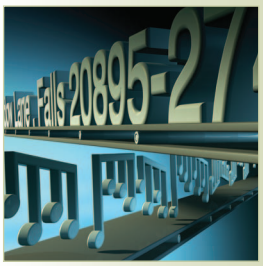

A Taxonomy of QoS-Aware, Adaptive Event-Dissemination Middleware

Event broker networks — scalable versions of the publish–subscribe paradigm — act as peer-to-peer overlays on broker nodes. Various frameworks support different overlay topologies and routing schemes for event dissemination, but attention is now turning to the nonfunctional attributes (such as quality of service) of such systems. Although many research efforts are starting to address the need for adaptation and QoS, no taxonomies or comprehensive surveys of adaptive middleware, which provide support for service guarantees, exist yet. To tackle this knowledge gap, the authors examine existing event-based middleware efforts, focusing on quality of service and adaptation.

Shruti P. Mahambre, Madhu Kumar S.D., and Umesh Bellur
Indian Institute of Technology Bombay

Middleware for event broker networks (EBNs) both alleviates the issues related to underlying platform heterogeneity and provides a uniform application interface. A common service interface that such middleware provides is *publish–subscribe*,[1] a paradigm in which producers publish information and consumers subscribe to it. In EBNs, information of interest is encapsulated in *events*; on such networks, middleware stores and manages subscriptions as well as routes events. This model's fully decoupled nature in terms of time, space, and synchronization[1] makes it highly suitable for large-scale distributed applications. Middleware for event broker networks is also known as event-dissemination middleware (EDM).

To disseminate events, an EBN overlay network[2] uses a subset of the underlying existing physical network (an overlay link itself is virtual but could consist of several physical links to the underlying network). Therefore, middleware must execute on each overlay node that has to work cooperatively to accomplish its functions. Such event-based architectures appear in various domains, many of which themselves require quality-of-service (QoS) guarantees from the underlying infrastructure. However, our survey of existing efforts in research-and-industry-based event dissemination middleware

1089-7801/07/$25.00 © 2007 IEEE

Related Work in Information Dissemination

In building our taxonomy, we first researched all the existing work we could find on event dissemination. The taxonomy presented by Rene Meier and colleagues[1] came closest to what we were looking for: it deals with the classification of existing event-based systems as a programming paradigm and identifies fundamental properties of event-based middleware. Their taxonomy then classifies middleware broadly based on *event model* and *event service* criteria, with a further classification on the basis of the middleware's functional and nonfunctional features and organization and interaction model.

Although we reviewed core functionality, our focus is on quality-of-service guarantees and adaptation in middleware rather than with programming models for functional aspects of event-dissemination middleware (EDM). We also extend Meier and colleagues' definition of an event model to a *communication model* by taking into account event characteristics such as attributes and hierarchical and compositional relationships between event types. Patrick Eugster and colleagues[2] also present a detailed survey of event-based systems based on a common denominator of the various modes of decoupling in asynchronous systems. Although they discuss decoupling in three dimensions (time, space, and synchronization), their main focus is on implementing the issues underlying publish–subscribe schemes. Their survey forms the basis of our work. Bruce Martin and colleagues[3] present a taxonomy as a hierarchy of questions and answers about the features of distributed computing systems (DCS). Although their taxonomy isn't specific to the publish–subscribe paradigm, its comparison of existing general-purpose DCSs provides a means of classifying systems and also serves as a basis for designing a DCS that offers novel feature combinations.

References

1. R. Meier and V. Cahill, "Taxonomy of Distributed Event-Based Programming Systems," *The Computer J.*, vol. 48, no. 5, 2005, pp. 602–626.
2. P.Th. Eugster et al., "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, 2003, pp. 114–131.
3. B.E. Martin, C.H. Pedersen, and J. Bedford-Roberts, "An Object-Based Taxonomy for Distributed Computing Systems," *Computer*, vol. 24, no. 8, 1991, pp. 17–27.

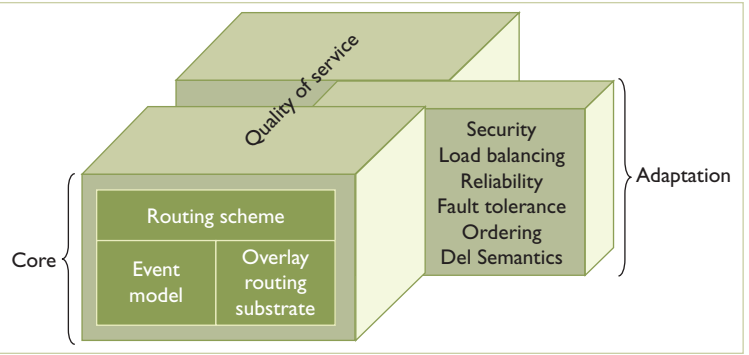

Figure 1. Event-based architecture. Here, adaptive middleware offers quality-of-service support.

indicates significant gaps in efforts to address these needs; in fact, we found that few middleware options[3–5] provide support for nonfunctional service guarantees. Because a comprehensive survey of current options doesn't yet exist, this article offers a review of existing event-based systems, classified with a taxonomy of adaptive, event-based middleware that provides QoS guarantees.

Middleware Architecture

As Figure 1 shows, event-based middleware has a layered architecture. The *core* comprises mandatory functional features, such as the event model, the subscription scheme used to disseminate events, and the overlay routing substrate. The layer on top of the core forms a set of optional *services* – that is, the nonfunctional QoS guarantees the middleware provides, such as reliability, delivery semantics, message ordering, security, and fault tolerance. QoS guarantees are orthogonal to middleware-supported core functions; the middleware's ability to reconfigure itself to ensure and maintain a QoS agreement is its *adaptability*, which can result in changes to the core's subscription scheme or overlay routing substrate.

The taxonomy we developed as part of our survey efforts provides a hierarchy of the identifying features that an event-based system requires for service guarantees. Specifically, it helps us identify groups of systems and lays out the functional characteristics that come into play when providing QoS guarantees and support.

The Core

As Figure 2 shows, our taxonomy begins with a differentiation between mandatory functional features and QoS properties; as mentioned in the previous section, we can break the core down further into an event model, overlay routing substrate, and subscription scheme.

Event Model

In any EDM, subscribers transmit subscriptions in the overlay network (filters sift out any nonmatching data elements from the event notifica-

tions), and publishers float advertisements about the occurrence of a new event type.

An event model has three message types — *advertisement*, *notification*, and *subscription*. The model defines the attributes that uniquely identify an event based on that event's name, type, or temporal or persistent nature. An event can also be characterized by the kind of data it stores — for example, the data contained in an event could have private or public access-control information associated with it. Hermes[3] uses an object model that represents every event as an object whose attributes in turn define the event's properties. Jedi[5] uses pattern (or string) matching to identify the events that occur in the system, whereas Siena[6] models an event in the form of a triplet: `<name,type,value>`.

Events can also be related to one another by a hierarchical or compositional relationship as shown in Figure 3.

Event hierarchy. A new event type can inherit from an existing event type, thus extending it and eventually building up a hierarchy and enabling polymorphic dispatch of events in the system. A subscriber who subscribes to this sort of super type receives notifications from all the event's subtypes, as well as from the super type itself. DAC[7] supports a topic-based event hierarchy — that is, events in DAC are represented in the form of topics. Subscribers express their interest in these topics and receive notifications about them along with notifications for all their subtopics. In Hermes, every rendezvous node (which is any node in the overlay network at which subscriptions and notifications converge) is aware of its descendent types, so inheritance routing supports any subscriptions to parent rendezvous nodes.

Event composition. Composite events[3] provide a higher level of abstraction for subscribers by enabling complex event patterns; essentially, subscribers don't have to subscribe to all the primitive events that make up the pattern and then perform the detection of the composite themselves. Composite events can be either temporal or spatial in nature: temporal composite events are based on time dependencies between primitive events, whereas spatial composite events are a conjunctive–disjunctive combination of individual events, published based on a pattern of subscriptions observed by the composite event-detection engine. Hermes[3] supports a composite event service.

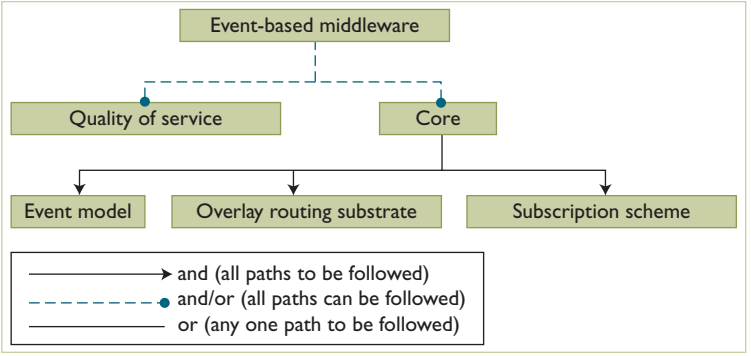

Figure 2. The core. This part of the taxonomy focuses on the middleware's functional features.

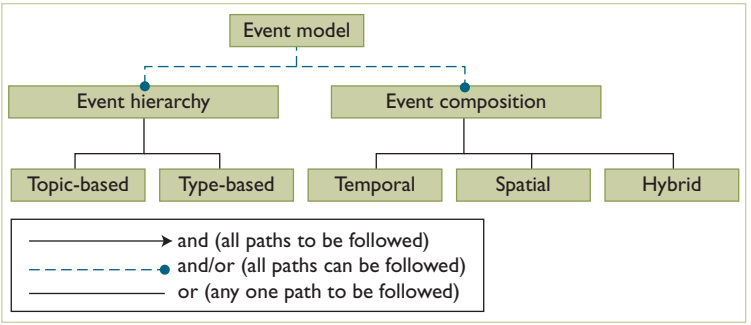

Figure 3. Event model taxonomy. This is the basis for classifying event models in event-dissemination middleware.

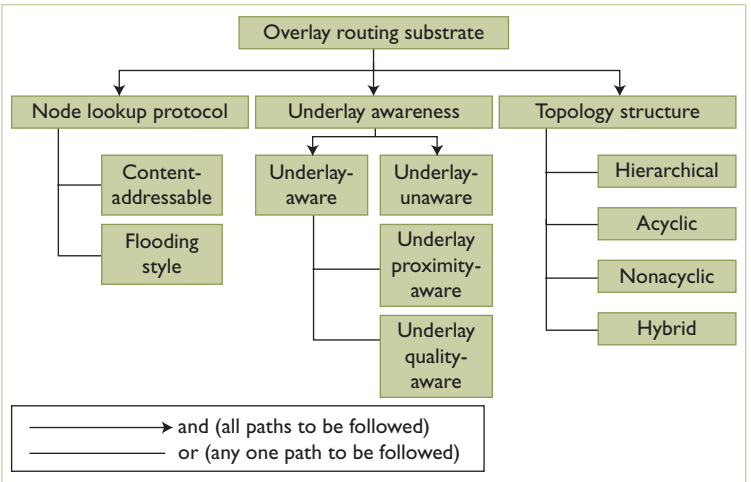

Figure 4. Overlay routing substrate. These three properties form the basis for categorizing overlay routing substrates

Overlay Routing Substrate

As Figure 4 shows, we can categorize EDMs on the basis of their overlay routing substrate, which provides an IP-address-independent abstraction on a physical underlay. Middleware architecture can support one or multiple overlay network topolo-

gies, which are useful for providing service guarantees. Such networks can also be deployed by using end hosts that run overlay protocol software that isn't dependent on routers or ISPs.

Our taxonomy further categorizes overlay substrates based on one of three properties: node lookup protocol, underlay awareness, and topological structures.

Node lookup protocol. By using the node lookup protocol, we can classify overlay networks into one of two types:[8]

- *Content-addressable overlays*, in which network topology and object content are related. Distributed hash table- (DHT-) based techniques[9] convert object identifiers into overlay node identifiers, thus guaranteeing object location (if the object is present); search times are bounded.
- *Flooding-style overlays*, in which object lookup is based on time-to-live (TTL) controlled flooding. A node looking for a particular key value sends a request to all its known neighbors with a specific TTL value. The neighbors reply to the request by looking for a key match in their local databases; if they find one, they reply. Otherwise, they forward the request to *their* known neighbors and increase the hop count. If the hop count passes the TTL value, the forwarding stops. Unless the TTL is set to a very high value, the object might not be found, even if it's present.

Pastry[10] and Chord[11] are content-addressable overlays, and Freenet[12] is an example of flooding-style overlays.

Underlay awareness. The overlay network can be either aware or unaware of its underlying physical network topology:

- *Underlay-unaware overlays* are based on DHTs, which use logical identifiers to identify overlay nodes and their neighbors. The overlay assumes full network connectivity and doesn't measure physical network properties in the overlay construction process. Chord[11] constructs underlay-unaware overlays.
- *Underlay-aware overlays* maintain correspondence between physical and overlay networks. The overlay construction strategy considers some of the underlying physical network's properties, including path disjointedness (absence of common nodes or physical links in different overlay paths), hop count (number of physical nodes between a pair of overlay nodes), bandwidth, physical distance information, and failure statistics. This makes the overlay sensitive to changes in the underlying physical network — that is, the overlay dynamically adapts to resource-based adaptation triggers.

We can further split underlay-aware overlay networks into *underlay-proximity-aware* and *underlay-quality-aware* overlays. Underlay-proximity-aware overlays reflect only the physical nodes' network proximity — when adding a new node to an overlay, for example, a node chooses the nearest neighbor based on the round-trip time of a ping message broadcast to all nearby nodes. No other underlay parameter is used in overlay construction and maintenance; Pastry,[10] Medym,[13] and Hermes[3] belong to this category.

With underlay-quality-aware overlays, we extend the granularity of underlay awareness to physical link quality, failure probabilities, node degree, physical network diameter, and QoS guarantees. An underlying link's failure statistics information, for example, can help us select a reliable path for an overlay link. Information regarding the degree of physical nodes is useful when constructing an overlay because it emphasizes the degree of availability. To the best of our knowledge, no currently reported work uses underlay-quality-aware-overlay networks in event-based middleware.

Topological structure. As part of our research, we also identified the topologies on which most of today's event-based middleware are founded. These topologies determine the formation of event-dissemination trees when routing events. All event-based middleware studied in this survey adhere to one or a combination of the following topologies:[6]

- *Hierarchical* — nodes are connected in a hierarchy of parent–child relationships, hence they form a tree-like structure. Each server in the topology has several clients that could be publishers, subscribers, or intermediate brokers. The parent server can receive notifications, advertisements, and subscriptions from all of its clients, but will only send back notifications. Jedi[5] uses a hierarchical overlay topology for event dispatch.

- *Acyclic* — the communication between servers is peer to peer. The links between servers aren't direct, so server connections produce an acyclic graph that enables bidirectional flow of subscriptions, advertisements, and notifications.
- *Nonacyclic* — removes the constraint of an acyclic graph from acyclic topology, which enables bidirectional communication between two servers. (Multiple paths can also exist between servers.) The topology is robust and resilient to failure, but could also lead to cycles.
- *Hybrid* — supports a combination of topologies. Siena[6] is a good example; some clusters of subnets in Siena have very intense local-event traffic. Visibility of events outside this cluster is very low, hence a generic graph topology is preferred inside the cluster and an acyclic graph topology at the higher level.

Figure 5 shows how the various structural topologies connect to clients and servers. Our taxonomy could be extended to accommodate newer topologies.

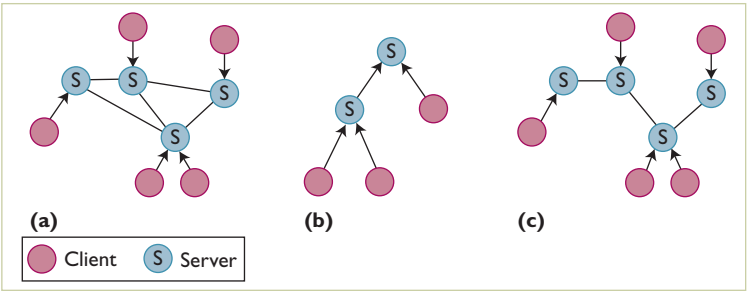

Figure 5. Structural topologies.[6] Note the different connections among nonacyclic, hierarchical, and acyclic topologies.

Subscription Scheme

Subscribers must follow certain semantics — called a subscription scheme — when specifying their interest in an event; ditto publishers, when publishing or advertising an event. In a *topic-based subscription scheme*, participants publish events and subscribe to individual topics, each of which is identified by a keyword. Topics can overlap, leading to a hierarchy as in DAC, but this differs from a *subject-based scheme*, which was introduced by TIBCO (www.tibco.com) and assumed a static subject hierarchy with paths described in dot notation. Java Messaging Service (JMS; http://java.sun.com/products/jms/docs.html) uses a topic-based subscription scheme because it's much more flexible in its use of predicates.

In a *content-based subscription scheme*, subscriptions are based on the event's actual content, which enables fine-grained search. Event properties could be internal attributes of the data structures that carry the events; Jedi,[5] Siena,[6] and Elvin[14] all use a content-based subscription scheme. IBM's Websphere MQ (www.ibm.com/redbooks) is a commercial message-queuing middleware platform that supports both content- and topic-based subscriptions. In JMS, clients can address messages to a topic.

In a *type-based subscription scheme*, events are classified according to their type,[15] which ties together commonalities in structure and content as well as facilitates a closer integration between language and middleware. IndiQoS[4] and Hermes support type-based routing schemes. A somewhat related approach (one that's concept based[16]) enhances the event-notification service by enabling it to pass semantic information across components. A notification service that uses this sort of concept-based approach can thus deliver ready-to-process notifications to subscriber applications that require no further processing. Finally, as its name suggests, the *hybrid subscription scheme* combines some portions of all these routing schemes.

Quality of Service

As we saw in Figure 1, event-based middleware can optionally provide support for QoS guarantees. These service guarantees typically follow a publisher-offered, subscriber-requested pattern. The EBN is tasked with ensuring that quality constraints are met during event delivery. As Figure 6 shows, we've identified five classes of service guarantees: latency, bandwidth, reliability, delivery semantics, and message ordering.

Latency

Also known as *time to delivery*, latency is defined as the time from when a publisher publishes an event and a subscriber to that event receives notification that it's available. The overlay network must effectively reduce the overall latency of event notifications in an EBN. If T_d represents transmission delay, P_d represents propagation delay (which is constant for a network), Q_d represents buffering/queuing delay, e represent an event of type τ, and b is the total number of nodes occurring in the path between a publisher and subscriber, then the latency L for a single notification of an event of type τ between a publisher and subscriber is as follows:

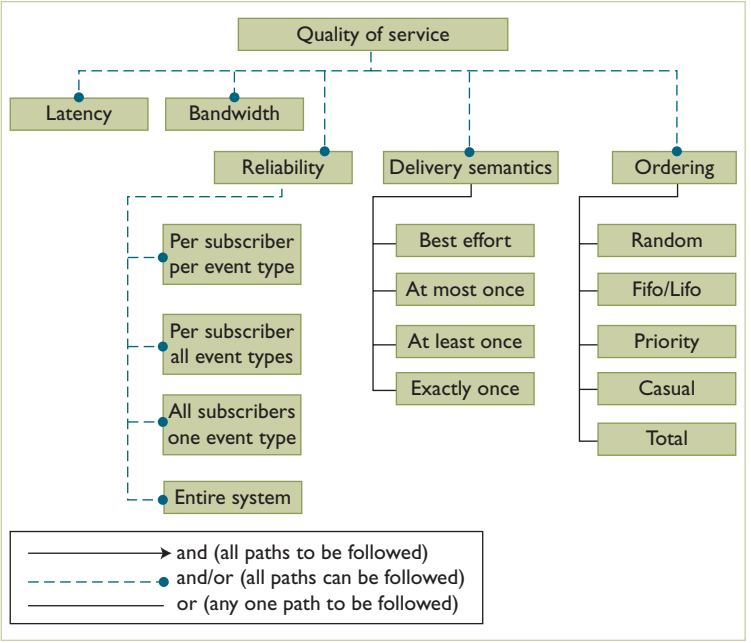

Figure 6. Quality of service. Event-based middleware can provide support for five classes of QoS guarantees: latency, bandwidth, reliability, delivery semantics, and message ordering.

$$\mathrm{L}(e_\tau) = b(P_d) + \sum_{i=1}^{n} \left(T_d[i] + Q_d[i]\right). \quad (1)$$

We calculate the latency as the summation of the delay in propagation at all broker nodes and the transmission and queuing delay at each node's buffer.

Bandwidth

Bandwidth represents the resources available across a path during event transfer, which is denoted by the number of events transferred between the publisher and subscriber per time unit. If a subscriber doesn't specify a requirement, then the broker network assumes default values, which provide the maximum possible bandwidth available along a path. IndiQos[4] provides support for latency and bandwidth parameters by using netpipe, an infopipe that transfers events between different hosts (infopipes are software components connected to each other to form an overlay network). The netpipe's length represents latency, and its width represents maximum bandwidth.

Reliability

The reliability attained at a subscriber node in the EBN is the ratio of the notifications a subscriber receives for an event type to the number of event types published.[17] If $n(e_\tau)$ is the number of notifications received by a subscriber s for an event type, and $p(e_\tau)$ is the number of publications of an event type τ, then reliability R attained by subscriber s is

$$\mathrm{R}\left[e_\tau^s\right] = \frac{n(e_\tau)}{p(e_\tau)}. \quad (2)$$

Using Equation 2, we can measure reliability at different levels in an event-based system, including per subscriber per event type, per subscriber for all event types, all subscribers for a single event type, and the entire system. Detailed definitions appear elsewhere.[17]

Delivery Semantics

Delivery semantics come into play at the last hop (just before notification reaches a subscriber) and depend on two values, network reliability and support for duplicate messages. The lowest level of a delivery guarantee is *best effort*, which means no guarantee of reliability, and events can be duplicated — that is, notifications are sent without guarantee of delivery and with duplication. The *at most once* delivery semantic ensures that the subscriber receives a maximum of one notification of an event type instance; if the subscriber doesn't receive any notification at all, then the EBN isn't reliable. The *at least once* delivery semantic ensures that the subscriber receives at least one notification of an event type instance (it can still receive multiple notifications of the same instance, though). Finally, the *exactly once* delivery semantic, ensures that the subscriber receives a notification from exactly one event type instance, but it could retain duplicates at intermediate nodes to guarantee reliability. Gryphon[18] follows exactly once delivery semantics, even during periods of disconnection.

Message Ordering

An EBN can adopt a default temporal message ordering in the form of first in/first out (FIFO) or last in/first out (LIFO). If Θ is a time stamp associated with events e of type τ_i, τ_j in the system, and n is a notification message, then FIFO ordering is expressed as shown in Equation 3, and LIFO message ordering is expressed as in Equation 4:

$$\Theta\left[e(\tau_i)\right] \prec \Theta\left[e(\tau_j)\right] \Rightarrow n\left[e(\tau_i)\right] \rightarrow n\left[e(\tau_j)\right] \quad (3)$$

$$\Theta\left[e\left(\tau_i\right)\right] \prec \Theta\left[e\left(\tau_j\right)\right] \Rightarrow \neg\left(n\left[e\left(\tau_i\right)\right] \rightarrow n\left[e\left(\tau_j\right)\right]\right). \quad (4)$$

If the notifications aren't ordered, we assume that the default ordering is either *random* or *unordered*. The EBN can also follow *total ordering*. In this case, if an event type notification is meant for multiple subscribers, then all the event type's notifications should be delivered in the same order to all subscribers. Assuming that n and n' are notifications of events received at subscribers s and s' for event types τ_i and τ_j, we get[18]

$$n\left[e\left(\tau_i\right)\right] \prec n\left[e\left(\tau_j\right)\right] \Rightarrow \neg\left(n'\left[e\left(\tau_j\right)\right] \prec n'\left[e\left(\tau_i\right)\right]\right). \quad (5)$$

Causal ordering is required if the order between events is defined on the basis of a relative cause-and-effect relationship (all events in Jedi are causally ordered). We can express it as

$$e\left(\tau_i\right) \rightarrow e\left(\tau_j\right) \Rightarrow n\left[e\left(\tau_i\right)\right] \rightarrow n\left[e\left(\tau_j\right)\right]. \quad (6)$$

Message *prioritization* is a form of ordering in which the publisher assigns priority numbers to events when published. If p denotes the publication of event e of type τ_i, τ_j, for example, then we would express prioritization in an event-based system as

$$p\left[e\left(\tau_i\right)\right] < p\left[e\left(\tau_j\right)\right] \Rightarrow n\left[e\left(\tau_i\right)\right] \rightarrow n\left[e\left(\tau_j\right)\right]. \quad (7)$$

If event τ_i, has a higher priority than event τ_j, then notification of event τ_i will be sent before event τ_j.

QoS Support in EBMs

As the "Classification of Event-Based Middleware Based on Core Properties" sidebar highlights, most existing middleware doesn't support, or provides limited support for, service guarantees. Typically, latency is assumed to be a default QoS parameter during event notification; IndiQos is the only option that explores latency and bandwidth as QoS parameters in detail.

Most approaches support reliability, message ordering, and delivery semantics on a limited scale, even in industry middleware projects. The JMS API ensures the *exactly once* delivery semantic and provides for different levels of reliability for those applications that can afford to drop messages or receive duplicates. IBM's Websphere MQ claims to be reliable, fault tolerant, scalable, and secure. Preliminary work has explored QoS parameters in a publish–subscribe domain,[19] but neither option formally defines any parameters. The routing algorithms in Hermes use built-in fault tolerance features that enable event brokers to recover after failure, but Hermes doesn't provide support for client-specified reliability as a service guarantee.

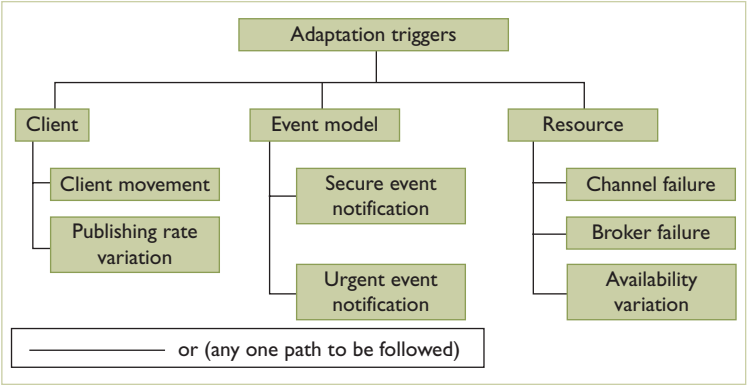

Figure 7. Adaptation triggers in middleware. The event-dissemination middleware adapts to client-related adaptation trigger events (client mobility), event-model-based trigger events (notification of an urgent event), and resource-based trigger events (broker or channel failure).

Adaptation in Middleware

Certain events that occur after application deployment can result in the violation of committed service guarantees, forcing the middleware to execute a series of actions to ensure committed service guarantees. Figure 7 shows how we classify these *adaptation trigger events*, which involve the EDM adapting to

- client-related trigger events (client mobility),
- event-model-based trigger events (notification of an urgent event), and
- resource-based trigger events (broker or channel failure).

Client-mobility-triggered events occur when a "disconnect" operation is followed by a "reconnect" operation. During the period of disconnection, multiple subscriptions from different subscribers arrive at the broker node to which the publisher was originally registered. Once the publisher reconnects to a new broker node in the network, these subscriptions are forwarded to that new broker, which in turn triggers changes in intermediate brokers' routing tables and leads to traffic-load variations. Similar issues arise when a subscriber disconnects from one broker and reconnects to another one somewhere

Classification of Event-Based Middleware Based on Core Properties

Our survey aims to classify event-dissemination middleware (EDMs) based on their functional characteristics and the support they provide for quality-of-service (QoS) parameters. We picked the most popular EDMs available today as a basis for this classification; Table 1 classifies those efforts based on the functional characteristics required for architecting such middleware.

Jedi, Siena, Hermes, and IndiQoS provide partial support for some of the service guarantees we identify in the main text. Siena doesn't provide support for any QoS parameters, but describes an event model, comprising single or multiple event servers, with different overlay topologies for routing events. Table 2 gives us a clear insight into the current level of support provided by existing middleware for service guarantees and adaptation.

Table 1. Classification of event-based middleware.

Middleware	Event model	Overlay routing substrate	Subscription scheme
Elvin	Events represented as strings; no event composition or hierarchy	Underlay-unaware topology (static)	Content based
Jedi	Object based (active objects and event dispatcher); no event hierarchy or composition	Underlay-unaware hierarchical event dispatcher (static)	Content based; pattern matching of events
Siena	Pattern-based model in the form of a triplet; no event hierarchy or composition	Underlay-unaware acyclic, peer-to-peer, hierarchical, or hybrid topology (static)	Content based
Gryphon	No event hierarchy or composition	Underlay aware; hierarchical structure, with publisher hosting broker as root and subscriber hosting broker as leaf	Content based
Hermes	Object based; supports event hierarchy and composition	Underlay aware; hybrid (static)	Hybrid of type based and type and attribute based
Rebeca	No event hierarchy or composition	Underlay unaware; hierarchical, symmetrical, and acyclic-tree-like topology	Content based with covering and merging of filters
Medym	No event hierarchy or composition	Underlay-aware overlay (proximity)	Content based
IndiQoS	Object based with QoS profiles; no event hierarchy or composition	Underlay aware; hybrid (static)	Type based

Table 2. Classification of middleware based on support for quality of service and adaptation.

Event-based middleware	Elvin	Jedi	Siena	Gryphon	Hermes	Rebeca	Medym	IndiQoS
QoS-supported parameters								
Latency	No	No	No	No	No	No	No	Yes
Bandwidth	No	No	No	No	No	No	No	Yes
Reliability	No	No	No	No	Yes	No	Yes	No
Delivery semantics	No	No	No	Yes	No	No	No	No
Message ordering	No	Yes	No	No	No	No	No	No
Adaptation applicable at level								
Event model	No	No	No	No	No	No	No	No
Overlay routing substrate	No	No	No	No	Yes	No	Yes	No
Subscription scheme	No	No	No	No	Yes	No	Yes	No
Adaptation triggers								
Client movement	No	Yes	Yes	Yes	No	No	No	No
Publishing rate variation	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Secure event notification	No	No	No	No	No	No	No	No
Urgent event notification	No	No	No	No	No	No	No	No
Link failure	No	No	No	Yes	No	No	Yes	No
Broker failure	No	No	No	No	Yes	No	Yes	No
Availability variation	No	No	No	No	No	No	No	No

Table 1. Classification of quality of service and adaptation triggers.

QoS parameters	Latency	Bandwidth	Reliability	Ordering	Delivery semantics
Adaptation trigger (client mobility)					
Client movement	Yes	Yes	Yes	No	No
Publishing rate variation	Yes	Yes	Yes	No	No
Adaptation trigger (event model)					
Secure event notification	No	No	No	No	No
Urgent event notification	Yes	No	No	Yes	No
Adaptation trigger (resource variability)					
Link failure	Yes	Yes	Yes	No	No
Broker failure	Yes	Yes	Yes	No	No
Availability variation	Yes	Yes	Yes	No	No

else in the network. Modern distributed applications allow simultaneous mobility of both subscribers and publishers, but this also triggers changes and necessitates adaptive behavior on the EDM's part. The service guarantees affected here include latency, bandwidth, and reliability.

Changes in event frequency can increase or decrease traffic on communication channels as well. If an event's frequency of occurrence increases beyond a threshold (the buffer space available at the broker node or the communication channels' bandwidth), then buffer overflows, traffic congestion, or communication delay can occur; this is commonly called *load unbalancing*. Similarly, when the publishing rate decreases, some channels can become relatively free, which means they can invite more traffic through those links; this is commonly called *load redistribution*. This trigger also affects latency, bandwidth, and reliability.

Adaptation triggers related to an event model typically fall into one of two categories: secure event notifications and urgent event notifications. In the former, event notifications defined as secure are routed through specific brokers only, which have defined encryption rules. In the latter, an event can have a TTL value associated with it that says event notification is valid only if delivered to the client during the TTL period. Both of these event notifications trigger the discovery of express paths (or alternate paths that can route the event within a specified TTL). The service guarantees that event-model triggers affect are latency and event ordering.

Adaptation triggers related to changes in resource availability also fall into one of two categories: link or broker failure and resource availability variation. If a link or broker fails, the result is no delivery of the events routed through that link or broker. Once the failure is detected, the middleware must establish alternate paths to the disrupted destinations. The service guarantees that this trigger affects are latency, bandwidth, and reliability. Resource availability variations, such as introduction of a high-bandwidth link, an increase in a broker node's buffer size, and withdrawal of a high-speed link in the physical network, might change the event-dissemination trees that were based on the resource availability at that point in time. The service guarantees this trigger affects are latency, bandwidth, and reliability.

Table 1 summarizes the QoS parameters directly affected as a result of adaptation triggers in event-based middleware.

It's apparent from this survey that providing service guarantees in event-based middleware is still in its nascent stages: most existing middleware don't fully support any of the QoS parameters we identified. Although researchers have studied some parameters (such as latency) individually, no one has yet identified and resolved the issues that arise from the interplay of QoS parameters.

Another area rich in research potential appears to be that of dynamic adaptation. This area involves challenges such as identifying the triggers that cause adaptation, dynamically reconfiguring the topology to ensure service guarantees, extending event-dissemination algorithms, and exploring client mobility in more detail. Adaptive load distribution and resilience to path outages in EBNs are also topics that have been touched on only lightly so far.

Our ongoing work deals with the development of EDM that adapts to resource failure triggers and

provides reliable routing in the face of such failures. We're also putting together a comprehensive QoS model for broker networks. An EDM with these adaptive and reliable properties would be immensely useful for the development and deployment of self-stabililizing, distributed, asynchronous applications on the Internet.

References

1. P.Th. Eugster et al. "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, 2003, pp. 114–131.
2. P. Pietzuch and J. Bacon, "Peer-to-Peer Overlay Broker Networks in an Event Based Middleware," *Proc. 2nd Int'l Conf. Distributed Event Based Systems*, IEEE CS Press, 2003, pp. 1–8.
3. P. Pietzuch, *Hermes: A Scalable Event-Based Middleware*, PhD thesis, Queens College, Univ. of Cambridge, UK, 2004.
4. F. Araujo and L. Rodrigues, "On QoS-Aware Publish Subscribe," *Proc. 22nd Int'l Conf. Distributed Computing Systems Workshops*, IEEE CS Press, 2003, pp. 511– 515.
5. G. Cugola, E.D. Nitta, and A. Fuggetta, "The JEDI Event-Based Infrastructure and Its Application to the Development of OPSS WFMS," *IEEE Trans. Software Eng.*, vol. 27, no. 9, 2001, pp. 827–850.
6. A. Carzaniga, *Architectures for an Event Notification Service Scalable to Wide-Area Networks*, PhD thesis, Dept. of Computer Science, Politecnico di Milano, Italy, 1998.
7. P.Th. Eugster, R. Guerraoui, and J. Sventek, "Distributed Asynchronous Collections: Abstractions for Publish/Subscribe Interaction," *Proc. 14th European Conf. Object-Oriented Programming*, Springer-Verlag, 2000, pp. 252–276.
8. D. Doval and D. O'Mahony, "Overlay Networks: A Scalable Alternative for P2P," *IEEE Internet Computing*, vol. 7, no. 4, 2003, pp. 79–82.
9. S. Rhea et al., "OpenDHT: A Public DHT Service and Its Uses," *Proc. 2005 Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm.*, ACM Press, 2005, pp. 73–84.
10. A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. 18th IFIP/ACM Int'l Conf. Distributed Systems Platforms*, ACM Press, 2001, pp. 329–350.
11. I. Stoica et al., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM 01 Conf.*, ACM Press, 2001, pp. 149–160.
12. I. Clarke et al., "FreeNet: A Distributed Anonymous Information Storage and Retrieval System," *Proc. Int'l Workshop on Design Issues in Anonymity and Observability 2000*, LNCS 2009/2001, Springer-Verlag, 2000, pp. 46–66.
13. F. Cao and J.P. Singh, "MEDYM: Match-Early and Dynamic Multicast for Content-Based Publish-Subscribe Service Networks," *Proc. 6th Int'l Conf. Middleware*, LNCS 3790, 2005, pp. 292–313.
14. I. Clarke et al., "Elvin Has Left the Building: A Publish/Subscribe Notification Service with Quenching," *Proc. AUUG Technical Conf. '97*, 1997; www.elvin.org/papers/auug97/auug97.html.
15. P.Th. Eugster, R. Guerraoui, and C.H. Damm, "On Objects and Events," *Proc. 16th ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications*, ACM Press, 2001, pp. 254–269.
16. M. Cilia et al., "Dealing with Heterogeneous Data in Pub/Sub Systems: The Concept-Based Approach," *Proc. 3rd Int'l Workshop on Distributed Event-Based Systems*, IEE Press, 2004, pp. 26–31.
17. S.P. Mahambre and U. Bellur, "Reliable Routing of Event Notifications over P2P Overlay Routing Substrate in Event Based Middleware," to be published in *Proc. 4th Int'l Workshop on Hot Topics in P2P Systems* (IPDPS 07), IEEE Press, 2007.
18. S. Bhola et al., "Exactly-Once Delivery in a Content-Based Publish-Subscribe System," *Proc. Int'l Conf. Dependable Systems and Networks*, IEEE CS Press, 2002, pp. 7–16.
19. S. Behnel, L. Fiege, and G. Muhl, "On Quality-of-Service and Publish/Subscribe," *Proc. 5th Int'l Workshop Distributed Event-Based Systems*, IEEE CS Press, 2006; www.kbs.cs.tu-berlin.de/publications/fulltext/DEBS2006_1.pdf.

Shruti P. Mahambre is a PhD student in the Department of Computer Science and Engineering at the Indian Institute of Technology, Bombay. Her research interests include QoS issues in event-based middleware, P2P overlay networks, and publish–subscribe models. Mahambre is a student member of the IEEE Computer Society and the Computer Society of India. Contact her at shruti@cse.iitb.ac.in.

Madhu Kumar S.D. is a senior lecturer in the Department of Computer Engineering at the National Institute of Technology Calicut, India. His research interests are in distributed computing and middleware systems. Kumar is currently pursuing a PhD in computer science and engineering at IIT Bombay. He is a life member of the Indian Society for Technical Education, a member of the IEEE, and an associate member of the Computer Society of India. Contact him at madhu@cse.iitb.ac.in.

Umesh Bellur is an associate professor in the Department of Computer Science and Engineering at IIT Bombay. His research interests include distributed systems such as those created by P2P overlays, sensor networks, and application networks, semantic matchmaking algorithms in service-oriented architectures, and autonomic computing. Bellur has a PhD in computer engineering from Syracuse University. He is a member of the ACM and the IEEE. Contact him at umesh@cse.iitb.ac.in.