# Information Flow Based Event Distribution Middleware

Guruduth Banavar[1], Marc Kaplan[1], Kelly Shaw[2], Robert E. Strom[1], Daniel C. Sturman[1], and Wei Tao[3]

[1]*IBM T. J. Watson Research Center*
*Hawthorne, NY*
*{banavar, kaplan, strom, sturman}*
*@watson.ibm.com*

[2]*Dept. of Computer Science*
*Stanford University*
*kashaw@cs.stanford.edu*

[3]*Dept. of Computer Science*
*University of Utah*
*tao@cs.utah.edu*

## Abstract

*Event distribution middleware supports the integration of distributed applications by accepting events from information producers and disseminating applicable events to interested consumers. In this paper we present a flexible new model, the* Information Flow Graph (IFG)*, for specifying the flow of information in such a system. We illustrate the use of the IFG for: (1) content-based publish/subscribe; (2) stateless event transformations that consolidate events from diverse sources; and (3) stateful event interpretation functions for deriving trends, summaries, and alarms from published events and for defining equivalent event sequences. We introduce two techniques for efficient implementation of such systems: (1) a flow graph rewriting optimization which allows stateless IFGs to be converted to a form which can exploit efficient multicast technology developed for content-based publish/subscribe systems; and (2) an algorithm for converting a sequence of events to the shortest equivalent sequence of events with respect to an event interpretation function.*

## 1 Introduction

Event distribution middleware is growing in importance with the need to glue together heterogeneous, distributed, and dynamically changing components of large information systems. The middleware performs the function of collecting messages from producers, filtering and transforming them as necessary, and routing them to the appropriate consumers. This approach is currently being applied in domains such as finance, process automation, and transportation. The Gryphon project at IBM Research is advancing the technology of event distribution middleware and extending its range of application.

Using subject-based publish/subscribe systems as a starting point, Gryphon has introduced the following extensions:

1. *Content-based publish/subscribe*. Rather than treating events as uninterpreted data with a single “subject” field, we associate schemas with event streams, and express subscriptions as predicates over all fields in the event.
2. *Stateless event transformations.* To support scenarios where events from multiple publishers are similar but not identical, Gryphon supports transformations on events. These operations are stateless in the sense that they do not depend upon prior events.
3. *Event stream interpretations.* To support subscribers who are interested not only in published events but also in events such as summaries, trends, and alarms, derived from a sequence of related events, the Gryphon model supports several “stateful” operations as well (operations whose results depend on the event history). State can also be used to express the “meaning” of an event stream, and by implication, the equivalence of two event streams.

In this paper, we describe Gryphon’s approach to event distribution middleware based on the concept of *information flow graphs* (IFGs). We show that IFGs not only are a flexible and powerful model for expressing event flows, but also can be efficiently implemented on a distributed network of event brokers.

Section 2 defines the IFG model. The motivations for content-based subscription, and efficient and scalable algorithms developed by the Gryphon project for matching events to subscriptions and delivering them are omitted here, since they are discussed in detail in [1] and [3]. Section 3 introduces motivating examples of stateless and stateful event transformations. Section 4 discusses the implementation problems of the IFG approach, and then presents an overview of two implementation techniques we have developed to address these problems. Section 5 discusses related work, the current status of this work, and concludes.

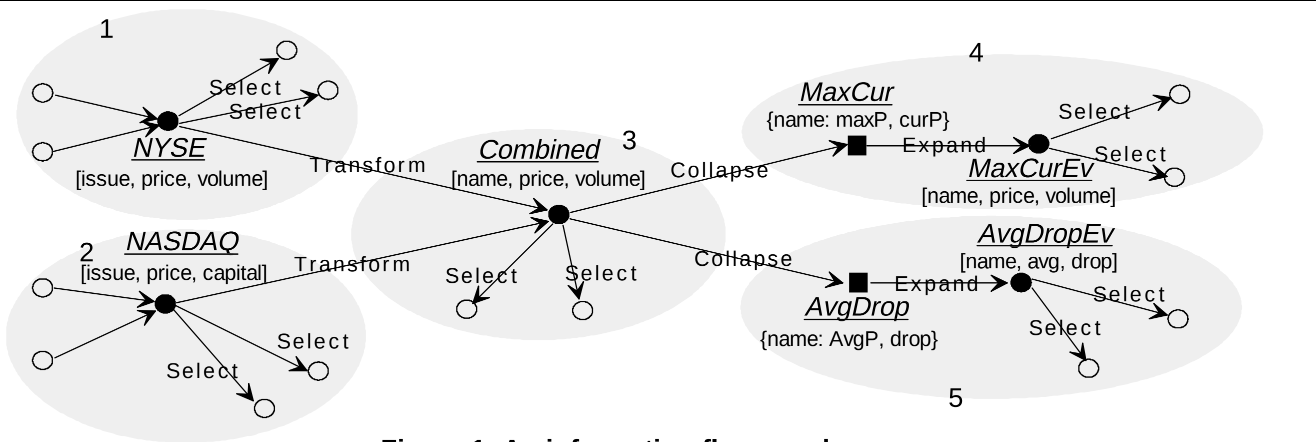

**Figure 1: An information flow graph.**

## 2 The Information Flow Graph

In Gryphon, an event system is modeled as an information flow graph. Figure 1 illustrates such an IFG for a collection of stock services. An IFG contains the following components:

- *Information spaces*. They are either *event histories* (circles, e.g. NYSE) or *states* (squares, e.g. MaxCur). Event histories are lists of events. They grow monotonically over time as events are added. States capture selected information about event streams, and are typically not monotonic. The type of an information space is defined by an *information schema*. In this paper, we assume that each event is a typed tuple. For instance, the NASDAQ information space is a sequence of events having the schema [issue: string, price: integer, capital: integer]. The MaxCur information space is a state represented as a keyed relation associating the name of a stock issue with its maximum price and current price. Certain event histories, represented as unfilled circles, are *sources* or *sinks*; these represent the information providers and consumers.
- *Dataflows*. These are directed arcs (arrows) connecting nodes in the graph. The graph is required to be acyclic. Sources must have only out-arcs and sinks in-arcs. State nodes must have only a single in-arc. The arcs determine how the contents of the information spaces change as events enter the system.

There are four types of dataflows, indicated by the labels on the arcs:

- *Select*. This arc connects two event histories having the same schema. Associated with each select arc is a predicate on the attributes of the event type associated with the information space. An example of a predicate is the expression (issue="IBM" & price<120). All events in the information space at the source of the arc which satisfy the predicate are delivered to the information space at the destination of the arc.
- *Transform*. This arc connects any two event histories which may have different event schemas $E_S$ and $E_D$. Associated with each transform arc is a *rule* for mapping an event of type $E_S$ into an event of type $E_D$. For example, the transform arc connecting the space NASDAQ to the space Combined is labeled with the rule
  [issue:i, price:p, capital:c] $\rightarrow$ [name: NAS(i), price:p, volume:c/p]
  which maps the issue to a name using the function NAS, and derives volume as capital divided by price. Whenever a new event arrives at the space at the source of the arc, it is transformed using the rule and delivered to the space at the destination of the arc.
- *Collapse*. This arc connects an event history to a state. Associated with each collapse arc is a rule for collapsing a sequence of events to a state. The rule maps a new event and a current state into a new state. For example the following rule defines the collapse arc from the space Combined to the space *Maxcur*:

  $$[n, p, v], \langle n: p > maxP, curP\rangle \cup s \rightarrow \langle n: p, p\rangle \cup s$$
  $$[n, p, v], \langle n: p \leq maxP, curP\rangle \cup s \rightarrow \langle n: maxP, p\rangle \cup s$$

  This rule contains two patterns: in each, the tuple in the state is found whose key matches the name field n of the event [n, p, v]. If the price p in the event is greater than the current max price maxP, the first pattern is triggered, and the state is updated by replacing maxP and curP with p. Otherwise, the second pattern is triggered, and only curP is replaced. Given an initial state (in this example, a maximum and current price of zero for all stocks), the state at

*Maxcur* is updated each time a new event is added to *Combined*.

- *Expand.* This is the inverse of *Collapse*. This arc links a state to an information space. Associated with each arc is a collapse rule. When the state at the source of the arc changes, the destination space is updated so that the sequence of events it contains collapses to the new state. Notice that unlike the other dataflows, *expand* is non-deterministic. For a given state, there may be many possible event sequences which map to the state, or there may be none. The non-determinism is further constrained by the need for information spaces to be observably monotonic: that is, an expansion may not "undeliver" an event already delivered to a consumer. We restrict the language to avoid the case in which there is no possible event sequence, but we exploit the non-determinism to give flexibility to the implementation to deliver one of a set of equivalent event sequences.

In addition to the above four operations, there are two operations implicit in the graph. Fan-in to an event history produces a merge of the events --- there is non-determinism here too, as multiple interleavings are possible. Fan-out from an event history replicates the events.

## 3 Motivating Examples of IFGs

Consider regions 1 and 2 of the stock event system shown in Figure 1. Each of the regions has an information space, a collection of producers, and a collection of consumers with content-based selections on the events of the information spaces *NYSE* and *NASDAQ*. These regions are examples of "pure content-based pub/sub" systems. The consumers with content-based selections correspond to *subscribers.*

Region 3 represents a service attempting to integrate the two spaces *NYSE* and *NASDAQ.* These exchanges have different conventions for issue names; therefore it is desirable to map the local issue names to a common name via some conversion table. Furthermore, one exchange delivers trades using price and volume, the other using price and total capital (price times volume). It is therefore necessary to map these into either one of the two formats or a common format. The result is a new information space *Combined,* containing the union of the two original information spaces, but in a common format, enabled by the use of stateless event transforms. Subscribers to the new service can deal with this new space and need not even be aware of the existence of the original suppliers.

Region 4 represents a collection of subscribers to *Combined* who are interested in particular stock events, but whose requirements on guaranteed delivery are weaker. It should be pointed out that an event history, such as *Combined,* has a total order. Even though the total order depends upon non-deterministic factors, such as the order in which events from *NYSE* and events from *NASDAQ* are merged, the dataflow semantics discussed in the previous section guarantee that all subscribers to *Combined* receive the events in the same order. Guaranteeing this total order adds to the cost of the delivery protocol.

However, the subscribers to region 4 have a weaker requirement: they are interested only in tracking the maximum price and current price of each stock issue. They cannot ignore ordering entirely (otherwise they might swap today's price of IBM with yesterday's price), but they can ignore the order between today's IBM price and today's HP price. And under appropriate conditions, messages may be dropped altogether. These subscribers express this requirement by defining an *event interpretation* — a mapping of the event sequence into a state which captures precisely the information relevant to these consumers, namely the current and maximum price of each issue.

The *collapse* arc converts the event sequence from *Combined* into a state representing this event interpretation. The *expand* arc converts the state back into an event sequence. The associated rule on this arc is the identical rule from the *collapse* arc. Therefore, the events in *MaxCurEv* can be any sequence of events whose interpretation is the same as the interpretation of the events in *Combined*. A trivial solution is to treat the *collapse* and *expand* as a null operation and deliver exactly the same events to *Combined* and to *MaxCurEv*. However, the non-determinism of *expand* permits cheaper solutions, in which some events can be dropped or permuted. One instance where this flexibility is important occurs when the subscriber disconnects from the network without terminating the subscription and later reconnects. Rather than bombarding the subscriber with all the events which would have been delivered during the disconnect period, the system instead delivers a much shorter equivalent system that preserves the specified interpretation: the current and maximum price of each stock. In the next section, we show an algorithm for computing the minimal event sequence after a disconnect and reconnect.

Once the *collapse* operation has been introduced, it is possible to use it not only for equivalent event sequences, but also for deriving new types of events from the state. In region 5, we show a *collapse* operation introduced to compute a state *AvgDrop* which tracks for each stock

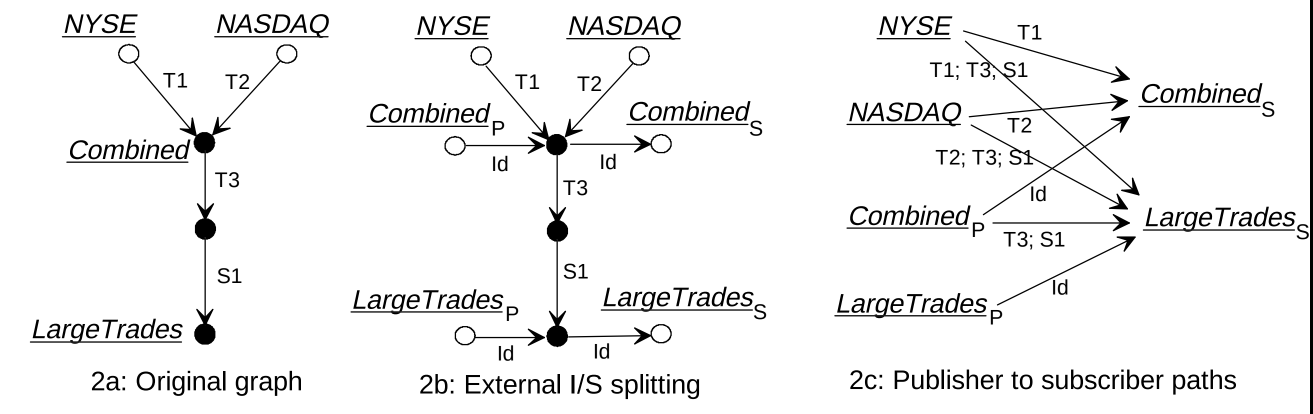

**Figure 2: Example of graph rewriting**

issue, the average price and the magnitude of the largest recent price drop. From that state, we can introduce an *expand* operation to produce a new event space named *AvgDropEv*. Consumers wishing to be alerted to "alarms" such as a drop exceeding 20 can then subscribe to this derived event space.

## 4 Implementation Techniques

IFGs are logical descriptions of the flow of events in a system. Ultimately, this description must be realized on a physical network of message brokers. The problem of mapping an arbitrary logical IFG to a physical broker network is nontrivial. If done naively, the performance of efficient content-based routing systems (such as the one in [3]) cannot be exploited at all.

In this section, we present solutions to two implementation problems.

The first problem is the consolidation of transform operations at the periphery and select operations at the interior, so that we can use existing techniques for efficient content-based subscription as the basis for an implementation of an IFG with both selects and transforms.

The second problem is how to implement *expand* by producing the shortest event sequence corresponding to a given change in state.

### 4.1 Reordering Selects and Transforms

Our approach to efficient realization of IFGs is to *reduce* an arbitrary IFG to one that can be efficiently implemented on a content-based routing system. The basic idea is to rewrite the IFG so that all the select operations are lumped together and moved closer to publishers, and all the transform operations are lumped together and moved closer to the subscribers. (Because transforms may destroy information, they cannot, in general be pushed ahead of selects.) This will allow us to use the content-based routing protocols described in [3] to implement the select operations within the broker network, then perform the transform operation at the periphery of the broker network. Furthermore, we may be able to optimize away transform operations on events that would be eliminated by later select operations.

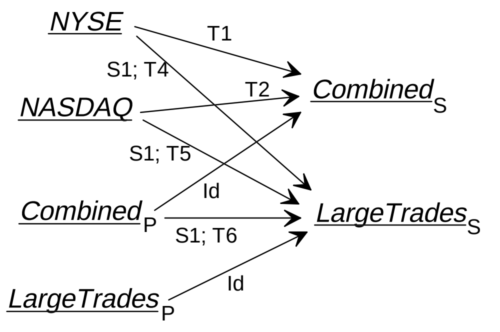

**Figure 3: Final optimized IFG**

Rewriting the IFG can be done by an automated system so that, while users specify information flows as a series of selects and transforms that closely matches the way they think about the flow and processing of events, the system can optimize the processing of information flows.

The rules for rewriting graphs are described below.

**Selects can be pushed ahead of transforms.**

For any dataflow in which a transform TA is followed by select SA, there is an equivalent dataflow of the form SB followed by TB. To see how, observe that the predicate of SA must be a function of constants and the function outputs of TA. We can construct a predicate for SB that will choose the same messages as SA, by simply substituting the appropriate functions of TA for the attributes in the predicate of SA.

For example (in all of the examples in this section, the semicolon is used to mean "followed by" in an information flow):

```
TA: [x1, x2] => [y1=f1(x1,x2,c1), y2=f2(x1,x2,c2)];
SA: (p(y1,y2,d))
```

can be rewritten as:

```
SB: (p(f1(x1,x2,c1), f2(x1,x2,c2),d));
TB: [x1,x2] => [y1=f1(x1,x2,c1), y2=f2(x1,x2,c2)]
```

where x1,x2,y1,y2 are attribute names, and c1,c2,d are constants.

**Selects and Transforms can be combined.**

Observe that a sequence of selects is just a conjunction of predicates. Thus, selects SA: (p(...)) followed by SB: (q(...)) can be rewritten as SC: (p(...) & q(...)).

Similarly, we can perform variable substitutions from a first transform into a second. For example:

```
[x,y] =>  [y1:=f1(x1,x2,c1), y2:=f2(x1,x2,c2)];
[y1,y2] => [z1:=g1(y1,y2,d1), z2:=g2(y1,y2,d2)]
```

can be rewritten as

```
[x,y]=> [z1:=g1(f1(x1,x2,c1),f2(x1,x2,c2),d1),z2:=
        g2(f1(x1,x2,c1),f2(x1,x2,c2),d2)]
```

By applying the above rewriting rules, any sequence of selects and transforms can be reduced to a single select followed by a single transform. For example, starting with the sequence [T T S T S T], we push selects ahead of transforms to get [S S T T T], and combine selects and transforms to get [S T].

Once a dataflow has been reduced so that all paths from publishers to subscribers may be represented by a [Select;Transform] pair, the single select can be further optimized. A straightforward application of the rewriting rules may cause many common subexpressions to appear within the combined predicate. A smart implementation can discover those, just like any good compiler, and avoid recomputation of sub-functions. Likewise, the final, single transform will likely contain many common subexpressions and many of those will have already been computed for the select. A smart implementation can cache them and/or tag each selected message with them as auxiliary attributes.

**Externalizing I/S nodes as terminal nodes.**

As we combine and rearrange the select and transform operations specified by an IFG, we may eliminate or change the meaning of the non-terminal I/S nodes. However, the users who specify IFGs may wish to use a non-terminal or internal I/S node as a publication and/or subscription point. (e.g., the node labeled *Combined* in Figure 1.) To avoid losing such I/S nodes due to rewriting an IFG:

1. For each internal node that may be used as a publication point we add an explicit terminal node with an identity arc that connects to the internal node.
2. For each internal node that may be used as a subscription point we add an explicit terminal node with an identity arc from the internal node to the terminal node.

Thus, all publication and subscription points are represented by the terminal nodes of the IFG. All arcs and internal nodes can be subjected to rewriting rules and optimizations.

**Rewriting the entire IFG.**

Consider an IFG, G1, with all interesting publication and subscription points externalized. We can construct an equivalent IFG, G2, as follows. For each publication and subscription point in G1, add a like-named publication or subscription point to G2. For each possible pair (p,s) of publication and subscription points in G1, if there is a path from p to s in G1 consisting of arcs labeled with transforms and/or select operations:

p =>(ts1; ts2; ...; tsk) =>s

then add a single arc from p to s in G2 that is labeled with the (select;transform) pair of operations that is equivalent to (ts1; ts2; ...; tsk), as given by the above rewrite rules.

**Example**

Consider the IFG for stock services shown in Figure 2a. It is similar to Figure 1, integrating two independent stock markets *NYSE* and *NASDAQ* into the combined information space *Combined*. In this example, the messages from both sources must first undergo a lookup conversion (T1 and T2). A *capital* field is then added to each message which is the product of the number of shares in the trade and the price per share (T3). The new value-added information space *Large Trades* is derived from *Combined* by using a select operation (S1), which selects those trades involving over a million dollars.

As shown in Figure 2b, we split each externally visible I/S into two: one for publishers and another for subscribers, e.g., *Combined* is split into $\textit{Combined}_P$ and $\textit{Combined}_S$. This step is necessary to ensure that, after transform, all the advertised content is still available to dynamically joining publishers and subscribers.

Next, we identify all paths in the graph of Figure 2b to arrive at Figure 2c. For each path that has more than one select or transform, we then apply the rewrite rules so that 1) selects are moved before transforms and 2) a series of selects or a series of transforms are combined into a single instance of each. In this example, we simplify three such paths:

1. *NYSE* to *LargeTrades$_S$*: T1; T3; S, which reduces to:
   S1: (price*vol >= 1000000);
   T4: [issue, price, vol] => [com=NYS(issue), cap=price*vol]
2. *NASDAQ* to *LargeTrades$_S$*: T2; T3; S, which reduces to:
   S1: (price*vol >= 1000000);
   T5: [issue, price, vol] => [com=NAS(issue), cap=price*vol]
3. *Combined$_P$* to *LargeTrades$_S$*: T3; S. This reduces to:
   S1: (price*vol >= 1000000);
   T6: [com, price, vol] => [com, cap=price*vol]

With this, each path from a publisher to a subscriber is of the form select followed by transform, as shown in Figure 3.

The selects can now be implemented by an efficient content-based routing system, and the transforms performed before delivering to subscribers. Going one step further, we can combine the individually derived paths back into a single I/S, which may be implemented as a content-based publish/subscribe system. These paths can then be split by adding an additional select based on message source, then tagging the transforms with a source.

Before an event is delivered to a subscriber, it is transformed based on the I/S to which the client subscribed and the source of the message. Tagged transforms can be stored in a table for lookup and execution before the system delivers a message to a client. New subscriptions coming into any of the subscription points (*Combined* or *LargeTrades*) have their content filters modified based on the filter arcs out of the root into these spaces using a simple application of the rewrite rules.

## 4.2 Expanding State to Event Streams

Suppose a mobile client subscribes to the IBM events from the information space *MaxCurEv* of events equivalent to the events in *Combined* using the state *MaxCur* defined by the rule shown below:

$$[n, p], \langle n\text{: } p > maxP, curP \rangle \cup s \rightarrow \langle n\text{: } p, p \rangle \cup s$$

$$[n, p], \langle n\text{: } p \leq maxP, curP \rangle \cup s \rightarrow \langle n\text{: } maxP, p \rangle \cup s$$

(These are the identical rules discussed in the illustration of *collapse* in Section 2, except that we are ignoring the volume field v of events.)

Say that a number of events have been delivered to *Combined* and received by the client, who then disconnects. Suppose that at this point, the state in *MaxCur* is ⟨IBM: 160, 140⟩. While the client is disconnected, a long series of events is published, arriving at a new state ⟨IBM: 200, 120⟩. The mobile client then reconnects to the system. If the system is able to exploit the knowledge of the client's interpretation of event sequences, it should be able to deliver just the two events [IBM, 200] and [IBM, 120] rather than the much longer sequence of published events. The following table shows the original events, the generated state, and the compressed set of delivered events.

| | Original Events | States <issue: maxP,curP> | Delivered Events |
|---|---|---|---|
| | [IBM, 150] | <IBM: 150, 150> | [IBM, 150] |
| | [IBM, 160] | <IBM: 160, 160> | [IBM, 160] |
| | [IBM, 140] | <IBM: 160, 140> | [IBM, 140] |
| disconnect: | | | |
| | [IBM, 200] | <IBM: 200, 200> | |
| | [IBM, 180] | <IBM: 200, 180> | |
| | ... | ... | |
| | [IBM, 120] | <IBM: 200, 120> | |
| reconnect: | | | |
| | | | [IBM, 200] |
| | | | [IBM, 120] |

Given a state space *S*, a start state $s_0$ and a goal state *g* in *S*, and a *collapse* rule, the *expansion problem* is defined as the generation of the most economical sequence of events which, starting from $s_0$, yields *g*. The expansion problem can be converted into a shortest path graph search problem. We represent the states in *S* as vertices in a graph, and define each possible event transition as an edge. We then label these edges with a cost. For the purpose of this paper, we will assume each event has unit cost 1. Figure 4 shows a fragment of the state transition diagram for the above example (but for just IBM events -- thus, issue name has been left out of both events and states).

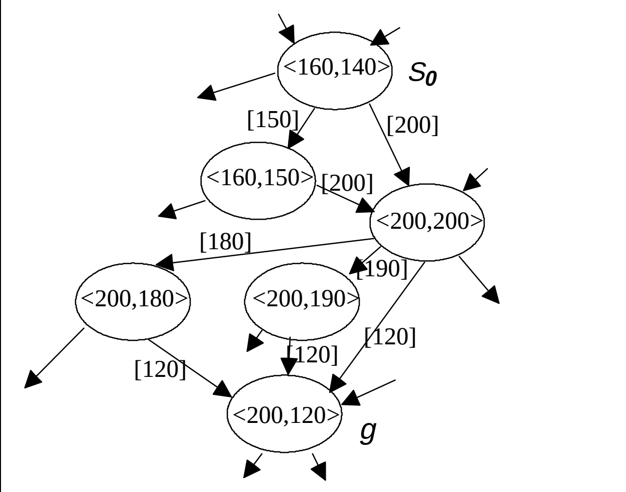

**Figure 4: State transition graph for collapse**

To solve the shortest path problem, we use the known A* algorithm [4]. This algorithm requires an estimator function *h,* where *h(s)* is a lower estimate of the shortest path from an arbitrary state *s* to state $s_0$. Working backwards from the goal state *g* toward $s_0$, we keep a set of candidate paths. We sort these paths based upon the actual length from of the path from *g* to *s* plus the estimated length *h(s)* from *s* to $s_0$. Beginning with the node *n* at the end of the best candidate path, of length *f(n),* we locate that neighbor *n'* of *n* that minimizes *(f(n)* + 1*)* + *h(n').* We extend the candidate path in the direction to *n'*. (We ignore other neighbors unless and until all candidates of at least this distance have been explored.)

The problem is to find suitable estimator functions *h(s).* One can obtain an estimator *h(s)* for a particular graph by constructing an exact solution for an extended graph with a strict superset of edges. We have developed a strategy for finding *h* for an important subclass of summarization functions — those which can be converted to the *replacement form* described below — by solving a hierarchy of problems.

Let us assume that the incremental formulation of the summarization function can be represented by a table such as the following:

| | s1 | s2 | s3 |
|---|---|---|---|
| e1(a,b,c) | a | | b |
| e2(a,b,c) | a | a | |
| e3(a,b,c) | | b | c |

Each row of the table corresponds to a particular collection of events meeting a particular condition, and having parameters, e.g. a, b, and c. Each column of the table corresponds to a component of the state. Each blank entry in the table indicates that the event in the corresponding row leaves the corresponding component of the state unchanged; each non-blank entry indicates that the event in the corresponding row replaces the corresponding component of the state. Not every summarization function can be put in this form; however many can, such as the stock example illustrated above.

If a summarization function is in replacement form, and it contains no rows such as the second in the example above, in which two or more columns are constrained to be replaced by the same value, then it is in *unconstrained* form. If a summarization function is in unconstrained form, and each row changes *k* columns, and for any *k* columns there is a row which changes those columns, then it is in *uniform unconstrained* form. The exact solution to a problem in uniform unconstrained form requires a number of events equal to *ceil(m/k),* where *m* is the number of state components in which the start and goal states differ. Any problem in unconstrained form but not uniform unconstrained form can be solved by extending it to uniform unconstrained form, and using the exact solution to the extended problem as an estimator for the original problem, and then applying A*. Similarly, any problem in constrained form can be extended to unconstrained form by assuming that all steps except the first (for which the constraints are known) may follow new rules in which additional parameters have been added as necessary to eliminate constraints.

The extension from constrained to unconstrained form, or from unconstrained to uniform unconstrained form is equivalent to adding edges to the state graph. The optimal solution to the extended problem therefore serves as an estimator for the original problem.

For example, the stock price example can be put into the form of a constrained problem as follows:

| | Maxprice | Curprice |
|---|---|---|
| p > Maxprice | p | p |
| p <= Maxprice | | p |

This problem can be solved by using the corresponding unconstrained problem as an estimator:

| | Maxprice | Curprice |
|---|---|---|
| p > Maxprice,q | p | q |
| p <= Maxprice | | p |

In this case, the estimation function is straightforward: the estimated distance from start to goal equals the number of issues for which the current state differs in cur price or max price from the goal state.

In this example (see Figure 4), finding a path from $s_0 = \langle 160, 140\rangle$ to state $g = \langle 200, 120\rangle$, the search is straightforward. Starting at goal state $\langle 200, 120\rangle$, we find that only the second row of the above matrix leads to a predecessor state, which has the form $\langle 200, *\rangle$. For any value of the second state except 200, the first row is blocked and therefore the estimated distance from the start state is 2. For the state $\langle 200, 200\rangle$ the estimated distance is 1. We therefore choose state $\langle 200, 200\rangle$ as the best candidate to continue the search. In fact, this state satisfies the conditions for firing row one of the matrix to reach a predecessor$\langle *, *\rangle$, so we can insert the start state $\langle 160, 140\rangle$.

## 5 Discussion

### 5.1 Related Work

Many concepts in Gryphon have been synthesized from a large base of results in group communication, databases, programming languages, and software engineering. As mentioned in Section 1, existing publish/subscribe technologies were our starting points.

The basic idea of representing system behavior in terms of the flow of data from inputs through functional modules to output, is used extensively in software design, see for example [7]. It is also common practice to allow software designers to depict the structure of a system using a high-level visual representation similar to data flow diagrams. The tool then converts the high-level representation into lower-level executable code.

Ideas for graph rewriting and analysis of data flow graphs have their origins in very early work in programming languages and code optimization in compilers [2].

Some systems commercially available today (e.g., NEON, [11]) claim to support arbitrary filtering and transforming operations. However, there is scant literature on the exact technical nature of these operations. It is not clear that these systems support a systematic approach to specifying the flow of events. Furthermore, there is no evidence that these systems support efficient routing of events by optimizing IFGs and mapping them to distributed broker networks.

### 5.2 Current Status and Future Work

We have developed a Gryphon system prototype that supports content-based publish/subscribe via efficient matching and multi-broker networks. We have also developed initial prototypes of tools for supporting stateless and stateful information flows. One tool supports the visual specification of information flow graphs and applies the rules of Section 3 to rewrite the graph. Another tool supports a restricted language to specify the meaning of event sequences, using which the tool generates equivalent, but shorter event sequences.

Several directions of work are ongoing and appear promising. Besides extending our graph rewriting techniques to encompass a more expressive language, we are working on ways to efficiently map optimized IFGs onto physical broker networks of various configurations. We are also incorporating protocols for reliable and ordered delivery within this framework of event distribution middleware.

We believe that stateful operations within IFGs are the next major step in the functionality of event distribution middleware. We are currently exploring the breadth of applicability of derived events and equivalent event sequences. Finally, we are beginning to deploy this new generation of event distribution middleware in real-world application integration scenarios.

## 6 Bibliography

[1] Marcos Aguilera, Rob Strom, Daniel Sturman, Mark Astley, Tushar Chandra. 1999. Proceedings of ACM Symposium on Principles of Distributed Computing, 1999, Atlanta, GA.

[2] Aho, A., Sethi, R., and Ullman, J. 1985. *Compilers, Principles, Techniques, and Tools.* Addison-Wesley publishing, Reading, MA.

[3] Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R., Sturman, D. 1999. "An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems", Proceedings of IEEE International Conference on Distributed Computing Systems '99, Austin, TX.

[4] Barr, A., and Feigenbaum, Edward A. 1986. *The Handbook of Artificial Intelligence.* Volume 1. Addison-Wesley Publishing, Reading, MA.

[5] K. P. Birman. "The process group approach to reliable distributed computing," pages 36-53, Communications of the ACM, Vol. 36, No. 12, Dec. 1993.

[6] Antonio Carzaniga, Architectures for an Event Notification Service Scalable to Wide-area Networks". Ph.D. Thesis. Politecnico di Milano. December, 1998. Available from http://www.cs.colorado.edu/~carzanig/papers/

[7] Ghezzi, C., Jazayeri, M., and Mandrioli, D. 1991. *Fundamentals of Software Engineering.* Prentice-Hall, Englewood Cliffs, NJ.

[8] John Gough and Glenn Smith. "Efficient Recognition of Events in a Distributed System," Proceedings of ACSC-18, Adelaide, Australia, 1995.

[9] Shivakant Mishra, Larry L. Peterson, and Richard D. Schlichting. Consul: A Communication Substrate for Fault-Tolerant Distributed Programs, Dept. of computer science, The University of Arizona, TR 91-32, Nov. 1991.

[10] Object Management Group. CORBA services: Common Object Service Specification. Technical report, Object Management Group, July 1998.

[11] New Era of Networks (NEON). http://www.neonsoft.com.

[12] Brian Oki, Manfred Pfluegl, Alex Siegel, Dale Skeen. "The Information Bus - An Architecture for Extensible Distributed Systems," pages 58-68, Operating Systems Review, Vol. 27, No. 5, Dec. 1993.

[13] David Powell (Guest editor). "Group Communication", pages 50-97, Communications of the ACM, Vol. 39, No. 4, April 1996.

[14] Bill Segall and David Arnold. "Elvin has left the building: A publish/subscribe notification service with quenching," Proceedings of AUUG97, Brisbane, Austrailia, September, 1997.

[15] Dale Skeen. Vitria's Publish-Subscribe Architecture: Publish-Subscribe Overview, http://www.vitria.com/