A Case for Message Oriented Middleware

Guruduth Banavar, Tushar Chandra, Robert Strom, and Daniel Sturman

IBM T. J. Watson Research Center, Hawthorne, New York

{banavar,tushar,strom,sturman}@watson.ibm.com

Abstract. With the emergence of the internet, independent applications are starting to be integrated with each other. This creates a need for technology for glueing together applications both within and across organizations, without having to re-engineer individual components. We propose an approach for developing this glue technology based on message flows and discuss the open research problems in realizing this approach.

1 Introduction

Recent advances in networking and the pervasive deployment of the internet have created new opportunities for computing research. Many applications are evolving from monolithic to distributed systems. Business processes are increasingly being automated and interconnected in spontaneous ways. Companies increasingly require the integration of once independent applications, either because they are vertically integrating components of the business, or because of mergers or outsourcing of function to separate organizations. In summary, there is a convergence to loosely integrated distributed systems, where each component can evolve independently.

In this paper, we make the case for research in "glue technology" for loosely integrating distributed systems. The basic observation underlying this technology is that widely disseminated, often real-time "events" (or messages) — e.g. stock quotations, advertisements, offers to buy and sell, weather reports, traffic conditions, etc. — are becoming ubiquitously available through the Internet. These public events, as well as internal events, such as orders, shipments, deliveries, manufacturing line changes, can form the "glue" to link applications within and across organizations. Since this technology is based on messages, we use the term Message Oriented Middleware, or MOM for short, to refer to it.

Consider the opportunities that arise within a single company. Today, most large companies consist of several smaller organizations that are supported by applications that were developed and have evolved largely independently of each other. In most cases, people provide the linkage between these smaller organizations. For example, when a sale occurs, the sales representative sends a request to manufacturing to produce the product. In such cases, what is needed is a technology that allows companies to automate such processes by tying together its organizations' independent applications. Once tied together, the integrated computing system is potentially cheaper to operate, faster and less prone to errors

than the system in which humans glued together independent organizations. Further, this kind of integration opens up new opportunities for macroscopic analysis such as forecasting, and requirements analysis across the company that can further streamline operations.

The value of tying together applications can easily be extended to partnerships between companies. One obvious scenario is when two companies merge. Usually each company has its own set of independent applications — the challenge is to provide "computing glue" to tie these applications together. Another obvious scenario is when two companies do business with each other. In today's model, people are heavily involved in transacting business between companies. Business integration offers the opportunity to automate business interactions between companies by tying together their applications.

As more consumers shop, invest, pay bills and taxes, read, and perform cooperative work and play online, the same glue technology may be extended to support business-to-consumer interactions. Better integration between business software and end-user software such as web browsers offers the promise of more power and greater ease of use to the end-user.

We propose an approach for developing the glue technology required for MOM. Our proposed approach is derived from the publish and subscribe model [OPSS93] and event delivery systems [WIS]. In this model, the basic unit of data is a *message*, which corresponds to what are called "events" in publish/subscribe or event delivery systems [SV97]. Clients register as publishers or subscribers of messages. Messages are sent to and delivered from *information spaces*, each of which has a predefined *message schema* [BKS+99]. Information spaces may be tied together via *message flow graphs* that specify how messages are propagated and transformed between producers and consumers. A message flow graph may route a filtered subset of messages from one information space to another, merge messages from multiple sources, or deliver a transformed version of a message from one information space to another. Some information spaces contain states summarizing the message history of other information spaces. This state can be re-mapped back into a message sequence, often in more than one way. Systems can exploit this non-determinism by relaxing the ordering of messages, by dropping obsolete messages, by compressing the past history being sent to a newly connecting subscriber or to a subscriber who has reconnected after being off-line.

In the near future, we envision a pervasive MOM environment that glues together a large number of stand-alone applications. Each application may evolve independently from the others in this environment. The MOM environment will support such evolution without requiring changes in other applications, and in fact, without requiring the other applications to be aware of the addition and removal of applications and clients. (The only exceptions would be those applications dealing with access control and security). The MOM environment will allow new applications to "tap into" information generated by existing applications without disturbing them. This will allow users to add higher order features such as auditing, monitoring, and data mining on top of existing information flows, after the fact, and without disrupting the underlying applications.

Several crucial research problems remain unsolved in the MOM approach, and even those that are solved have not been completely implemented yet. A complete and precise model for this approach has not yet emerged. Several key distributed computing problems remain open such as scalability, and how to provide end-to-end guarantees on message delivery. Some of the known algorithms tackle subsets of the overall problem. Questions related to fault-tolerance, security, message ordering and topology changes that have been well studied in the context of other types of messaging systems are open areas for further research in the context of MOM.

There are several efforts from various other communities to provide glue technology to tie together applications, and it is not clear at this stage whether a single glue technology is best suited for all environments. The database community has extended the classical ideas underlying databases to distributed environments via distributed transactions [TGGL82] and federated databases [Hsi92]. The languages community has extended the concept of objects to a distributed environment via remote method invocation (CORBA [Gro98], RMI [BN83], etc.). Group communication systems such as Isis have also been used to glue together applications in a distributed environment [BCJ+90 Bir93,Edi96]. Finally, there exist higher-level approaches such as Workflow that are targeted towards specific subsets of the overall problem [WfM,PPHC98]. We will compare our proposed MOM approach with these other approaches.

The rest of this paper is organized as follows. In section 2 we examine several examples that motivate the need for message oriented middleware. In section 3 we elaborate on the message oriented middleware model. In section 4 we discuss open areas for further research in this area. In Section 5 we examine alternative approaches. Section 5 concludes with work related to content-based publish/subscribe systems.

2 Examples

In this section, we provide two examples that motivate the need for Message Oriented Middleware and illustrate the various requirements imposed on this approach:

1. *Stock Trading.* This example of application integration demonstrates the need for MOM-based systems to be highly scalable and for on-the-fly transformation of messages into formats suitable for different clients. This example also illustrates the need for anonymity between message publishers and message subscribers.
2. *Home Shopping.* This example is a home shopping application and further demonstrates the need for cross-domain messaging. This example also illustrates the need for interpreting message streams to capture application-specific meaning.

2.1 Stock Trading

To illustrate an instance of application integration, consider a publish-subscribe based stock trading application written for a particular stock exchange, say the New York Stock Exchange (Figure 1). In such an application, stock trades, bids, and sales are published as messages. Brokers affiliated with the NYSE have access to this information, and subscribe to events of particular interest to them. Stock trade events are published in a format beginning with the following four attributes: (1) NYSE ticker symbol, (2) share price, (3) share volume, (4) broker id. In a similar application running at the NASDAQ stock exchange, a separate application may also publish events corresponding to trades, but with a different format for the first four attributes: (1) NASDAQ ticker symbol, (2) share price, (3) capital in this trade, (4) broker code. For both markets, message rates are very high (thousands of messages/second), and there are large numbers of publishers and subscribers. Thus, an important requirement for MOM is performance and scalability.

Notice that to extend the system to new applications, such as direct customer trading, it is simply necessary to "tap in" to the message streams. Provided that the infrastructure can scale from hundreds of brokers to hundreds of thousands of on-line investors, each investor can specify an appropriate selection of interest — such as the issues in his portfolio. Of course the applications used by customers and by professional stockbrokers will be very different, but the message stream "glue" will remain the same. This example illustrates both the importance of anonymity between producers and consumers, and the need to cross organizational barriers. Applications posting events need not be aware of their destinations; applications subscribing to events need not be aware of their sources. Extending the system to bring stock trade events directly to home computers may change the system load, but not the fundamental architecture.

Now let us suppose that brokers and analysts previously dealt separately with both the NYSE and NASDAQ exchanges, and that in future they wish to run the same analysis programs for trades on both exchanges. In order to run their internal analyses on information from both sources, they may wish to convert the data into a common internal format. Each of them will have to administer access to multiple sources, track changes to event formats, etc. It will be substantially easier to access information from both exchanges by accessing one data stream instead of two.

An integrated application consisting of both the NYSE and NASDAQ stock trading applications can solve this problem by transforming data from the two sources into a common format, and merging the two information streams into a single stream. For instance, both stock trade formats can be converted to a unified format consisting of the following first four attributes: (1) unique company name, (2) share price, (3) share volume, and (4) unique broker name. That is, the MOM must transform those messages it is delivering to clients requesting the common format while preserving the original format for legacy applications. The transformation enables clients accessing both stock exchanges to access this integrated data without disrupting the operation of legacy applications.

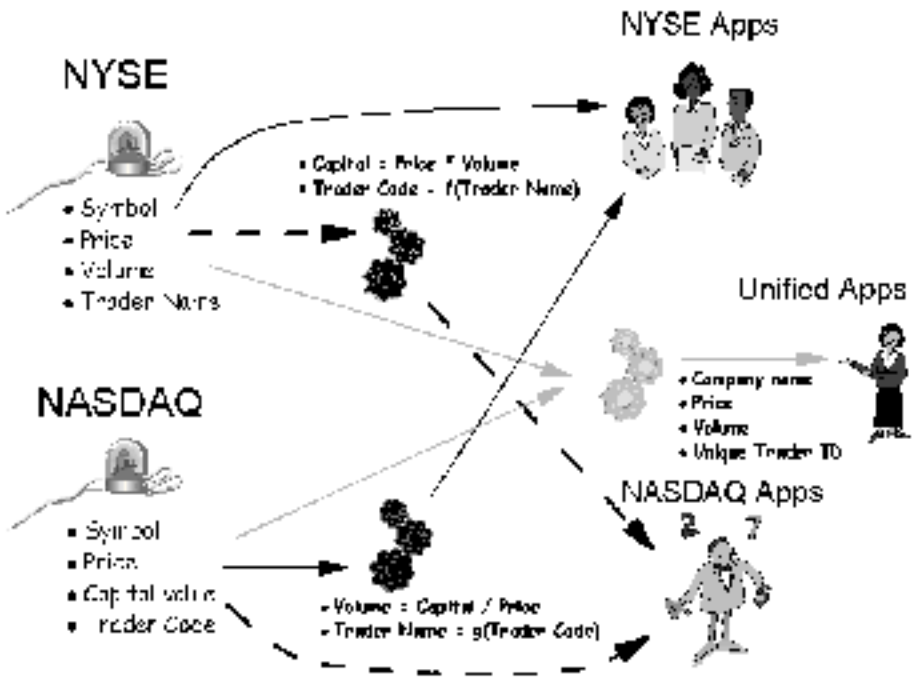

Fig. 1. Applications using both NYNEX and NYSE to exchange data.

In order to continue to use legacy client applications, it may be necessary to integrate these applications by performing other transformations. For example, legacy NYSE client applications may wish to access NASDAQ trade events in the NYSE format and vice versa.

2.2 Home Shopping

Consider a home shopping application where consumers may request up-to-the-minute information and pricing on retail items from "virtual markets" for products such as automobiles, computers, or camera equipment (Figure 2). Each message represents a seller's ad: the seller's identity and location, the type of article, and attribute-value pairs representing the attributes of the article being sold. For example, automobile advertisements would include the make, model, year, mileage, and options. The price might either be fixed, or left open to competitive bidding in a real-time auction. Additional messages represent bids by buyers, and the open and closing of auctions. Consumers subscribe through a number of tests on these attributes. As in the previous example, messages must be routed to some subset of all subscribers based on their information content.

An important function that MOM must support for this scenario is the communication of dynamic changes in the availability and the price of items. The seller may lower the price or withdraw his ad in response to lack of interest by customers. Or buyers may raise the price as a result of competitive bidding. Typically a buyer would subscribe not just to a stream of events, but to a *state* determined by the message history, such as the current price of items matching his criterion. As a result, new subscribers would not receive all messages from the beginning of time but instead only those messages representing the current valid prices for these selected items. Messages which have become superseded by updates or which correspond to items no longer available for sale need not be delivered. This example illustrates the importance of interpreting event histories as a state — specifying such a summarization requires the system to understand that a new price for the same item supersedes the previous price, and that the termination of an auction or the withdrawal of an ad cancels all previous prices for that item.

The ability to subscribe to a state summarizing a message history has additional effects on message delivery if the buyer wants to subscribe not merely to the current offer for each selected item, but instead wants to track for instance, the *lowest-priced* ad matching his criteria, plus any items for which the buyer is still the low bidder. In this case, messages which do not impact the lowest price because they are ads for articles with higher prices are not delivered. Note that if the low-priced ad is modified or withdrawn, or if its price is bid up in an auction so that it is no longer the low-priced ad, then it may be necessary to either deliver the ad for the second-low-priced item (if it had previously been suppressed) or to redeliver it. For example, consider the auto advertisements in Figure 2. The $20,000 price is the low price, so the $30,000 price is not delivered. However, once the $20,000 price is withdrawn or raised above $30,000, the ad for the $30,000 automobile is delivered to replace it.

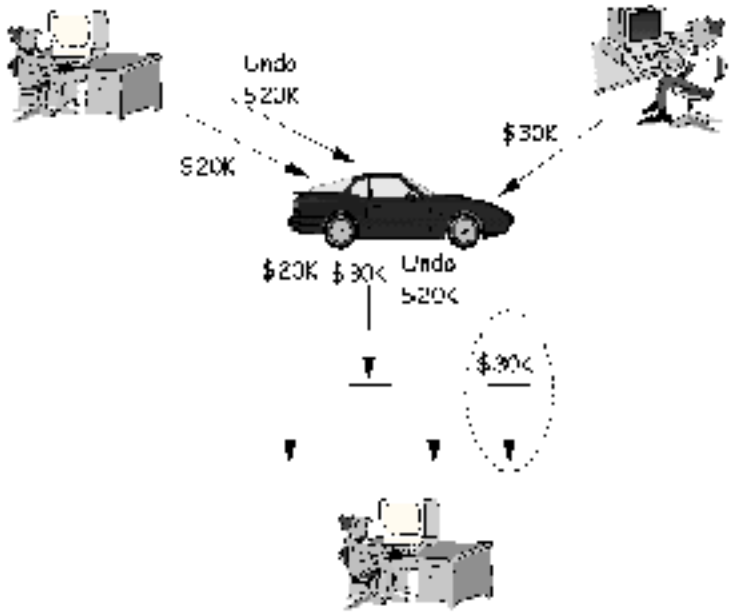

Fig. 2. Home Shopping.

Notice that, in this example, there is no real organisational boundary. Anyone might register as a publisher (potential seller) or as a subscriber (potential buyer) of an information space for a particular product.

3 A Flow-Graph Model for Message Oriented Middleware

For supporting the examples in the previous section, we desire a model which: (1) facilitates expression of clients' requirements; (2) is easy to reason about, both for validating specifications and implementations; (3) is rigorous; and (4) permits the widest possible latitude for the implementor.

Good models are founded upon a small number of basic concepts. For example, the key concepts underlying transactional database models are atomicity and serializability. The data, whether distributed and/or replicated, whether concurrently accessed or not, behave as if located at a single site and accessed one transaction at a time. Similarly, the key concepts underlying the Isis model [BCJ+90] are group membership [Bir93,Edi96] and virtual synchrony [BT87]. Once a model is chosen, we have a rigorous requirement for developing clever implementations for preserving the appearances of the model under a wide variety of design points and physical environments.

The Gryphon project at IBM Research (http://www.research.ibm.com/gryphon) is exploring a particular model for MOM based upon the concepts of *information spaces* and *message flows*. A system is modeled as a *message flow graph*: an abstract representation of the propagation of the messages in a system, divorced from any realization on an actual network topology [BKS+99]. A message flow graph is a directed acyclic graph whose nodes are information spaces and whose arcs are message flows. Information spaces are either totally ordered message histories or states derived from message histories. Each information space has a schema that specifies the type of the messages or of the state. Publishers and subscribers are source and sink message histories respectively. Message flows specify the propagation of messages or the updating of state, in order to preserve specific relationships between information spaces as new messages are added to the system. These relationships include:

- *selection*: the destination message history receives a copy of the subset of the source message history that satisfies some boolean predicate P. For example, an analyst may request all trades of more than 1000 shares of XYZ company having a price greater than $40 per share.
- *transform:* the destination message history receives each message of the source message history after applying a transform T. For example, the conversion "Volume = Capital/Price" might be used to convert all messages from the NASDAQ format to the NYSE format.
- *merge:* when multiple message histories have arcs to a single destination, the destination receives a merge (in a non-deterministic order) of the all the messages of the sources. This operation is involved in any application involving multiple publishers.

- *collapse:* the destination state is computed by applying some summarization function S over all the messages in the source message history. In the home shopping example, a client may be interested in the lowest priced ad for Saab cars with less than 80 000 miles and a price under $6000.
- *expand: expand* is the inverse of *collapse*. The source message history is (nondeterministically) computed to be any message history which summarizes to the destination state under the summarization function S. All such message histories are said to be *equivalent* — the system is free to choose which one of the many message histories to deliver to a destination.

Message flow graphs can evolve dynamically and in fact the changes to these graphs and requests to change the graph are themselves meta-events that can be subscribed to. As in similar systems, access control policies limit who may add and delete nodes and arcs, and where they may be added.

Notice that this model has some characteristics of database systems, some of group communication systems, and some unique characteristics of its own. Information spaces in MOM are analogous to tables in relational databases. Just as database tables have data schemas, information spaces have message schemas. The selection relation between information spaces is similar to the select operator between relational tables. Just as a relational database allows linkages between tables and views, MOM allows linkages between information spaces via message flow graphs.

Like group communication systems, messages flow from producers to consumers without explicit requests from consumers — i.e., both systems are push-based. In a push-based environment, it is natural that operations on the message flow, such as selections, transforms, and summarizations, are performed incrementally.

Defining and implementing the flow graph model gives rise to a number of open research issues. These issues are discussed in detail in the next section.

4 Research Issues

4.1 Model

The message flow graph is a useful abstraction for specifying many problems such as the ones discussed in the previous section. It is easy to explain to users familiar with either dataflow graphs or with spreadsheets.

There are, however, a number of open issues. One is the type system for defining schemas for messages and information spaces. Whereas it makes sense to organize relational databases as tables in normal form with each row consisting of a tuple of scalars, a similar "normal form" is probably not feasible for message histories. One reason is that while relational tables containing rows of different formats can be factored into separate tables, message histories cannot be so factored without losing the intrinsic total order characteristic of histories. It is probably necessary to allow messages to contain variant types, as well as embedded lists or bags.

Another open issue is the language for expressing the selection predicates, the transforms, and the summary functions used by *expand* and *collapse*. We want to be able to express problems like "cheapest current offer for a Saab" without elaborate programming. There is a tradeoff between convenience, expressive power, and analyzability.

Another issue is how to handle access control. In this model, access control can involve more than merely saying "this user may subscribe from this space". Some subscriptions require all messages to be archived forever, while others allow messages to be expired relatively quickly — these differences have consequences for physical resource requirements on broker servers, so there should be a way to allow access control to take these consequences into account.

4.2 Scalability

There are many dimensions of scalability. In this section, we deal with the potentially large fanout of select, transform, or summarization arcs from a single information space. In a large application, or in an anonymous environment such as home shopping where a single information space may be advertised very widely, the number of subscribers may be very large — perhaps in the tens of thousands or more. In this environment it becomes necessary to deploy algorithms that quickly match events against a large number of potential subscriptions — we refer to this problem as the *message matching* problem. Though the number of subscriptions to an information space may be large, say N (where N may be 10,000 or more), we expect only a few subscriptions to match any single event, say K. Efficient algorithms exist for solving the message matching problem in messaging systems based on subject-based subscription; a simple table lookup based on the subject of the message yields a constant time algorithm, which is also optimal. In the more general content-based subscription systems, this approach does not work since different subscriptions may refer to different fields of the message schema. Naive solutions take O(N) steps to solve the message matching problem. It is highly desirable to develop algorithms to solve the message matching problem that are sublinear in N. This is an active area of ongoing research [ASS+99].

Note that the message matching problem is complementary to the query problem in databases. In a database query, a single query (typically a select) is matched against a large amount of data — the challenge here is to develop algorithms that are sublinear in the amount of data in the database. In the message matching problem a single piece of data (a message) needs to be matched against a large number of standing queries (subscriptions). Since the problems are complementary, neither solution is useful in the other context. Note that the matching problem was first studied in the context of active databases. Efficient solutions to this problem are thus applicable in MOMs and in databases [HCKW90].

A problem similar to the message matching problem arises when there is a large fanout of arcs between a single information space and multiple states. By analyzing the multiple summarization functions we may be able to avoid the

need to make multiple copies of the same state update and to exploit the fact that if one state doesn't change as a result of a message, a set of related states will also not change.

4.3 Distributing broker networks

In a distributed implementation, the above solutions for matching have to be modified to reflect the fact that the message flow graph will typically be implemented over a geographically distributed network of server processes, which we call *message brokers*. Message brokers must combine the functions of routing and multicasting with the functions of implementing selections, transformations, and summarizations. Thus it becomes necessary to develop distributed solutions to these problems — this is an active area of ongoing research [BCM+99].

Consider two naive solutions to this problem:

- Perform message matching at the publisher and use the result of the matching to route to the destinations. With this solution, straightforward routing techniques will not work when there are a large number of clients, since (a) point-to-point routing will not take advantage of common paths, (b) routing based on destination lists could result in large message headers, and (c) routing based on multicast groups could require a very large number of groups (if there are N subscribers, the system may need as many as 2^N multicast groups).
- Broadcast the message to all message brokers and let each message broker match the message against its locally connected subscribers. This solution is likely to waste communication bandwidth in very large networks, since if subscriptions are sufficiently selective, messages will often be sent to a broker none of whose attached clients requires the message.

Approaches to the problem being studied include: (1) performing partial matching at each broker, forwarding messages (either by conventional point-to-point or by multicast) to the subset of neighboring brokers requiring the message; (2) matching messages to a combination of pre-allocated multicast groups; (3) exploiting the relationships between the subscriptions to reduce the combinatorial possibilities of multicast groups.

The existence of message transformations complicates the situation even further — some transformations can be moved and/or replicated on multiple brokers; others cannot, either because they involve data that cannot be moved (e.g. a large database mapping names to social security numbers), or because they involve "opaque" algorithms not visible to the middleware.

4.4 Message Reliability

The fault model that is typically implemented in traditional group communication systems — that a failed or slow process is automatically removed from the group [BT87] — is inappropriate for MOM applications. In MOM, the message

flow graph is viewed as an abstract reliable entity: Subscriptions are persistent, and messages may not be lost, permuted or duplicated, nor must spurious messages be generated (unless such distortions can be masked as a result of filtering or state equivalence). The implementation must preserve the appearances of persistence even though in the message distribution scenarios shown above the distributed system may contain intermittently connected and intermittently faulty hardware components. This means that when a faulty subscriber reconnects it must be possible to either deliver all the messages that it has missed, or else to compute (via analysis of the effects on the state of interest) a shorter set of messages which will re-create this state. Unlike with group communication systems, it is not sufficient to report to a faulty or disconnected subscriber that its subscription has been dropped. For example, the Replenishment Analysis teams must continue to receive inventory updates after a dropped connection is reestablished. We need algorithms to provide this appearance of persistence in a distributed network of message brokers. We also need algorithms to exploit the cases where state equivalence permits dropping of messages, and to exploit the properties of state equivalence to deliver compressed message sequences after a reconnection.

4.5 Message Ordering

Information spaces support the abstraction of a total order on messages. Since subscribers specify their interest in states derived from message histories, the middleware has the opportunity of relaxing total order deliveries for specific clients while preserving the meaning of the overall message history. This is in contrast with the approach taken by group communication systems in which ordering guarantees are driven by low level protocol options (e.g., publisher ordering, causal ordering, etc.) [BCJ+90].

For instance, if a subscriber is subscribing to the current price of a set of advertised items, the subscriber may be sensitive to the order of the last two updates to the same item, since the current price depends upon which update is first. The subscriber may not be sensitive to the order of earlier updates to that item or to the order of updates to different items. This gives the system the flexibility to weaken the ordering requirement where it is legitimate to do so while preserving it in the cases where it matters. However, it gives rise to open issues of how these situations are detected. It also creates the opportunity for optimistic delivery of out-of-order messages, as discussed below.

4.6 Optimistic Delivery

Efficient message delivery implementations that address fault-tolerance and ordering make a distinction between the receipt of a message and its actual delivery to a client — it is often necessary for the system to delay the delivery of a received message until certain control messages have arrived, such as for example, notifications that the data is stable and that no earlier messages are still en

route. It is desirable wherever possible to deliver messages optimistically without waiting for this control information. In the simplest cases, the subscriber's state of interest doesn't depend upon order or isn't affected by extraneous unlogged messages. However, in more interesting cases the state of interest does depend upon order, but the state interpretation makes it possible for "recovery" messages to retrieve the correct state after an out-of-order or unlogged message has arrived. For example, in the home shopping example, it may be that an offer to sell for $20000 is followed by an offer to sell for $30000. If the offer for $30000 arrives first, it can still be immediately delivered; when the offer for $20000 arrives later, the recovery action is to deliver it if it is for a different item than the $30000 offer, and to discard it as obsolete if it is for the same item.

It is an open problem to analyze a set of subscriptions to state derived from message histories, and determine (a) under which conditions messages can be optimistically delivered without waiting for control messages, and (b) what "recovery" messages must be inserted if it is later determined that the state needs to be corrected.

4.7 Topology Changes

End-users don't care about the topology of the underlying network. Ideally (a) it should be possible to reconfigure the topology of the underlying network non-disruptively, and (b) it should not require complex planning on the part of a network administrator to configure. Any approach to the topology reconfiguration problem must address scenarios in which multiple organisations may own parts of the communications links and logging disks, and these organizations must be able to reconfigure and/or control use of their facilities.

4.8 Security

MOM needs at least three varieties of security: (1) control of who may publish to or subscribe from the information spaces of the virtual message flow graph; (2) control of the physical resources; (3) privacy protection of the data that flows between publishers and subscribers. Any security solution must accept the fact that there is no single "application" and no single owner of the whole network.

There are open issues about: (1) preventing a user from overloading system resources by either generating messages too quickly or by requesting states that make it impossible to discard any old messages; (2) how to deal with clients to the same information space from different organizations having different access rights; and (3) the tension between the requirement for brokers to do content-based matching and the requirement for some brokers not to be able to interpret the content.

5 Alternative Approaches

Other technologies, including object request brokers (ORBs) and database management systems (DBMSs) are being used to glue applications together in the

kinds of scenarios presented in this paper [Gro98,Hsi92,WfM,BN83]. However, each of these approaches has its limitations for the purpose of MOM applications, as described below.

5.1 Remote Method Invocation (RMI) Systems

ORBs (e.g. [Gro98]) can be used to glue applications by having one application call methods of objects in another remote application [BN83]. The interfaces supported by an application are specified in an interface definition language (IDL) which are compiled into stubs for the caller and into language templates for the callee. RMI systems have several shortcomings that make them unsuitable for MOM applications:

- *Application evolution:* With this approach, applications tend to get tightly integrated, right from the design stage. Changes are difficult if not impossible to make after an application has been deployed. Also, since remote method invocation is a point-to-point concept, it is not possible to interpose new applications between existing information flows without disrupting the existing applications.
- *Disconnected operation.* RMI systems support a synchronous style of interaction — this makes them unsuitable in environments where clients may disconnect.

5.2 Database Systems

In general, database systems are optimized for a different set of applications than the ones that are presented in this paper. For example, databases are optimized for queries over a large amount of saved data as opposed to matching a message against a large number of standing queries or computing a summary state from a sequence of messages. Also, database systems usually support a small number of views whereas MOM systems must support a large number of views and must be optimized for frequent view updates. Furthermore, the overhead of distributed transactions in databases is prohibitively large for MOM applications.

The database community has developed a variety of techniques to use shared databases to glue together applications in a distributed environment. With this approach, one application adds data to a shared database and another application retrieves the data from it. Shared databases can be used in several configurations for this purpose, but all of them have their limitations:

- *Pull-based:* The receiving application may poll the shared database for new "incoming" data; this unnecessarily introduces extra network traffic.
- *Active Databases.* The receiving application may be alerted about new data in the database using a trigger mechanism. However, the trigger mechanism may not scale over a large number of receivers interested in different kinds of updates to the shared database.

- *Client-server architecture*: In this architecture, distributed clients access a centralized database. This approach offers limited scalability, and does not have the ability to cross organizational boundaries. Changes must first propagate to the centralized database before being sent to interested viewers.
- *Distributed database architecture*: In this architecture, the database is replicated at multiple sites. In many scenarios for gluing applications together, the replication guarantees provided by distributed databases may be too strong.
- *Federated databases:* In this architecture [KK90, Hsi92, GL94], a collection of independently designed databases is made to function as a single database. This involves name conflict resolution, schema conversion, and the execution of transactions on multiple databases as a single global transaction. Although this approach may be appropriate for organizing multiple databases within a single company, or for merging two companies together, it is unlikely to be feasible to run global serial transactions across multiple organizations and thousands of anonymous subscribers worldwide.

In general, a database used as a communication channel is too heavyweight as glue between applications. This approach has a significant administrative overhead and may not provide the same kind of communication throughput that a more specialized communication channel could provide. Thus, although databases cannot be a total solution, especially given heavyweight commit protocols that we often don't need in the message flow graph solutions, databases are still potential clients of MOM systems.

5.3 Group Communication Systems

Group communication systems (e.g., [Edi96]) can be used to glue applications by having applications join process groups meant for exchanging particular types of messages. This technique is commonly used to implement subject-based pub/sub, where a subject (or a channel) is implemented as a process group. In fact, we view MOM as a natural evolution of the group communication approach. However, this approach has several shortcomings if used to support the full spectrum of MOM applications.

- *Flexibility*: The group communication based approach imposes a fixed subject structure on all applications — this reduces the flexibility of the overall system. In large systems, it may be necessary for different applications to select messages based on different fields in a message — a subject structure cannot capture this requirement.
- *Scalability*: Group communication system implementations tend to be tightly coupled; thus it is natural to deploy group communication systems over small numbers of computers (100s) on a tightly coupled network (e.g., a LAN). Scaling group communication to larger numbers of computers and onto WAN environments is an open area of active research [BFH97].

- *Fault model.* The fault model that is typically implemented in group communication systems — that a failed process is automatically removed from the group — is inappropriate for MOM applications. In MOM, subscriptions are persistent: when a failed process recovers it needs to be updated with all the messages that it did not receive.
- *Opaque messages:* In general, group communication systems do not interpret the content of messages. This forces them to support qualities of service based on low level properties of the protocol, not on the semantics of messages. MOM systems can get more information from applications and use it to provide various qualities of service more effectively.

5.4 Workflow Systems

Workflow systems (e.g., [WfM,Lut94]) are used for coordinating potentially distributed tasks via a specification of the sequence and control of tasks. While these systems are typically used to solve problems at a "higher-level", they may also be used to glue applications by treating each application as an "activity" that communicates with other "activities" via a workflow manager (which is the software component that controls the flow of work between activities). However, the major shortcomings of this approach when used for integrating applications are:

- Workflow specifications are relatively static in nature — activities and their interactions do not change once the flow has been defined. Applications requiring integration, on the other hand, may need to support changes to subscriptions at a frequent rate.
- Workflow managers are centralized in practice. This may limit the scalability and the throughput of the system. Building a distributed Workflow system is an active area of ongoing research [PPHC98].

6 Related Work in Content-based Publish-Subscribe Systems

Several projects [WIS], such as SIENA [Car98], READY [GKP99], Elvin [SA97], and Gryphon (http://www.research.ibm.com/gryphon), are exploring the use of content-based publish-subscribe systems as the basis for various MOM applications. While the motivations for the work in these projects are similar, the approaches they are pursuing are different in important respects.

From a model point of view, all of the above systems support rich subscription languages which approach the expressiveness of SQL. Some of the systems, e.g., READY, also support the use of temporal relationships between messages for expressing "compound events". As described earlier, the content-based publish-subscribe model can be generalized to include transformations, merges, and stateful operations in the single framework of message flow graphs – this approach is being explored by the Gryphon project. None of the above projects has explored the notion of cross-domain message flows in any detail.

From an implementation point of view, none of the above projects has addressed all the research issues mentioned earlier in this paper. The first scalable solutions for matching and multicasting have appeared recently [ASS+99 BCM+99]. The SIENA project has explored issues relating to efficient propagation of subscriptions. Questions of efficient matching and multicast message reliability, message ordering, and optimistic delivery are currently being explored in the Gryphon project.

References

[ASS+99] Marcos Aguilera, Rob Strom, Daniel Sturman, Mark Astley, and Tushar Chandra. Matching events in a content-based subscription system. In *Proceedings of the 18th ACM Symposium on the Principles of Distributed Computing, Atlanta, GA*, May 1999.

[BCJ+90] K. P. Birman, Robert Cooper, Thomas A. Joseph, Kenneth P. Kane, and Frank Bernhard Schmuck. *ISIS - A Distributed Programming Environment*, June 1990.

[BCM+99] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Rob Strom, and Daniel Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the International Conference on Distributed Computing Systems 1999, Austin, TX*, June 1999.

[BFH97] Ken Birman, Roy Friedman, and Mark Hayden. The maestro group manager: A structuring tool for applications with multiple quality of service requirements. Technical Report TR97-1619, Cornell University, Computer Science, feb 1997.

[Bir93] K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, Dec 1993.

[BKS+99] Guruduth Banavar, Marc Kaplan, Kelly Shaw, Rob Strom, Daniel Sturman, and Wei Tao. Information flow based event distribution middleware. In *Proceedings of the Middleware Workshop at the International Conference on Distributed Computing Systems 1999, Austin, TX*, June 1999.

[BN83] A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. In *Proceedings of the ACM Symposium on Operating System Principles*, page 3, Bretton Woods, NH, October 1983. Association for Computing Machinery. Association for Computing Machinery.

[BT87] K. P. Birman and Joseph A. Thomas. Exploiting virtual synchrony in distributed systems. In *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, pages 123–138, November 1987.

[Car98] Antonio Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, December 1998. Available from http://www.cs.colorado.edu/ carzanig/papers/.

[Edi96] David Powell (Guest Editor). Group communication. *Communications of the ACM*, 39(4), April 1996. This is a collection of several papers in the area.

[GKP99] R. Gruber, B Krishnamurthy, and E. Panagos. An architecture of the ready event notification system. In *Proceedings of the Middleware Workshop at the International Conference on Distributed Computing Systems 1999, Austin, TX*, June 1999.

[GL94] P. Gupta and E. Lin. Datajoiner: A practical approach to multi-database access. In *Parallel and Distributed Information Systems (PDIS '94)*, pages 264–264, Los Alamitos, Ca., USA, September 1994. IEEE Computer Society Press.

[Gro98] Object Management Group. Corba services: Common object service specification. Technical report, Object Management Group, July 1998.

[HCKW90] Eric N. Hanson, Moez Chaabouni, Chang-Ho Kim, and Yu-Wang Wang. A predicate matching algorithm for database rule systems. In *SIGMOD 1990, Atlantic City N. J.*, pages 271–280, May 1990.

[Hsi92] D. Hsiao. Federated databases and systems: Part I – A tutorial on their data sharing. *VLDB Journal*, 1(1):127–179, July 1992.

[KK90] Magdi N. Kamel and Nabil N. Kamel. The federated database management system: an architecture of distributed systems for the 90's. In *Proceedings, Second IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 346–352, 1990.

[Lut94] R. Lutz. IBM flowmark workflow manager - concept and overview. In G. Chroust and A. Benczur, editors, *Connectivity '94 - Workflow Management - Challenges, Paradigms and Products*, pages 65–68, Linz, Austria, October 1994. R. Oldenbourg, Vienna, Munich.

[OPSS93] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. The information bus - an architecture for extensible distributed systems. *Operating Systems Review*, 27(5):58–68, Dec 1993.

[PPHC98] S. Paul, E. Park, D. Hutches, and J. Chaar. RainMaker: Workflow execution using distributed, interoperable components. *Lecture Notes in Computer Science*, 1513:801–??, 1998.

[SA97] Bill Segall and David Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings, AAUG97*, September 1997.

[SV97] D. C. Schmidt and S. Vinoski. The OMG Events Service. *C++ Report*, 9(2):37–46, February 1997.

[TGGL82] I. L. Traiger, J. N. Gray, C. A. Galtieri, and B. G. Lindsay. Transactions and consistency in distributed database management systems. *ACM Transactions on Database Systems*, pages 323–342, Sept 1982.

[WfM] Workflow management coalition. http://www.aiai.ed.ac.uk/WfMC.

[WIS] Workshop on internet scale event notification. See http://www.ics.uci.edu/IRUS/wisen/wisen98 for details.