Event Systems
How to Have Your Cake and Eat It Too

P. Th. Eugster[1], P. Felber[2], R. Guerraoui[1], S. B. Handurukande[1],
[1] Distributed Programming Laboratory,
Swiss Federal Institute of Technology in Lausanne.
[2] Bell Labs, Lucent Technologies, 600 Mountain Ave,
Murray Hill, NJ 07974, USA.

Abstract

This paper addresses the fundamental tradeoffs in event systems between scalability *(of event filtering, routing, and delivery mechanisms),* expressiveness *(when describing interests in events), and* event safety *(ensuring encapsulation and type-safe interaction with polymorphic events). We point out some ramifications underlying these tradeoffs and we propose a pragmatic approach to handle them. We achieve scalability using a multi-stage filtering strategy that combines approximate and perfect matching techniques for the purpose of event routing and filtering. We achieve expressiveness and event safety by viewing events as objects, i.e., instances of application-defined abstract types.*

1 Introduction

The design and implementation of event systems have been an active field of research over the last few years. These systems have evolved from simple multicast-oriented topic-based systems (e.g., [Cor99, Ske98, TIB99, AEM99]) to elaborate, content-based, systems that filter and disseminate data events according to their content (e.g., [SAB+00, CRW00, SBCea98, CNF98, MÖ1, SCG01]). Content-based dissemination techniques permit accurate addressing of events to selected subscribers according to their interests.

Event systems do however face fundamental tradeoffs while attempting to satisfy several requirements made on them. First, these systems must *scale* to a potentially large number of subscribers (hundreds of thousands), subscriptions (millions), and events (hundreds per second). Second, they should provide *expressive* mechanisms to precisely specify the interests of subscribers, in order to avoid having these receive irrelevant events. Third, the event model must be *safe* enough to permit exchange of arbitrary information concealed behind application-defined abstract types, i.e., without revealing implementation details (preserving encapsulation) or requiring explicit marshaling and unmarshaling (enforcing type safety) of this information.

Until now, event systems have been focusing only on parts of the equation, such as scalability and expressiveness [CRW00], and we are not aware of any system that provides (even partial) support for all three aspects, namely *scalability*, *expressiveness*, and *event safety*. In fact, the tradeoffs are a consequence of an underlying conflict that prevents filtering techniques from scaling without reducing subscription expressiveness or violating encapsulation.

This paper proposes a way to pragmatically combine the benefits of 1) a highly-scalable filtering technique and 2) an expressive subscription language with 3) a generic yet safe event representation. Event safety is achieved by viewing events as objects which are instances of application-defined types, and subscription expressiveness is obtained by supporting subscriptions based on any public members of these types. Scalability is achieved using a multiple stage filtering approach, where events are pre-filtered using elaborate information retrieval techniques. While the use of these techniques generally has the undesirable consequence of breaking event encapsulation, we circumvent this problem by performing *approximate* filtering on the intermediate stages and preserving subscription expressiveness and type safety on an end-to-end basis.

In short, the contribution of this paper is twofold. First, we explicitly pose the inherent tradeoffs in devising event systems. Second, we propose a pragmatic way to handle these issues.

The rest of the paper is organized as follows. Section 2 describes the tradeoffs involving event safety, expressiveness and scalability. Section 3 introduces the idea underlying our multi-stage filtering approach and Section 4 presents an architecture for putting our approach to work. Section 5 presents the concluding remarks.

2 Tradeoffs

In this section, we first discuss in more detail the three desirable properties of events systems, namely *scalability*, *expressiveness*, and *event safety*, before pointing out inherent tradeoffs between these properties. We start by the latter property, in order to underline the impact of this often neglected constraint on the two former, better known, aspects.

2.1 The Desirable Properties

Event Safety. Most event systems make few or no assumption on the nature of the events or their format because of the loose coupling between the publishers and subscribers. As a consequence, these systems usually provide low-level message-passing abstractions. As discussed in [EGD01], working with typed events (and objects in particular) in event systems offers a number of advantages over unstructured and untyped events. When events are objects (encompassing implicit type inclusion information, a "value", and some behaviour), type hierarchies additionally permit the filtering of these events according to their polymorphic nature. In other terms, events can be filtered according to their conformance to types, including "content-based" queries expressed on any public members of these event types. Subscribers can register their interest to some event type (including all its subtypes), and if encapsulation and type safety are guaranteed, publishers can easily extend the hierarchy and create new event (sub)types without requiring subscribers to update their subscriptions.

Filtering Scalability. Most content-based event systems use an indirect form of addressing where selectivity is controlled by subscribers. One can imagine different alternative architectures for content-based event systems.

The first and naive architecture consists in broadcasting events to all subscribers and letting the subscriber filter out events that do not match the local subscriptions at runtime. This approach does not scale well as the number of messages increases.

The second one relies on a centralized server (e.g., Elvin3 [SAB+00]) for filtering events and forward those of interest to the appropriate subscribers. The major drawback of the "centralized" approach is that the server is a bottleneck both in terms of processing power and network bandwidth, in addition to being a single point of failure.

The last and most scalable approach relies on a set of networked nodes (e.g., [CRW00], also known as *overlay network*) for content distribution. Publishers and subscribers are connected to a local node (or a server) that is responsible for forwarding events in the system and delivering it to the local subscribers.

Overlay networks are a key for scalability in content-based event systems because the resource-consuming tasks can be split among all the nodes of the network. Each node is responsible for only a subset of all subscribers and filtering can be performed in a distributed manner. The architecture of the network nodes (e.g., hierarchical, peer-to-peer) and the techniques employed to filter and route events are also key factors in scaling to a large number of subscribers, subscriptions, and event types and instances.

Subscription Expressiveness. The expressiveness of subscriptions defines how accurately subscriptions can represent the interests of the subscribers. With different kinds of subscription languages, it is possible to achieve different "levels" of expressiveness.

The simplest form of subscription languages only permits string matching. Most subscription languages use filters [SAB+00, CRW00, MÖ1] in the form of a set of attributes and constraints on the values of those attributes, where constraints are specified using common equality and ordering relations ($=$, $\neq$, $<$, $>$, etc.), as well as regular expressions.

A further step in subscription expressiveness is to allow events to be filtered according to their type [EGD01]. Type-based filtering adds a new dimension to content-based event systems, by letting subscribers register their interests both to the nature and the value of published events.

Advanced subscription languages are highly desirable, because subscribers can more accurately express their interests.

2.2 The Conflicts

Event safety is a property of event representation, scalability depends on the system architecture, and expressiveness relates to the subscription language. These three aspects , depicted in Figure 1, are not orthogonal: desirable characteristics of one aspect may have unwanted effect on the other aspects. These conflicts are highlighted in the rest of this section.

Event Safety vs. Filtering Scalability. The use of typed information adds some overhead to event filtering. This overhead is generally small when events are not objects, since it only includes the cost of type verification and polymorphic data handling. Specially when events are objects (with their own behavior) and data is in principle accessible only through the object access methods, each object might have to be de-marshalled and instantiated in the runtime execution environment before filtering. When such a scheme is naively applied at each filtering step, scalability and performance decrease strongly.

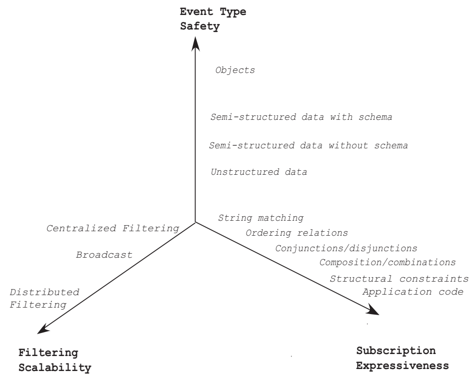

Figure 1. Three Aspects of Event Systems.

In general, as data representation becomes more powerful and type-safe (becoming even part of the application design), the subscription language should also become more expressive, which however might jeopardize scalability.

Expressiveness vs. Filtering Scalability. Highly-expressive subscription languages allow subscribers to accurately specify their interest. In some respect, this can increase scalability by limiting the number of irrelevant events delivered to subscribers. On the other hand, expressiveness increases filtering complexity and processing time. When dealing with real-time events, large numbers of subscriptions and high emission frequencies, filtering time must be kept as small as possible.

Event Safety vs. Expressiveness. Giving the application the possibility of defining own event types makes it difficult to ensure type safety and encapsulation of these events when describing subscriptions. Indeed, to ensure encapsulation and for expressiveness, a subscription should be able to involve methods defined by the type subscribed to, which is difficult to achieve ”reasonably” in a programming language in a way ensuring static type safety. Given the fact that, for a reasonable performance in filtering and routing of events, the publish/subscribe engine has to be given an insight into subscriptions (e.g., for collapsing subscriptions), describing subscriptions by implementing typed filter objects is clearly unappealing. Language extensions [HMN$^+$00, Eug01], or at least powerful language mechanisms, such as advanced reflection and genericity mechanisms are required [Eug01] to achieve satisfactory event safety and expressiveness..

3 Multi-Stage Filtering

In this section, we present a pragmatic approach for the provision of scalability with a type-safe data representation and an expressive subscription language. Our approach is based on multi-stage filtering. We first define some notions related to filtering before describing our approach.

3.1 Definitions

Publishers and subscribers communicate by exchanging events. A subscriber subscribes to specific events by registering a filter that is applied to incoming events: Subscriber only receives the events that matches its filter(s).

Definition 1 (Filter). *Consider a language $\mathcal{L}_D$ for representing events, and a language $\mathcal{L}_F$ for specifying filters. A filter is a function $f \in \mathcal{L}_F : \mathcal{L}_D \rightarrow \{true, false\}$ such that $f(d) = true$ if and only if data d matches the filter f.*

A filter corresponds to a subscription of a subscriber. An event is forwarded to a subscriber when at least one of its subscriptions returns $true$, and discarded otherwise. The filter f_T defined by “$\forall d \in \mathcal{L}_D \;\; f_T(d) = true$” expresses interest in all data events, while the filter f_F defined by “$\forall d \in \mathcal{L}_D \;\; f_F(d) = false$” discards all events.

Example 1. Consider the following events describing stock quotes (events are represented by name-value tuples):

$$
\begin{aligned}
d_1 &= \text{(symbol,“Foo”) (price, 10.0) (volume, 32300)} \\
d_2 &= \text{(symbol, “Bar”) (price, 15.0) (volume, 25600)}
\end{aligned}
$$

A filter selecting only the stock quotes for symbol “Foo” with price higher than \$5 can be defined as follows (filters are represented by name-value-operator tuples):

$$f = \text{(symbol, “Foo”, =) (price, 5.0, >)}$$

Applying filter f to events d_1 and d_2 yields f(d_1) = true and f(d_2) = false. □

We now introduce a covering relation for filters.

Definition 2 (Filter Covering). *A filter f covers another filter f' ($f \sqsupseteq f'$) if and only if the following property holds:*

$$f \sqsupseteq f' \quad \Leftrightarrow \quad \forall d \in \mathcal{L}_D \;\; f'(d) = true \Rightarrow f(d) = true$$

Informally, this means that f' is a more restrictive (or *stronger*) filter than f. The f_T filter covers all filters and the f_F filter is covered by all filters.

Example 2. The following filters cover filter f of Example 1:

$$
\begin{aligned}
f' &= \text{(symbol, “Foo”, =)} \\
f'' &= \text{(symbol, “Foo”, =) (price, 4.5, >=)}
\end{aligned}
$$

□

We also define a covering relation on data.

Definition 3 (Data Covering). *A data event d covers another data event d' for filter f, (i.e. $d \overset{f}{\sqsupseteq} d'$) if and only if the following property holds:*

$$d \overset{f}{\sqsupseteq} d' \quad \Leftrightarrow \quad f(d') = true \Rightarrow f(d) = true$$

Informally, this means that d can be filtered more accurately than (or as well as) d' by filter f, and d is therefore a more accurate representation of the data event.

Example 3. *The following data event covers data event d_1 of Example 1 for filter f:*
d'_1 = (symbol, "Foo") (price, 10.0)

Note that the data covering relation is bound to a filter.

3.2 Scalability through Pre-filtering

As described in Section 2.1, filtering data as early as possible on the path between the publisher and the subscribers in an overlay network can help to achieve scalability. If the data traverses multiple nodes, it may be filtered multiple times, but the amount of data filtered by each node — in particular nodes close to subscribers — will be smaller. This is especially important when matching is time-expensive, e.g., when using objects for events and filters. Without pre-filtering, each node or subscriber would need to filter the sum of all events published by all publishers. The main goal of pre-filtering is thus to scale to a large number of publishers (i.e., data events), while efficient indexing and matching techniques aim at scaling to a large number of subscribers (i.e., subscriptions).

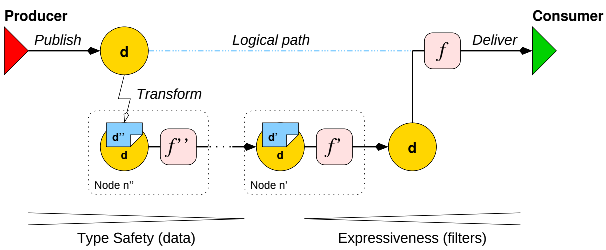

Figure 2. Multi-stage filtering increases scalability by weakening the data representation and the subscription language, and by filtering data on intermediary nodes.

The intuition behind our multi-stage filtering approach is illustrated in Figure 2. Subscription filters are transformed into covering subscriptions that are simpler to evaluate and can be easily indexed for efficient matching. Produced data is transformed into covering data adequate for matching against the weakened subscriber filters; with event objects, transformation typically leads to augmenting the event with some meta-data that describes the relevant attributes of the object's state.

3.3 Transformations

We now describe how the transformations are performed to guarantee that filtering will be consistent with the original data and subscriptions.

Filter Transformation. A filter f can be transformed into a *weaker* filter f' for the purpose of pre-filtering. Using f' at strategic locations in the infrastructure can reduce the network traffic (in comparison with having no filters) and the amount of data to be filtered by the subscribers.

Proposition 1 (Filter Transformation). *Given an original filter f, a filter f' can be used for data pre-filtering if and only if $f' \sqsupseteq f$.*

Proof (sketch) : Since $f' \sqsupseteq f$ and using the definition of filter covering, we have $\forall d \;\; f(d) = f'(d)$. if f' does not cover f then (from the covering definition), there exists some d' such that f(d') = true and $f'(d')$ = false; therefore d' is valid for f but is discarded by f'.

It follows, if $f' \sqsupseteq f$, then f' can be applied before f to the data without loss of consistency.

Data Transformation. Since the data covering relation depends on a filter, data and filter must be weakened in a coordinated manner. For that purpose, one should use transformation functions that generate covering filters in such a way that weakened data events cover original events for all covering filters.

Proposition 2 (Data Transformation). *Given two subscription languages $\mathcal{L}_F$ and $\mathcal{L}_{F'}$, a transformation function $t : \mathcal{L}_F \rightarrow \mathcal{L}_{F'}$, and an original data event d, an event d' can be used for pre-filtering if $\forall f \in \mathcal{L}_F \;\; d' \overset{t(f)}{\sqsupseteq} d$.*

A concrete example of a typed event, filter and data transformation is shown in the [EFGH02].

4 The Architecture

For the implementation of multi-staged filtering, we arrange a set of intermediate nodes in a hierarchy. Each node in the hierarchy has one or more child nodes (or subscribers). Filters associated with child nodes (or subscribers) are weakened and stored in the parent node with the corresponding child node (or subscriber) identity (ID). Hence a node has a set of filters and an associated set of child nodes (or subscribers). When a node receives an event, it is checked against each filter and if the event *passes* through the filter, it is forwarded to the child node/s (or subscriber/s) associated with the filter.

4.1 An Example

An example of this arrangement is shown in Figure 3. In this example there are four stages[1]. We consider the user-level stage as the lowest stage; subscribers are at this stage. Other three stages consist of intermediate nodes. Events are filtered and forwarded to the subscribers by these nodes. Published events are first forwarded to the top most stage. Then the events are forwarded from higher stages to lower stages. Filters are kept at each node. Weaker filters are applied at higher-stages; the strongest filters are at the lowest stage.

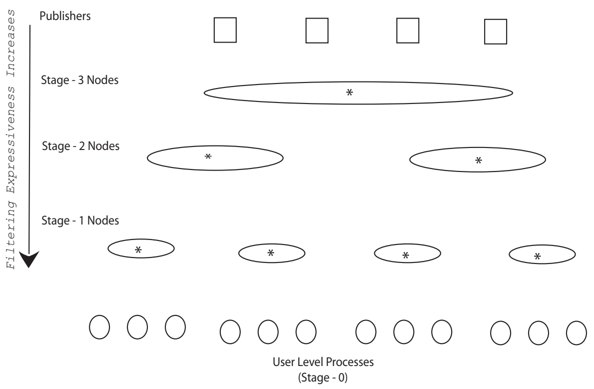

Figure 3. Multi-stage filtering using hierarchical nodes.

In our architecture the filters are arranged in the hierarchy as follows.

Stage-0. At this stage the received events are evaluated against the exact filters specified by the subscribers. Hence the perfect filtering is achieved at this level. For example, the following filters could be a set of filters specified by subscribers.

f_1 = (class, "Stock", =) (symbol, "Foo", =) (price, 10.0, $<$) (volume, 3000, $>$)
f_2 = (class, "Stock", =) (symbol, "Foo", =) (price, 11.0, $<$) (volume, 8000, $>$)
f_3 = (class, "Stock", =) (symbol, "Bar", =) (price, 8.0, $<$) (volume, 10000, $>$)
f_4 = (class, "Auction", =) (Product, "Vehicle", =) (Kind, "Car", =) (Capacity, 2000, $<$) (price, 10K, $<$)

Stage-1. Filters at this stage are constructed by weakening the subscriber filters. The weakening is done such that the weakened filters cover one or more user-level filters. As a result there will be lesser number of filters at this stage than the user level stage. The weakening of filters is done by transforming the least general[2] set of attributes. The most general set of attributes are kept unchanged when weakening the filters at this stage. The user-level filters (f_1, f_2, f_3, f_4,) are weakened to obtain the following filters (g_1, g_2, g_3) which will be used at first stage.

g_1 = (class, "Stock", =) (symbol, "Foo", =) (price, 11.0, $<$)
g_2 = (class, "Stock", =) (symbol, "Bar", =) (price, 8.0, $<$)
g_3 = (class, "Auction", =) (Product, "Vehicle", =) (Kind, "Car", =) (Capacity, 2K, $<$)

Stage-2. Filters at this stage are constructed by weakening the filters at first stage. The next set of least general attributes are weakened as appropriate to form the filter at the second stage. The filters h_1, h_2 and h_3 are constructed by transforming g_1, g_2 and g_3.

h_1 = (class, "Stock", =) (symbol, "Foo", =)
h_2 = (class, "Stock", =) (symbol, "Bar", =)
h_3 = (class, "Auction", =) (Product, "Vehicle", =) (Kind, "Car", =)

Stage-3. At this stage filtering is done only on the type of events. That is, the filters are constructed by weakening second stage filters such that the newly formed filters contain only type information. Finally h_1, h_2 and h_3 are transformed to obtain i_1 and i_2.

i_1 = (class, "Stock", =)
i_2 = (class, "Auction", =)

The above shown weakening process can be automated such that the system create all the weakened filters once the subscription filters are available for an advertised set of events. A simple automation scheme is described next.

4.2 Automating the Filter Weakening

In this automation process, at different stages a subset of attributes are removed from filters to form the weakened filters. For removing attributes at each stage, we first divide (categorize) attributes in to groups. Each group is associated with a stage. A group associated with a stage consists of one or more attribute which can be removed at that stage to form the weakened filter. The publisher at the

[1] The number of stages is a parameter the system designer can decide.

[2] In a filter such as f_1 = (class, "Stock", =) (symbol, "Foo", =) (price, 10.0, $<$) we choose "class" as the most general attribute and "price" as the least general attribute. The process of classifying attributes from most general to least general is described later in this section under "Grouping the attributes".

phase of advertisement specify which attribute is in which group. The information about the groups and their contents (attributes) is disseminated by the publisher with advertisements of events. As a result, any node which receives this information knows which attribute/s should be removed at particular stage and which attribute/s should be used in the weakened filters. A node knows their respective stage. As a result, when a node receives a filter to be weakened, the node can use the group information from the publisher and create the weakened filter in an automated fashion. Once the advertisement and subscriber filters are available, all the necessary filters for intermediate nodes can be generated automatically.

Grouping the attributes. Among all attributes, the attribute which divides the event space into large (with many events) but few sub categories is called the most general attribute. The least general attribute would be the attribute which divides the event space into many sub categories with small numbers of events in each. After identifying the generality of the attributes, they are arranged from the most general attribute to least general attribute. Once this has been done, groups can be created accordingly. In the filtering process, the most general attributes are used at higher stage nodes, and at these stages the least general attributes are ignored.

5 Conclusion

To achieve a type safe, expressive (these two properties add overhead to filtering) and scalable system, computational power requirement at each node should be as low as possible for the proposed filtering scheme. We performed simulations to evaluate the performance of our multi-stage filtering scheme. Due to space constraint we do not present the details and results in this paper but the details of the simulations and results are shown in [EFGH02]. The results are promising. In particular the power requirement for filtering at each node is considerably less than the centralized approach. Hence the event system scales better in terms of event rate. As a result events can be used more safely with more expressive filters without much performance degradation.

References

[AEM99] M. Altherr, M. Erzberger, and S. Maffeis. iBus - a software bus middleware for the Java platform. In *International Workshop on Reliable Middleware Systems of the 13th IEEE Symposium On Reliable Distributed Systems (SRDS '99)*, pages 43–53, October 1999.

[CNF98] G. Cugola, E. Di Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *Proceedings of the 10th International Conference on Software Engineering (ICSE '98)*, pages 261–270, april 1998.

[Cor99] Talarian Corporation. *Everything You need to know about Middleware: Mission-Critical Interprocess Communication (White Paper).* http://www.talarian.com/, 1999.

[CRW00] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proceedings of the ACM Symposium on Principles of Distributed Computing 2000*, pages 219–227, July 2000.

[EFGH02] P. Th. Eugster, P. Felber, R. Guerraoui, and S. B. Handurukande. Event systems : How to have ones cake and eat it too. Technical report, Swiss Federal Institute of Technology, Lausanne, 2002.

[EGD01] P.Th. Eugster, R. Guerraoui, and Ch.Heide Damm. On objects and events. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA) 2001*, October 2001.

[Eug01] P.Th. Eugster. *Type-Based Publish/Subscribe*. PhD thesis, Swiss Federal Institute of Technology, Lausanne, December 2001.

[HMN+00] M. Haahr, R. Meier, P. Nixon, V. Cahill, and E. Jul. Filtering and Scalability in the ECO Distributed Event Model. In *Proceedings of the 5th IEEE International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 2000)*, pages 83–92, June 2000.

[MÖ1] G. Müehl. Generic constraints for content-based publish/subscribe systems. In *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS)*, 2001.

[SAB+00] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with elvin4. In *Proceedings of the Australian UNIX and Open Systems User Group Conference ((AUUG2K)*, June 2000.

[SBCea98] R. Strom, G. Banavar, T. Chandra, and M. Kaplan et al. Gryphon: An information flow based approach to message brokering. In *International Symposium on Software Reliability Engineering (ISSRE '98)*, November 1998.

[SCG01] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-based content routing using xml. In *18th ACM Symposium on Operating System Principles, Banff, Canada*, October 2001.

[Ske98] D. Skeen. *Vitria's Publish-Subscribe Architecture: Publish-Subscribe Overview*. http://www.vitria.com, 1998.

[TIB99] TIBCO. *TIB/Rendezvous White Paper*. http://www.rv.tibco.com/, 1999.