarchical distribution that maps onto a semi-hierarchical topology of the network. The second condition is realized by the use of a subnet address plus a subnet mask.

In an event notification service, two content-based addresses may partially or even completely overlap in the sense that there is a non-empty intersection between the sets of notifications they define. Therefore, the idea of subnet routing can be adapted to content-based routing. Using subscriptions for purposes of illustration, in the special case in which

1. two content-based addresses s_1 and s_2 map into the same next-hop set H, and

2. $s_1 \sqsubset s_2$,

the routing table need only store the mapping $s_1 \mapsto H$ instead of the two mappings $s_1 \mapsto H$ and $s_2 \mapsto H$. In general, if $s_1 \mapsto H_1$, $s_2 \mapsto H_2$, $s_1 \sqsubset s_2$, and $H_1 \supseteq H_2$, then the routing table need only store $s_1 \mapsto H_1$. We specifically refer to this form of subnet induced by the covering relation between subscriptions as a *content-based subnet*.

2.4 Additional Considerations for Event Notification Services

There are a number of embellishments to the basic publish/subscribe protocol that can be found in event notification services. These include the capability for clients to subscribe for *patterns* of multiple notifications; the capability for clients to *advertise* the notifications they intend to publish; and the capability for clients to *cancel* previous subscriptions and advertisements. We discuss how these embellishments impact the model of content-based addressing and routing described above.

2.4.1 Subscriptions for Patterns of Multiple Notifications

A subscription for a pattern of multiple notifications comprises a set of simple subscriptions (each of which matches a set of notifications as before) formed into a compound expression through the use of composition operators. Such a *compound subscription* itself defines a set of matching notifications, and hence a compound subscription is another possible representation for a content-based address in an event notification service. Furthermore, compound subscriptions would populate the same routing tables as simple subscriptions. The decision of which operators to make available for construction of patterns is one of the many decisions to be made in designing a subscription language, but the increasing ubiquity of pattern operators in event notification services, and their potential impact on content-based routing, merits giving them a closer look.

Each composition operator for a compound subscription induces a corresponding set-theoretic construction over the sets of notifications defined by the subscriptions the operator composes. For instance, the subscription "s_1 **or** s_2" is typically interpreted as matching any single notification n such that either $s_1 \sqsubset n$ or $s_2 \sqsubset n$. In other words, the subscription matches any single n such that $n \in \mathcal{N}_{s_1} \cup \mathcal{N}_{s_2}$.

Other kinds of operators may require generalizing the definition of the universe $\mathcal{N}$ of notifications to a multiset, or to a time-ordered set. Let n^t represent the generation of a notification n at time t. We can, for instance, refer to n^{t_1} and n^{t_2} as occurrences of the same notification content n being generated at times t_1 and t_2, and we can then refer to relationships between the time values t_1 and t_2. This allows the definition of a sequence operator as in "s_1 **then** s_2", which is typically interpreted as matching any pair of notifications $n_1^{t_1}$ and $n_2^{t_2}$ such that $s_1 \sqsubset n_1$, $s_2 \sqsubset n_2$, and $t_1 < t_2$.

The temporal element allows some flexibility in the definition of the operators as well. For instance, the subscription "s_1 **and** s_2" is typically interpreted as matching any pair of notifications $n_1^{t_1}$ and $n_2^{t_2}$ such that $s_1 \sqsubset n_1$ and $s_2 \sqsubset n_2$. Since there may be a non-empty intersection between $\mathcal{N}_{s_1}$ and $\mathcal{N}_{s_2}$, it may be possible for s_1 and s_2 to match the same notification. Under one interpretation, it may be allowed that $n_1 = n_2$ and $t_1 = t_2$—that is, both sides of the compound subscription are matched by the same exact notification. Under another interpretation, it maybe required that if $n_1 = n_2$, then $t_1 \neq t_2$—that is, even if the content is the same, then two different, temporally-separated occurrences of that content must be matched.

Given this definition of compound subscription, the routing of notifications to their matching compound subscriptions can be treated in a variety of ways. For instance, the compound subscription can be treated simply as representing a content-based address whose set of notifications is the union of all the notification sets defined by the constituent simple subscriptions, irrespective of the operator used to compose the simple subscriptions. In this case, final matching of the specified pattern must be performed locally by the subscriber. Alternatively, the compound subscription could be *factored*, with routing table entries defined in such a way that the individual notifications are matched as close as possible to their publishers and then opportunistically aggregated with other matching notifications as soon as possible along the path to the subscriber. We discuss this further in Section 3 in the context of a particular realization of an event notification service.

2.4.2 Advertisements

An *advertisement* a can be used to advertise a capability or intent to publish a particular class of notifications. The notifications that are eventually published belong to the set of notifications $\mathcal{N}_a$ defined by the advertisement. Hence, an advertisement is another representation of a content-based address in an event notification service. Just as subscriptions define content-based addresses of notifications, advertisements define content-based addresses of subscriptions, in the sense that a subscription need be routed only along paths toward advertisements that cover the subscription. The notifications that are eventually published are then routed in reverse back to the subscriber.

Because they serve a different purpose from subscriptions, advertisements would be routed with their own routing tables separate from the routing tables for subscriptions. There are a number of possible choices for the definition of the covering relation between advertisements and subscriptions, and we describe one particular definition in detail in Section 3.

2.4.3 Unsubscriptions and Unadvertisements

Unsubscriptions and *unadvertisements* serve to cancel previous subscriptions and advertisements, respectively. As is the case with a subscription (advertisement), an unsubscription (unadvertisement) defines an associated set of notifications, it can be related with other subscriptions (advertisements) according to a covering relation, and hence it is another possible representation for a content-based address in an event notification service. However, note that the effect of unsubscriptions (unadvertisements) is to remove entries from the routing tables for subscriptions (advertisements) rather than to populate those tables.

There are a number of possible choices for the definitions of such covering relations, all of which have the interpretation that an unsubscription (unadvertisement) cancels all subscriptions (advertisements) that it covers. We describe particular definitions of the covering relations in detail in Section 3.

3 An Example Realization of Content-Based Addressing and Routing

In this section we apply the general model discussed in Section 2 to the particular realization of a content-based routing infrastructure found in the Siena event notification service [3].

3.1 Notifications and Filters

A notification is set of attributes, where attribute has a name a type, and a value. Attribute names are simply character strings. The attribute types belong to a predefined set of primitive types commonly found in programming languages and database query languages, and for which a fixed set of operators is defined.

An *event filter*, or simply a *filter*, selects event notifications by specifying a set of attributes and constraints on the values of those attributes. Each attribute constraint is a tuple specifying a type, a name, a binary predicate operator, and a value for an attribute. The operators provided by Siena include all the common equality and ordering relations ($=, \neq, <, >$, etc.) for each of its types, substring ($*$), prefix ($>*$), and suffix ($*<$) operators for strings, and an operator *any* that matches any value.

An attribute $\alpha = (type_\alpha, name_\alpha, value_\alpha)$ matches an attribute constraint $\phi = (type_\phi, name_\phi, operator_\phi, value_\phi)$ iff $type_\alpha = type_\phi \wedge name_\alpha = name_\phi \wedge operator_\phi(value_\alpha, value_\phi)$. We say an attribute α satisfies or *matches* an attribute constraint ϕ with the notation $\phi \sqsubset_f^n \alpha$. When α matches ϕ, we also say that ϕ *covers* α. Figure 1 shows a filter that matches price decreases for stock DIS on stock exchange NYSE.

string	*class*	$>*$	*finance/exchanges/*
string	*exchange*	$=$	*NYSE*
string	*symbol*	$=$	*DIS*
float	*change*	$<$	*0*

Figure 1: Example of an Event Filter.

When a filter is used in a subscription, multiple constraints for the same attribute are interpreted as a conjunction—all such constraints must be matched. Thus, we say that a notification n *matches* a filter f, or equivalently that f *covers* n ($f \sqsubset_S^N n$ for short):

$$f \sqsubset_S^N n \Leftrightarrow \forall \phi \in f : \exists \alpha \in n : \phi \sqsubset_f^n \alpha$$

Notice that the notification may contain other attributes that have no correspondents in the filter. Table 2 gives some examples that illustrate the semantics of $\sqsubset_S^N$. The second example is not a match because the

subscription		*notification*
string what = alarm	$\sqsubset_S^N$	*string what = alarm* *time date = 02:40:03*
string what = alarm *integer level > 3*	$\not\sqsubset_S^N$	*string what = alarm* *time date = 02:40:03*
string what = alarm *integer level > 3* *integer level < 7*	$\not\sqsubset_S^N$	*string what = alarm* *integer level = 10*
string what = alarm *integer level > 3* *integer level < 7*	$\sqsubset_S^N$	*string what = alarm* *integer level = 5*

Table 2: Examples of $\sqsubset_S^N$.

notification is missing a value for attribute *level*. The third example is not a match because the constraints specified for attribute *level* in the subscription are not matched by the value for *level* in the notification.

3.2 Advertisements

We have shown how the covering relation $\sqsubset_S^N$ defines the semantics of filters in subscriptions. We now define the semantics of advertisements with a similar relation $\sqsubset_A^N$. Recall that the motivation for introducing advertisements in an event notification service is to identify the potential sources of notifications so that the service can employ an efficient strategy in setting up routing paths for those notifications.

The relation $\sqsubset_A^N$ defines the set of notifications covered by an advertisement as follows:

$$a \sqsubset_A^N n \Leftrightarrow \forall \alpha_n \in n : \exists \phi_a \in a : \phi_a \sqsubset_f^n \alpha_n$$

This expression says that an advertisement covers a notification if and only if it covers each individual attribute in the notification. Note that this is the dual of subscriptions, which define the minimal set of attributes that a notification must contain. In contrast to subscriptions, when a filter is used as an advertisement, then multiple constraints for the same attribute are interpreted as a disjunction rather than as a conjunction; only one of the constraints need be satisfied. Table 3 shows some examples of the relation $\sqsubset_A^N$. The second example is not a match because the attribute *date* of the notification is not defined in the advertisement. The fourth example is not a match because the value of attribute *what* in the notification does not match any of the constraints defined for *what* in the advertisement.

3.3 Subnet Covering Relation

So far we have defined the covering relations that define content-based addresses in Siena, in particular, $\sqsubset_S^N$ (semantics of subscriptions) defines *destination* addresses, whereas $\sqsubset_A^N$ (semantics of advertisements) defines *source* addresses. From these, following the definitions of Section 2, we derive the covering relations that define Siena's content-based subnet addresses:

advertisement		*notification*
string what = alarm *string what = login* *string user any*	$\sqsubset_A^N$	*string what = alarm*
string what = alarm *string what = login* *string user any*	$\not\sqsubset_A^N$	*string what = alarm* *time date = 02:40:03*
string what = alarm *string what = login* *string user any*	$\sqsubset_A^N$	*string what = login* *string user = ingmar*
string what = alarm *string what = login* *string user any*	$\not\sqsubset_A^N$	*string what = logout* *string user = ingmar*

Table 3: Examples of $\sqsubset_A^N$.

- $f_1 \sqsubset_S^S f_2$: subscription filter f_1 covers subscription filter f_2. Formally,

$$f_1 \sqsubset_S^S f_2 \Leftrightarrow \forall n : f_2 \sqsubset_S^N n \Rightarrow f_1 \sqsubset_S^N n$$

  which means that f_2 defines a subnet destination address of f_2.

- $a_1 \sqsubset_A^A a_2$: advertisement filter a_1 covers advertisement filter a_2 (where a_1 and a_2 are interpreted as advertisements). Formally:

$$a_1 \sqsubset_A^A a_2 \Leftrightarrow \forall n : a_2 \sqsubset_A^N n \Rightarrow a_1 \sqsubset_A^N n$$

  which means that a_2 defines a subnet source of a_1.

3.4 Routing Tables: The Filters Poset

The representation of a content-based routing table in Siena is a partially ordered set of filters (a *filters poset*) The partial order is defined by the covering relations $\sqsubset_S^S$ for subscription filters, and $\sqsubset_A^A$ for advertisement filters. Let P_S be a poset defined by $\sqsubset_S^S$, and P_A a poset defined by $\sqsubset_A^A$. Figure 2 shows an example of a

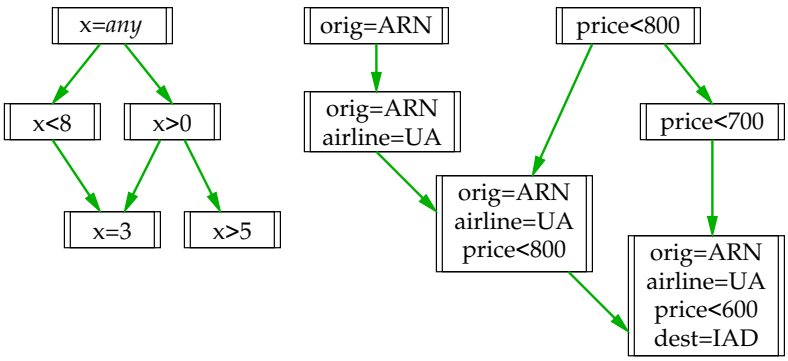

Figure 2: Example of a Poset of Simple Subscriptions. Arrows Represent the *immediate* Relation $\sqsubset_S^S$.

poset of subscriptions.

Note that $\sqsubset_S^S$ and $\sqsubset_A^A$ are transitive relations; however the diagram and its representation in memory store *immediate* relations only. In a poset P_S, ordered according to $\sqsubset_S^S$, a filter g is an *immediate successor* of

another filter f and f is an *immediate predecessor* of g iff $f \sqsubset_S^S g$ and there is no other filter x in P_S such that $f \sqsubset_S^S x \sqsubset_S^S g$. Filters that have no immediate predecessors are called *root* filters.

As shown below, only *root* filters cause network traffic, due to the propagation of subscriptions (or advertisements). Thus the "shape" of a poset roughly indicates the effectiveness of the routing strategies used. In particular, a poset that extends vertically indicates that content-based addresses are very much interdependent and that there are just a few subscriptions summarizing all the other ones. Conversely, a poset that extends horizontally indicates that there are a few overlapping addresses. Note that the degenerate case of a flat poset corresponds to the case of systems like IP multicast in which subscriptions (IP group addresses) define completely disjoint sets.

3.5 Routing Algorithm

In discussing the main principles underlying the content-based routing information protocol, we limit the discussion the propagation of subscriptions; advertisements are treated symmetrically with respect to subscriptions, so the same concepts apply to advertisements as well.

In order to keep track of the propagation of subscriptions, every router augments the basic routing table by associating with each subscription s a set of subscribers *subscribers*(s) and a set of *neighbors* routers to which s has been forwarded *forwards*(s).

When a router receives a subscription **subscribe**(U, f), from a client or a neighbor server (U), it looks in its subscriptions poset for either

1. a subscription f' that covers f and has U among its *subscribers*: $f' \sqsubset_S^S f \wedge U \in$ *subscribers*(f'). In this case, the procedure that handles the subscription returns with no effect; or
2. a subscription f' that is equal to f and does not contain U in its *subscribers*: $f' \sqsubset_S^S f \wedge f \sqsubset_S^S f'$. Here the server adds U to *subscribers*(f'); or
3. two possibly empty sets $\overline{f}$ and $\underline{f}$, representing the immediate predecessors and the immediate successors of f respectively. Here the server inserts f as a new subscription between $\overline{f}$ and $\underline{f}$, and adds U to *subscribers*(f).

In cases 2 and 3, the server also removes U from all the subscriptions covered by f, and then removes those subscriptions that have no other subscribers.

Then, the router forwards the subscription to some of its neighbors. Formally, given a subscription f in P_S, let *forwards*(f) be defined as follows:

$$\mathit{forwards}(f) = \mathit{neighbors} - \mathit{NST}(f) - \bigcup_{f' \in P_S \wedge f' \sqsubset_S^S f} \mathit{forwards}(f') \tag{1}$$

In other words, f is forwarded to all neighbors except those not on any spanning tree rooted at an original subscriber of f (*NST*(f)), and those to which subscriptions f' covering f have been forwarded already by this server (the last term in the formula).

The last term in the formula represents the optimization that the server can make in the situation where more generic subscriptions have been propagated already to some neighbors. In terms of our general model, this means exploiting the fact that the new address represents a subnet of a previously defined address.

Figure 3 shows a fragment of an event service. In this example, server 1 is connected to servers 2, 3, and 4. Server 1 also has a local client a. Server 3 sends a subscription [airline=*any*] to server 1. The subscription poset shown on the right side of the figure represents the routing table of server 1. As shown in the figure, the new subscription is inserted as a *root* subscription in and then forwarded to servers 2 and 4 but not to server 3, which is in the *NST* set of the subscription. In this figure and the following ones, for each subscription in P_S, *subscribers* are denoted with an outgoing arrow while *forwards* are denoted with an incoming arrow. Intuitively, arrows indicate the direction of notifications.

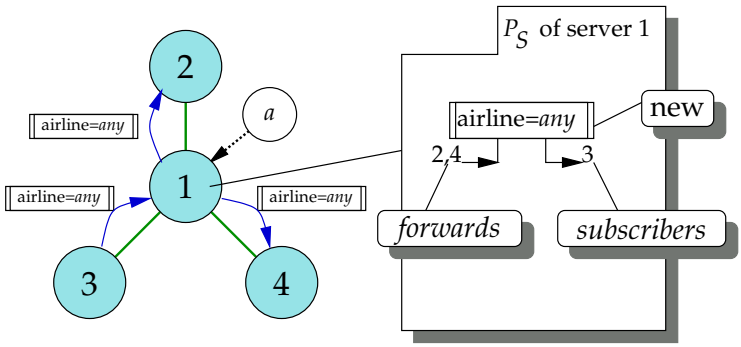

Figure 3: Example of How a Router Processes a Subscription (Step 1).

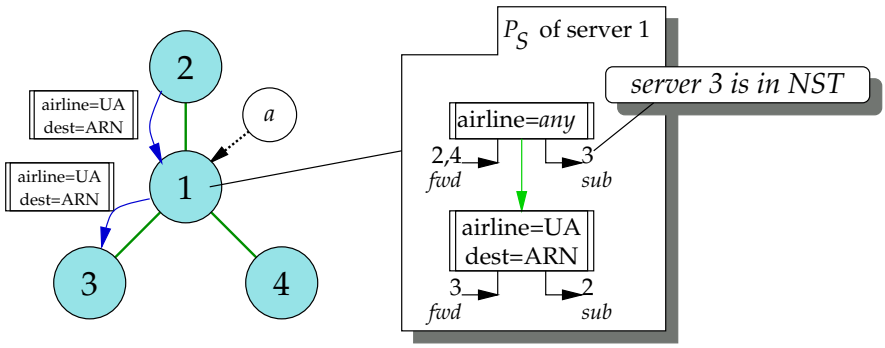

Figure 4: Example of How a Peer-to-Peer Server Processes a Subscription (Step 2).

Figure 4 shows the effect of a second subscription [airline=UA, dest=ARN] sent to server 1 by server 2. This subscription is inserted as an immediate successor of the previous (*root*) subscription and is forwarded to server 3, which is the only neighbor that is not in the *forwards* set for the previous subscription [airline=*any*].

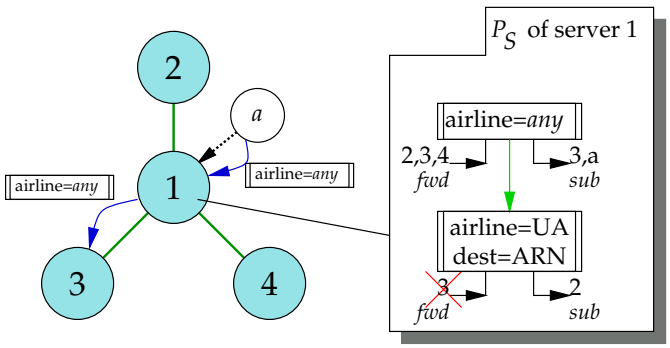

Figure 5: Example of How a Peer-to-Peer Server Processes a Subscription (Step 3).

In Figure 5 client a subscribes for [airline=*any*]. In this case, the subscription is found in P_S and a is simply added to its *subscribers* set. As the *NST* set for that subscription is now empty, so the subscription is then forwarded to server 3. Every time the server forwards a subscription f to a neighbor server Y, it adds Y to the *forwards* set of f and consequently removes Y from the *forwards* sets of all the subscriptions covered by f. In the example, the server removes 3 from the *forwards* sets of subscription [airline=UA, dest=ARN].

3.6 Matching Patterns

Siena provides support for the matching of patterns of notifications. This functionality is implemented with distributed monitoring in such a way that patterns are assembled as close as possible to the sources of their elementary pieces. In practice, servers assemble sequences of notifications from smaller sub-sequences or from single notifications according to the paths along which notifications will be provided. Such paths are determined by advertisements, which is why this technique requires publishers to advertise their notifications. The providers of sub-sequences of notifications are other monitors that the server has already set up for previous compound subscriptions. We will use the term *pattern factoring* to refer to the process by which the server breaks a compound subscription into smaller compound and simple subscriptions. After a subscription has been factored into its elementary components, the server attempts to group those factors into compound subscriptions to forward to some of its neighbors. We will call this process *pattern delegation*.

Every server has a table of advertisements T_A, maintained by means of the algorithm of Section 3.5. This table stores available classes of notifications and sub-patterns. Each element p has an associated set *providers*(p) listing the peer servers from which p has been advertised.

When a server X receives a compound subscription **subscribe**(U, p) where $p = f_1 \cdot f_2 \cdot \ldots \cdot f_k$, it scans p trying to match each f_i with a pattern a_i, or trying to match a sequence $f_i \cdot f_{i+1} \cdot \ldots \cdot f_{i+k_i}$ with a compound pattern $a_{i \ldots i+k_i}$ using its available patterns.

Once a compound subscription is divided into available parts, the server must

- send out the necessary subscriptions in order to collect the required sub-patterns; and
- set up a *monitor* that will receive all the notifications matching the sub-patterns and will observe and notify the occurrence of the whole pattern

In deciding which subscriptions to send out, the server tries to re-assemble the elementary factors in groups that can be delegated to other servers, thereby pushing the recognition of patterns as close as possible to its sources. The selection of sub-patterns that are eligible for delegation follows some intuitive criteria. For example, only contiguous sub-patterns available from a single source can be grouped and delegated to that source.

Space limitations prevent us from presenting a detailed illustration of these concepts.

3.7 Complexity of the Routing Operations

The main operations that the router executes are (1) evaluations of covering relations and (2) maintainance of filter posets, which in turns requires the evaluation of series of covering relations.

Computing $\sqsubset_S^N$ and $\sqsubset_A^N$ is relatively easy. $\sqsubset_S^N$ and $\sqsubset_A^N$ are respectively a conjunction and a disjunction of elementary constraints, each one evaluated on the corresponding attribute value in the notification. With a complexity $O(1)$ for each individual comparison,[2] the evaluation of both $\sqsubset_S^N$ and $\sqsubset_A^N$ has a total complexity of $O(m + n)$, where n is the number of constraints in the filter and m is the number of attributes in the notification.

The complexities of computing $\sqsubset_S^S$, $\sqsubset_A^A$, and $\sqsubset_A^S$ are instead $O(nm)$, where n and m represent the number of attribute constraints appearing in the respective subscription and/or advertisement filters. This complexity represents a comparison between each attribute constraint in one filter and any corresponding attribute constraints in the other filter. Note that checking a covering relation between filters amounts to a universal quantification. But given the particular types and operators in Siena, comparing a pair of attribute constraints can be reduced to evaluating an appropriate predicate on the two constant values of the constraints, with a complexity $O(1)$. For example, to see if $[\mathrm{x} > k_1]$ covers $[\mathrm{x} > k_2]$, it suffices simply to verify that $k_2 \geq k_1$.

A server typically executes two classes of operations on a filter poset (for example on a subscriptions poset): in the first case, when processing a notification, it looks for the set of all the addresses of that notification. In practice, it looks for every subscription that covers ($\sqsubset_S^N$) n. The second case is when a new address f, say a new subscription, is inserted. To do that, the router must find $\overline{f}$, the set of immediate

[2] This is true for all the predefined types of Siena except for the string type.

predecessors of f, and $\underline{f}$, the set of immediate successors of f. In the worst case (a flat poset), both kinds of operations involve a sequential scan of the routing table, so their complexity would be $O(R(n+m))$ (when applying a covering relation between a filter and a notification) or $O(Rnm)$ (for a covering relation between two filters), where R is the cardinality of the routing table. This complexity can be reduced in the presence of subnets relations in the routing table: Because of the definition of $\sqsubset_S^S$ and its relation to $\sqsubset_S^N$, elements in the poset are arranged in such a way that for every element s, all its immediate successors define subsets of the set of subscriptions defined by s. Thus, if a notification n is not covered by s, the evaluation of every subscription s' covered by s can be safely skipped.

4 Related Work

We have been unable to find any systematic treatment in the literature of content-based addressing and routing for communication networks. And we have also been unable to find any papers that uniformly relate content-based addressing and routing with traditional unicast and multicast addressing and routing.

The term "content-based routing" is used in a number of research areas with varying connotations. Some of these areas use a completely different meaning for the term, and hence they bear virtually no real relationship to the work described in this paper. Other areas are relevant because they contribute motivating applications rather than a generic solution for content-based addressing and routing. Finally, there are some research efforts that share some of the goals and the overall approach of this work.

One example of the first area is the mechanism used in clusters of Web servers for mapping HTTP requests to individual servers within a cluster [15]. The mechanism is called content-based routing because each HTTP request is directed by a gateway of some kind (e.g., a master server, a scheduler, a switch) to a particular server based on its content. Another example is that of multimedia information systems, in which the term content-based routing describes the processing, querying, and navigation of various documents based on the nature of their content [5]. In both cases the "routing" is not being performed to establish a communication path, but simply to demultiplex an incoming stream of content for specialized processing. And in both cases it is something other than the network infrastructure that implements the particular notion of content-based routing.

Among the research areas of that provide suitable application domains, we find systems that realize or make use of some sort of event service. For example, in agent-based systems, agents exchange information by means of "message boards" or "facilitators" that pass information based on subscriptions just like an event notification service [11]. Similarly, in virtual reality environments, the interaction between a pair of objects (such as two objects in the same virtual room) are modeled as events, and some form of event service is used to connect objects [14]. Obviously, those systems that provide an event notification service as their main purpose are good candidates for exploiting the general model and the techniques described in this paper. Some well-known technologies of this kind include the CORBA Notification Service [9] and the Java Distributed Event Specification [12].

Of the many event service technologies, some are indeed implemented as distributed network services [2, 3, 4, 6, 14]. Of those, some also offer an expressive data model and subscription language, not limited to channel-based or subject-based filters [2, 3]. Such systems implement a content-based addressing and routing service. The Siena system described in Section 3 is clearly one of those. Recently, Yu et al. [16] proposed a system whose model of notifications and subscriptions resemble those of Siena.

Another representative system is IBM's Gryphon. The focus of Gryphon is on the techniques used by event routers to store and match subscriptions [1]. In particular, subscriptions are modeled as conjunctions of basic predicates defined over specific attributes (much like in Siena and other related systems), and routing tables are represented by a forest of trees in which each tree node represents an attribute name and each outgoing arc from the node represents a predicate on the attribute value. The forest is populated through left-to-right scans of subscription expressions, and the forest is then used by each router to compute the set of next-hops for a notification. Two subscriptions that share a common prefix of conjuncts can be represented by the same tree in the forest. The advantage of this approach is that (1) it provides a way of reducing the size of the routing tables, and (2) a common prefix of conjunctions is evaluated only once for all subscriptions sharing that prefix. But the main disadvantage of Gryphon's routing strategy with respect to the model of Section 2 is that Gryphon does not attempt to exploit semantic similarities between

conjuncts. In particular, Gryphon's routing strategy exploits only identical prefixes of conjuncts, rather than covering relationships between different conjuncts. Therefore, Gryphon cannot apply the subnetting technique described in Section 2 to combine entries for related subscriptions. As a consequence, every router must store every subscription in its routing table.

5 Conclusion

In this paper we have described a general model of content-based addressing and routing, and we have described a realization of the model in an event notification service.

The main advantage of content-based addressing over traditional, explicit addressing is that the former allows information producers to send information over a network without regard to or knowledge of the eventual recipients of that information or the location of those recipients. And it allows potential recipients to express interest in information of interest in order to receive that information, without regard to or knowledge of the senders of that information or the location of those senders. In this respect, content-based addressing transforms the concept of an "address" from one of location to one of message content.

The general model of content-based addressing and routing presented in this paper has several noteworthy characteristics. First, while we have formulated the model in terms of event notification services, the model is applicable to other applications requiring content-based addressing and routing such as delivery of streaming content.

Second, the model is defined independently of the particular data model and language used to form and request, respectively, the routed content. Thus, even within the domain of event notification services, the model can be used to characterize and compare a range of different services that have been studied, and to compare a range of design choices for a single service.

Finally, the model characterizes in a uniform way three different forms of addressing and routing—traditional unicast addressing and routing, traditional multicast addressing and routing, and content-based addressing routing. This allows us to exploit well-understood algorithms and processing strategies employed in traditional routing and to tailor them for use with content-based routing. This also allows for the possibility of hybrid forms of addressing and routing, a subject we intend to explore in our future work.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Eighteenth ACM Symposium on Principles of Distributed Computing (PODC '99)*, Atlanta GA, USA, May 4–6 1999.

[2] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *The 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, Austin, TX USA, May 1999.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Interfaces and algorithms for a wide-area event notification service. Technical Report CU-CS-888-99, Department of Computer Science, University of Colorado, Oct. 1999.

[4] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, To appear.

[5] IEEE. *1999 IEEE Workshop on Content-Based Access of Image and Video Libraries (CBAIVL '99)*, June 1999.

[6] S. Maffeis. iBus: The Java Intranet Software Bus. Technical report, SoftWired AG, Zurich, Switzerland, Feb. 1997.

[7] M. Mansouri-Samani and M. Sloman. GEM: A generalized event monitoring language for distributed systems. *IEE/IOP/BCS Distributed Systems Engineering Journal*, 4(2):96–108, June 1997.

[8] Object Management Group. CORBAservices: Common object service specification. Technical report, Object Management Group, July 1998.

[9] Object Management Group. Notification service. Technical report, Object Management Group, Nov. 1998.

[10] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUG97*, Brisbane, Queensland, Australia, Sept. 3–5 1997.

[11] N. Skarmeas and K. L. Clark. Content-based routing as the basis for intra-agent communication. *Lecture Notes in Computer Science*, 1555:345–362, 1999.

[12] Sun Microsystems, Inc., Mountain View CA, U.S.A. *Java Distributed Event Specification*, 1998.

[13] TIBCO Inc. Rendezvous information bus.
http://www.rv.tibco.com/rvwhitepaper.html, 1996.

[14] M. Wray and R. Hawkes. Distributed virtual environments and VRML: an event-based architecture. In *Proceedings of the Seventh International WWW Conference (WWW7)*, Brisbane, Australia, 1998.

[15] C.-S. Yang and M.-Y. Luo. Efficient support for content-based routing in web server clusters. In *Second USENIX Symposium on Internet Technologies & Systems*, Boulder, CO, USA., Oct. 1999.

[16] H. Yu, D. Estrin, and R. Govindan. A hierarchical proxy architecture for Internet-scale event services. In *Proceedings of WETICE '99*, Stanford, CA, June 1999.