# Attribute-based filtering for embedded systems

Carlos Mitidieri *
Carlos.Mitidieri@informatik.uni-ulm.de

Jörg Kaiser †
Kaiser@informatik.uni-ulm.de

Dept. of Computer Structures
University of Ulm
James-Franck-Ring
89069 Ulm, Germany

## ABSTRACT

Filters are essential components of a publisher/subscriber communication systems. They provide the necessary selectivity enforcing that a subscriber only is notified about the events for which it actually has subscribed. The paper deals with the problem of establishing a filtering mechanism suitable for distributed systems in which the nodes have memory and performance constraints and the interconnection network has a limited bandwidth, e.g. as in systems composed from smart sensors and actuators. Thus the trade-off between expressiveness on the one side and efficiency and predictability on the other side has to be balanced adequately for coping with the resource constraints. The paper proposes attribute-based filtering which allows to filter on the structural properties of events. The notion of super conformance establishes a relation between these structural properties. Attribute-based filtering constitutes a variant of type-based filtering but reflects a more component-oriented view which is beneficial when dealing with smart sensor or actuators components.

## Keywords

publish/subscribe, embedded, sensors, real-time

## 1. INTRODUCTION

The publisher/subscriber communication model enjoys increased popularity in the area of control systems [14, 18, 10]. This is firstly because it supports many to many communication which is well suited for the dissemination of sensor data and commands. Secondly, it does not implicitly create control dependencies through communication which is a desirable feature to maintain control autonomy of the individual smart components [19]. Thirdly, it enables spontaneous communication which well reflects the needs of a sensor system to react on external stimuli. Finally, it supports dynamic configuration and interaction because its routing scheme is based on the content of a message rather than on a destination address. This property is particularly useful because it allows to communicate without having to know names or references. Actually, a component has only to specify what kind of information it provides and what information it needs, leaving open which other component will provide this information. This encourages the dynamic deployment of components which can be developed independently and the spontaneous interaction without prior knowledge of communication relations. This is a substantial advantage in systems composed from autonomous networked sensors and actuators, each of which may comprise mechanical, hardware and software parts and may be independently designed and produced by different vendors.

Because communication relations are not fixed at design time, an overhead of resolving this issue occurs at a later time, e.g. at deployment, configuration or run-time. When mapping the abstractions of the publisher/subscriber model to an underlying communication medium, two problems have to be tackled, routing and filtering. Routing addresses the mechanism of disseminating the event only characterized by its content to the interested subscribers. Filtering is the other side of the coin which has to assure that only those events reach a subscriber, for which this subscriber has indicated an interest. As a matter of fact, routing and filtering are often combined [3, 14]. As indicated above, we are working on a publisher/subscriber middleware which is suited to accommodate resource constrained components. Therefore, approaches which rely on a purely content-based approach of routing and filtering [16, 3, 11] are not feasible. In these systems, arbitrary predicates over the event content have to be evaluated by the subscribers or intermediate event dispatchers, which is not feasible in tiny systems because of the computational overhead. An exception may be some sensor networks [4] which use content-based source discovery and routing. However, these networks have either no or very weak timing constraints. In contrast to these systems, we are striving for control systems in which smart sensors and actuators cooperate under stricter timing conditions [13]. To cope with these requirements, we propose a filter scheme that is based on subject filtering and attribute filtering which can be implemented with less overhead compared to content filtering. At the first stage of filtering, subject filters provide a coarse grain selection and can be

*
†

implemented with very little computational costs. Actually, we use a binding mechanism to dynamically map a subject to a network address and hence, combining routing and filtering by exploiting the hardware of the communication controllers. A detailed description of the mechanism can be found in [12]. However, it turned out that subject filters alone do not provide sufficient selectivity. Too many events pass this filter putting the burden to decide whether an event is useful or not to the subscriber's application software.

Consider a mobile robot in which the reactive system layer is composed from smart sensors and actuators. A smart sensor may publish a distance event which is tagged by the respective subject. A smart actuator, e.g. the motor controller, may subscribe to the distance event to reactively stop the motor when the distance is below a certain threshold, indicating that an obstacle is ahead. In a team of cooperating robots, the same distance information may also be used by other robots to control their behavior. In this case, they need additional information associated with the event, e.g. the position and orientation of the respective robot. Some software component inside the robot may add this information. Of course, we could assign a new subject to the event which now has additional attributes describing the context in which it was created. It is clear that this would sacrifice the relation between the content and the subject of an event which still is a distance. On the other hand, if the events are tagged with the same subjects, then some mechanism must be provided to distinguish them. Structural properties of the events can be exploited for this purpose, resulting in simple filters which detect the existence or absence of attributes and can cope with this problem.

Attribute-based filtering can be seen as a variant of type-based filtering [25, 5, 20, 11]. A certain type of event is related to a subject in the publisher/subscriber model. Type-based filtering imposes a structure on the properties of an event by establishing a hierarchy of subtypes. The type hierarchy reflects a refinement from the more general to the more specific properties of an event. In a system built from smart components it is difficult to construct a type hierarchy top-down. The hardware-software building blocks define the specific properties and attributes which make up the system. These building blocks are developed independently and may come from multiple vendors. Although delivering the same type of information, they may be equipped with largely varying properties and attributes. To define a bottom-up structure on the attributes of such diverse components needs a model which is different from the usual declarative type hierarchy in a object-oriented system.

In this paper, we will describe attribute filters which work on the structural properties of events. We will introduce the notion of conformance to express the relations between events of the same type even if the attributes can not be ordered in a strict top-down hierarchy. Conformance is exploited to define filters which can be implemented with a low computational overhead. The remainder of this paper is organized as follows. The rationale behind our definition for attribute-based filters is described in Section 2. In Section 3 the overall filtering scheme is presented in perspective with our P/S model. Implementation issues are briefly discussed in Section 4. The applicability of the proposed scheme is illustrated in Section 5 through the analysis of a typical embedded application. The related work is commented in Section 6. Finally, conclusions are provided in Section 7.

## 2. ATTRIBUTE-BASED FILTERING

For a publish/subscribe system, it is important to precisely filter the stream of events such that only those events are received for which a subscription has been performed. The degree to which such constraints can be defined is known as the expressiveness of a filtering model.

Regarding expressiveness, two main filtering paradigms are identified in the literature: subject-based (e.g. [17]) and content-based (e.g. [3]) filtering. In some systems, the notion of subjects has been mapped to that of abstract types in the context of object-oriented programming languages [5, 25]. A subject is a label acting as an event type identifier, placed in a single field that is orthogonal to the content. Content is application related data and may comprise several fields. Subject filtering can be implemented through a table lookup while content filtering requires the evaluation of arbitrary predicates; which is obviously expressive, but costly. Even considering the existence of algorithms affording for sub-linear time complexity and linear space (i.e., memory) complexity [1], the computational overhead and memory requirement of content-based filters are still excessive for embedded systems. Moreover, when striving for real-time properties, the variance between the best- and the worst-case evaluation times of such algorithms results in an inefficient utilization of already scarce computing resources. The reason is that reservations must be done for the worst-case.

Because of a better predictability and a lower overhead, subject-based filtering is the preferred mechanism for embedded control systems. Nevertheless, as already mentioned above, subject filters lack expressiveness. Hence, attribute-based filters are introduced which allow to exploit a more fine grained event filtering. As explained in detail below, attribute filters block or pass events based just on the presence or absence of attributes. Because presence/absence of attributes can be mapped to a simple bit vector, this filter can be evaluated through simple table lookup and hence can be implemented with low overhead and also a low temporal variance, as desired in a real-time setting.

To specify attribute filters, we need some formalism to describe the relationship of attributes which define a substructure of some event type. As mentioned above, we have to cope with the situation that the system is composed from building blocks which provide a certain type of information which may be qualified by a variety of attributes. Therefore we can not order the attributes in a usual type hierarchy, as the relations may have to be defined after the implementation of the components. Structural conformance seems to be an appropriate foundation to define the meaning and assess the properties of attribute-based filters. As a formalism, structural conformance was developed in the compiler research [2] and has been implemented in some programming languages, e.g. [21]. Intuitively, a type "P" conforms to a type "Q" if an instance of "P" can be used where an instance of "Q" is expected. It results that "P" must present at least the same attributes (or accessing methods, in an

object-oriented environment) as "Q". E.g., considering that each letter represents an attribute in Figure 1: <A,B,C> conforms to all, while <A,C> conforms to <A,C>, <A>, <C> and < >.

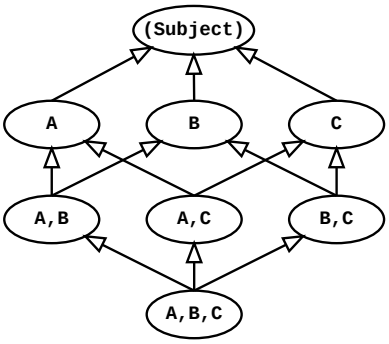


**Figure 1: Hierarchy defined by structural conformance.**

Taking structural conformance as the norm for building event hierarchies, it follows that an event may belong to several super-types. These super-types may not even be related. Moreover, super-types may be defined after sub-types. This property firstly supports the requirement which originates from building the system from components. Secondly, it enables dynamic evolution under continuous operation. These requirements were formulated in [17] and describes the ability to add new functions to a system without touching the existing working components.

Given the properties of structural conformance, an attribute filter is defined as one that matches the events conforming to a specific signature. Such signature can be specified as follows:

$$\mathcal{F}_{attr} = \{Name_i : Type_i, ..., Name_j : Type_j\}$$

The $Name_k : Type_k$ elements are the formal definitions of the attributes, where $Name_k$ is an identifier and $Type_k$ is a primitive type, e.g., integer, float, etc. Considering that each letter A, B and C in figure 1 represents an attribute, it follows that a filter $\mathcal{F}_{attr} = \{\}$ matches all events, while a filter $\mathcal{F}_{attr} = \{B\}$ matches the events <B>, <A,B>, <B,C> and <A,B,C>.

## 3. OVERALL FILTERING SCHEME

Our P/S model [14, 13] includes the abstractions depicted in Figure 2. The event channel handler (ECH) is a distributed middleware component offering an event channel abstraction to applications. There is an instance of the ECH in each node. Our approach contrast to others [10, 9] in the extent that we exploit event channels to abstract the non-functional properties of the underlying network [13]. To each event channel corresponds one subject. The subject is a long bit vector acting as an identifier which is globally unique in the system. As already mentioned, a binding mechanism is exploited to dynamically map subjects to network addresses, trading flexibility for efficiency. The dynamic binding is transparent to applications and is provided by the event channel broker (ECB). Given that the underlying network is a broadcast medium in our system (Section 4.2), the routing problem is tackled by broadcasting events. Filtering is then performed on the subscribers' side, under the control of the ECH component. Gateways have two roles in our system: they are exploited to define an hierarchy of different networks and to encapsulate zones where a certain level of quality of service (QoS) is assured. The network hierarchy may reflect containment relations of components. As an example consider a mobile robot, which may be viewed as a component in a team of robots, which is itself composed by other networked components like inertial and distance sensors, cameras, controllers, etc. These issues are discussed in more details in [27, 26, 13].

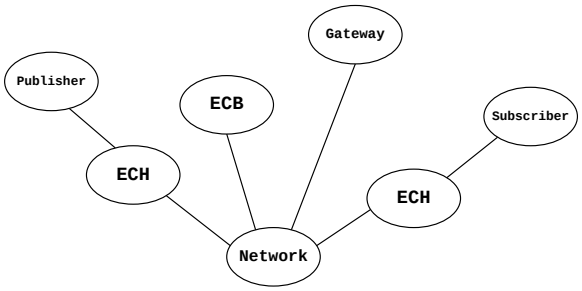


**Figure 2: Architectural components of the P/S middleware.**

An event is an instance of an event type, which is characterized by a subject, attributes and contents. Attributes qualify the event, and may relate to non-functional properties of event dissemination or to the context in which an event is generated, like location, time, etc. Attribute filters can be applied to attributes and to structured content parameters. The filtering scheme requires the specification of a subject filter and an attribute filter for each subscription. That means, a match depends on the conjunctive evaluation of the subject- and the attribute-based filter. Therefore, the complete specification of a filter is as follows:

$$\mathcal{F} = \{Subject, \{Name_i : Type_i, ..., Name_j : Type_j\}\}$$

Subject and attribute filters are managed and/or executed in the ECH component. Filtering events on the subscribers' side may simplify the management of the predictability in an open distributed system because the schedulability analysis of each node must consider only the subscriptions that are issued locally. Thus, regarding this aspect, there is no need for run-time mechanisms akin to an admission control of new subscriptions. When needed, content filtering can be performed in the application layer, as the definition of predicates is supported by virtually any programming language. It is expected that the need for content filtering will be reduced after the application of the other filters.

Event structures can be mapped to bit vectors, each bit representing the absence/presence of a given attribute. If such bit vectors are to be used for filtering purposes, every component in the system must agree on those mappings. In order to enforce such agreement, we introduce the notion of a *Super-Conformant Vector* (SCV). The SCV includes

the set of attributes from which a designer may pick those that will be used to "configure" an event source. Hence, the super conformant vector is the starting point for defining the hierarchical relations among events, from bottom to up. The ECH will manage to accept subscriptions and announcements only for events that are conformed by the SCV specified for a given event channel. The restrictions imposed by such scheme are alleviated by making the description of the SCV available online. The SCV consulting service could be co-located with the ECB or in a dedicated service, to be defined.

Filtering on the basis of structural conformance may lead to undesired matches. This is exactly the case when the conformity relation coincidentally applies to two (or more) event types which are actually unrelated regarding the application semantics. This potential anomaly is avoided in the proposed filtering model because the sub-typing associations are restricted to the scope of a subject, which is enforced to be globally unique in the system. Accordingly, the uniqueness of attribute definitions is enforced through the SCV mechanism.

# 4. IMPLEMENTATION ISSUES

Some issues related to implementation are discussed in the following.

## 4.1 Programming attribute filters

The simplest way for specifying attribute filters in programs is by means of bit vectors. This is a quite lightweight implementation scheme, which is something primitive and error prone. Yet, it is advantageous and preferable for programming mass produced, deeply embedded components, for which a high degree of optimization is required, debugging effort is worthwhile. An improvement on code readability and extensibility consists in specifying attributes through character strings and performing an automatic conversion to the compact form specified by SCV at runtime. A third alternative for specifying attribute filters is to exploit structural reflection [7], which is a built-in feature of some programming languages [24]. Supported by reflection, the middleware can assess the name and the primitive type of formal parameters contained on a structure passed in a subscription as a filter specification. Such information can serve the same purpose as the labels discussed above. That means, a list of formals can be converted to a tag, based on the SCV mapping (section 3) provided online. Reflection poses an overhead cost but offers a safety gain by enabling runtime type checking. This section aimed to provide an overview on the pros and cons of some models that may be employed for programming attribute filters. The selection depends on issues like target platform, efficiency requirements, etc. They can be used together in order to satisfy the requirements of an heterogeneous system.

## 4.2 Exploiting network properties

As described in details in [14], subject-based filtering was implemented via the frame-addressing mechanism of the CAN-Bus protocol (ISO 11898 an 11519-1, [22]). On the CAN-Bus, the frame identifier (i.e. the frame address) is related to the contents of the payload rather than to a receiver address. On receivers side, CAN-Bus controllers can be configured to receive frames based on the content-related frame identifier. The implemented approach for making subject filtering more efficient in embedded systems is to map the subject tags to the identifier of the CAN-Bus frame, rather than placing them in the payload. As a result, events are filtered in the MAC sub-layer through an efficient hardware mechanism, relieving the host for application related computing. The same approach can be employed for implementing attribute filters, i.e., an attribute vector can be mapped to a segment of the frame identifier.

# 5. AN APPLICATION ANALYSIS

In this section, the exploitation of attribute filters is illustrated through the analysis of a realistic application scenario, which is described in the next paragraph. The description is supported by Figures 3 and 4.

## 5.1 An automotive traffic monitoring system

Two of the most common roles found in an intelligent traffic system are related to vehicles velocity monitoring and statistics collection. These roles are modeled through the *VelocityMonitor* (VM) and *StatisticsCollector* (SC) classes, respectively. The *SensorLoop* (SL) class abstracts electro-magnetic sensors, which can detect the presence and categorize metallic masses (i.e., vehicles). Loop sensors are deployed in longitudinally oriented pairs, bonded to the ground of the road lanes. A *VelocityMonitor* object must coordinate with a pair of *SensorLoop* objects, in order to compute the velocity of passing vehicles. On the other hand, a central *StatisticCollector* object must coordinate with just one *SensorLoop* object out of each pair inside the metropolitan area. In a P/S system featuring subject-based filtering, *SensorLoop* objects would be mapped to publishers of events on the subject *Loop*. On the other hand, *VelocityMonitor* and *StatisticsCollector* objects would be mapped to subscribers of the events on the subject *Loop*. Apropos: the velocity of passing vehicles is computed dividing the distance separating the sensors pair by the time interval between vehicle-detection events. Such interval can be determined by timestamping the *Loop* events, assuming that clocks are synchronized inside the CANs.

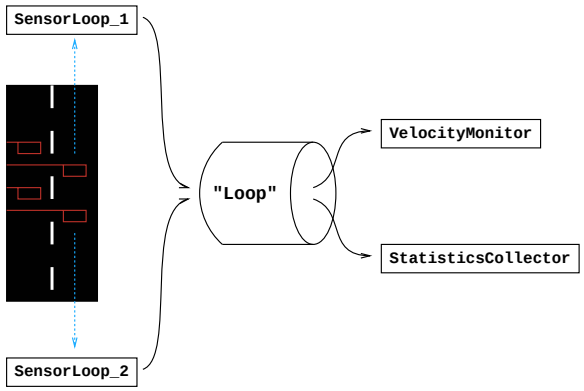


**Figure 3: Mapping of a traffic monitoring application to the P/S model.**

Notice that relating the sensor type to a subject enables a coherent representation of application semantics in the P/S system. It is inferable that this will be the case in

most applications. Further, the central *StatisticCollector* wants to receive presence-events from one sensor out of each control pair. Attribute filters can be employed in order to express and implement such a requirement in a way that further relates the application structure to the coordination environment.

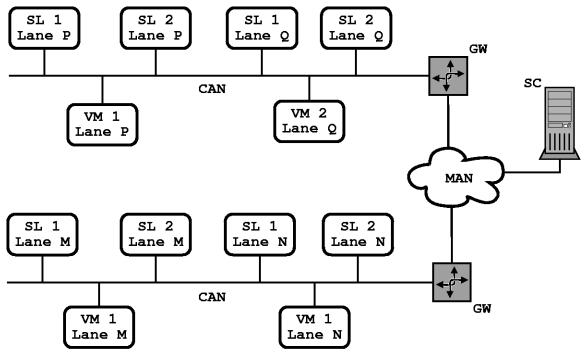


**Figure 4: Network deployment of a distributed traffic monitoring application.**

## 5.2 Applying attribute filters

At line 01 of figure 5 is indicated a plausible definition for the SCV related to the subject *Loop*. In this definition, *LaneIdentifier* is a local (to the CAN) identification of the lane, *TimeStamp* is the time at which a vehicle is detected, *Localization* can be simply an urban address and *VehicleCategory* can assume the values from an enumeration like *{truck, car, motorcycle}*. Other functional and non-functional attributes are omitted for simplicity.

Let's distinguish the objects on each sensor pair as *SensorLoop_1* and *SensorLoop_2*. Both objects publish events on the subject *Loop*, but with different attribute compositions. The formal event definitions are showed in the lines 02 and 03 (figure 5), respectively.

The subscribers are objects of the *VelocityMonitor* and *StatisticsCollector* classes. They are both interested on events pertaining to the subject *Loop*, but on distinct attribute compositions. They express their distinct interests by passing different event formal definitions in their subscriptions. Such formal definitions are showed in the lines 04 and 05 (figure 5).

As assured by attribute filtering, a subscription specified by the *VelocityMonitor* object is notified about the events published by both *SensorLoop_1* and *SensorLoop_2*. This is quite clear, as the formal definition for the *Event_For_VM* (line 04) is conformed by the formal definitions of both *Event_From_SL_1* (line 02) and *Event_From_SL_2* (line 03).

On the other hand, the StatisticsCollector object is notified about the events published just by the *SensorLoop_1* object. The reason is clear again: the formal definition for the *Event_For_SC* is conformed by Event_From_SL_1 (line

```
/ The SCV for the event channel "Loop"
01 SCVDEF{Loop,{li:LaneIdentifier;ts:TimeStamp;
                    lo:Localization;vc:VehicleCategory}};

// SensorLoop_1
02 Event_From_SL_1{li:LaneIdentifier;ts:TimeStamp;
                      lo:Localization;vc:VehicleCategory};

// SensorLoop_2
03 Event_From_SL_2{li:LaneIdentifier;ts:TimeStamp};

// VelocityMonitor
04 Event_For_VM{li:LaneIdentifier;ts:TimeStamp};

// StatisticsCollector
05 Event_For_SC{ts:TimeStamp;lo:Localization;vc:VehicleCategory};
```

**Figure 5: Formal definitions of event compositions following from the example application.**

02), but not by *Event_From_SL_2* (line 03).

The value on the *LaneIdentifier* attribute can be evaluated by the *VelocityMonitor* objects in order to coordinate with specific sensor pairs. Such evaluation of predicates in the application layer is foreseen in the proposed filtering model. And second, in this paper we were not concerned with the event routing problem, taking place in the gateways and beyond, in the metropolitan area network (figure 4). This is a significant problem to be considered in the elaboration of an architecture for real-time filtering.

Now, suppose that the traffic engineers have concluded that the vehicle categorization should be complemented by a *confidence factor* attribute, in order to improve their statistics. A new generation of loop sensors is designed, including such attribute. Then, one sensor out of each pair that is already deployed in the city must be substituted by a sensor of the new version. The substitution work may last several months. In this meantime, the *StatisticsCollector* is continually notified about events published by sensors of the new and the old version, because both conform to the original subscription (line 05 figure 5). When the substitution is finally complete, the *StatisticsCollector* can issue a single new subscription including the confidence factor attribute. The system has evolved without any disruption on the service provided.

# 6. RELATED WORK

The structural aspect of events have been exploited in many systems for elaborating filtering models but, to our knowledge, always tied to the evaluation of a predicate [3, 23, 15, 16, 8]. That means, the specification of a subscription in the cited works included, in a single artefact, the attribute itself, a test operator and a constant to which the attribute value should be compared. This association is omitted in the proposed definition of attribute filters which is based only on structural conformance. By considering attribute filters orthogonally to content filters (i.e., predicates) it was possible to put in perspective some characteristics of the former. In embedded systems, the proposed combination of subject and content filters provides an adequate balance among predictability, efficiency and expressiveness. To our knowledge, such particular combination of subject and attribute filters has not been proposed before.

JavaSpaces [25], CEO [11] and Obvents [6] implement type-based filtering. In these systems, events hierarchies are explicitly defined top-down, based on the support provided by object-oriented languages. In contrast, the proposed notion for attribute filters supports the definition of an event hierarchy from bottom to up, i.e., from the most specific to the most general type. Hence, relations can be defined after the design and implementation of the smart components which are used as building blocks. In JavaSpaces and CEO, filters are specified through templates. This mechanism includes a constrained form of content filtering akin to the structured naming in Linda, which is omitted in attribute filters.

## 7. CONCLUSIONS AND FUTURE WORK

The paper has focused on systems which 1. are composed from smart, performance constrained devices operating over bandwidth constrained communication links, and 2. have to achieve real-time behavior because their tight interaction with their physical environment. Therefore filtering must not incur a high computational overhead and secondly, must allow an easy temporal analysis. Structural filtering based on event attributes meets these requirements. Before applying a complex evaluation of the event contents, subject and attribute filtering enable a fast and low overhead filter stage. Particularly, it allows an early discarding of events which are not relevant. Moreover, structural filtering eases the temporal analysis of filtering and thus contributes to the predictability requirements in a real-time setting.

Attribute filters are defined over the structural properties of event representation. They match events based just on the presence or absence of attributes and does not include the evaluation of predicates. By considering attribute filtering as a paradigm on its own, separated from content filtering (i.e., without predicates), it is possible to assure an appropriate balance of the properties needed for meeting our requirements. The trade-off is the overhead of an additional table look-up.

The presented filtering model is being implemented, after which an experimental evaluation will be performed. Meanwhile, a filtering architecture for nodes and gateways is being developed. The filtering architecture for the gateways is specially important, as the gateways connect real-time subsystems to best-effort subsystems and are responsible for the scoping and routing of events.

## 8. ACKNOWLEDGMENTS

This work has been supported by the European Union's Information Society Technology Program under contract IST-2000.26031 (CORTEX: COoperating Real-time senTient objects: architecture and EXperimental evaluation).

## 9. REFERENCES

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Symposium on Principles of Distributed Computing*, pages 53–61, 1999.

[2] L. Cardelli. Structural sub-typing and the notion of power type. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 70–79, San Diego, California, 1988.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design of a scalable event notification service: Interface and architecture. Technical report, Department of Computer Science, University of Colorado, August 1998.

[4] D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Mobile Computing and Networking*, pages 263–270, 1999.

[5] P. Eugster, R. Guerraoui, and C. Damm. On objects and events. In *Proceedings for OOPSLA 2001*, October 2001.

[6] P. T. Eugster and R. Guerraoui. Content-based publish/subscribe with structural reflection. In *6th Usenix Conference on Object-Oriented Technologies and Systems*, 2001.

[7] J. Ferber. Computational reflection in class based object-oriented languages. In *Conference proceedings on Object-oriented programming systems, languages and applications*, pages 317–326, New Orleans, Louisiana, United States, 1989. ACM Press.

[8] D. Gelernter. Generative communication in Linda. *ACM Trans. Prog. Lang. Syst.*, 7(1):80–112, 1985.

[9] P. Gore, R. Cytron, D. C. Schmidt, and C. O'Ryan. Designing and optimizing a scalable CORBA notification service. In *LCTES/OM*, pages 1196–204, 2001.

[10] T. H. Harrison, D. L. Levine, and D. C. Schmidt. The design and performance of a real-time corba event service. In *OOPSLA'97*, pages 184–200, Atlanta, USA, October 1997.

[11] R. Hayton, J. Bacon, J. Bates, and K. Moody. Using events to build large scale distributed applications. In *ACM SIGOPS European Workshop 96*, September 1996.

[12] J. Kaiser and C. Brudna. A publisher/subscriber architecture supporting interoperability of the CAN-Bus and the internet. In *IEEE International Workshop on Factory Communication Systems (WFCS2002)*, Västeras, Sweden, August 2002.

[13] J. Kaiser, C. Brudna, and C. Mitidieri. A real-time event channel model for the CAN-Bus. In *11th Annual Workshop on Parallel and Distributed Real-Time Systems, in conjunction with the International Parallel and Distributed Processing Symposium IPDPS*, pages 22–26, Nice, France, April 2003.

[14] J. Kaiser and M. Mock. Implementing the real-time publish/subscribe on the controller area network (can). In *2nd International Symposium on Object-Oriented Real-time Distributed Computing* , Saint-Malo, France, May 1999.

[15] R. Meier and V. Cahill. Steam: Event-based middleware for wireless ad hoc networks. In *International Workshop on Distributed Event-Based Systems*, 2002.

[16] G. Mühl, L. Fiege, and A. P. Buchmann. Filter Similarities in Content-Based Publish/Subscribe Systems. In H. Schmeck, T. Ungerer, and L. Wolf, editors, *International Conference on Architecture of Computing Systems (ARCS)*, volume 2299 of *Lecture Notes in Computer Science*, pages 224–238, Karlsruhe, Germany, 2002. Springer-Verlag.

[17] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The information bus - an architecture for extensible distributed systems. In *ACM Symposium on Operating System Pronciples*, pages 58–68, 1993.

[18] G. Pardo-Castellote, S. Schneider, and M. Hamilton. NDDS: The real-time publish-subscrib middleware. In *IEEE Real-time Systems Symposium*, pages 222–232, 1997.

[19] C. E. Pereira, J. Kaiser, C. Mitidieri, C. Villela, and L. B. Becker. On evaluating interaction and communication schemes for automation applications based on real-time distributed objects. In *4th Int. Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'01)*, Magdeburg, Germany, 2001.

[20] P. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In J. Bacon, L. Fiege, R. Guerraoui, A. Jacobsen, and G. Mühl, editors, *Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, July 2002.

[21] R. K. Raj, E. D. Tempero, H. M. Levy, A. P. Black, N. C. Hutchinson, and E. Jul. Emerald: A general-purpose programming language. *Software - Practice and Experience*, 21(1):91–118, 1991.

[22] Robert Bosh GmbH. *CAN Specification version 2.0*, September 1991.

[23] G. Starovic, V. Cahill, and B. Tangney. An event-based object model for distributed programming. In *OOIS (Object-Oriented Information Systems) '95*, pages 72–86, London, 1995. Springer-Verlag.

[24] I. Sun Microsystems. Java Core Reflection: Overview and API Specification. http://java.sun.com/j2se/1.3/docs/guide/reflection/.

[25] Sun Microsystems, Inc. *JavaSpaces Service Specification Version 1.1*, October 2000.

[26] P. Veríssimo and A. Casimiro. Event-driven support of real-time sentient objects. In *Eighth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2003)*, Jan 2003.

[27] P. Veríssimo, V. Cahill, A. Casimiro, K. Cheverst, A. Friday, and J. Kaiser. Cortex: Towards supporting autonomous and cooperating sentient entities. In *European Wireless 2002*, Florence, Italy, February 2002.