STEAM: Event-Based Middleware for Wireless Ad Hoc Networks

René Meier and Vinny Cahill
Department of Computer Science, Trinity College Dublin, Ireland
{rene.meier, vinny.cahill}@cs.tcd.ie

Abstract

With the widespread deployment and use of wireless data communications in the mobile computing domain the need for middleware that interconnects the components that comprise a mobile application in distributed and potentially heterogeneous environments arises. Middleware utilizing an event-based communication model is well suited to address the requirements of the mobile computing domain, as it requires a less tightly coupled communication relationship between application components compared to the traditional client/server communication model. This is particularly useful with the use of wireless technology, where communication relationships amongst application components are established very dynamically during the lifetime of the components. Recent research in the area of event-based middleware for the mobile computing domain focuses on infrastructure network models for wireless data communication. In this paper, we present STEAM, an event-based middleware service that has been specifically designed for wireless local area networks utilizing the ad hoc network model. We argue that an implicit event model is best suited for the envisaged ad hoc environment and present our approach of exploiting a novel combination of three different types of event filter to address the problems related to the dynamic reconfiguration of the network topology as well as their impact on the scalability of a system and the timely delivery of events. [1]

1. Introduction

The widespread deployment and use of wireless data communications in the mobile computing domain is generally recognized as being the next major advance in the information technology industry. Both mobility and wireless networking represent key enabling technologies underlying the vision of *ubiquitous computing* [14], where interconnected computers will be embedded in a wide range of appliances ranging in size from door locks to vehicle controllers performing tasks, such as automatically opening doors and routing vehicles to their intended destinations, on behalf of their human users. The event-based communication model represents an emerging paradigm for middleware that asynchronously [2] interconnects the components that comprise an application in a potentially distributed and heterogeneous environment, and has recently become widely used in application areas such as large-scale internet services and mobile programming environments [1]. Event-based communication is well suited to address the requirements of the mobile computing domain [7]. It avoids centralized control and requires a less tightly coupled communication relationship between application components compared to the traditional *client/server communication model*. This is particularly useful with the use of wireless technology in the mobile computing domain, where communication relationships amongst heterogeneous application components are established very dynamically during the lifetime of the components.

Mobile computing environments can utilize either the *infrastructure* or the *ad hoc network model* for wireless data communication [6]. The infrastructure network model exploits *access points* to establish communication relationships among mobile application components and to coordinate their transmissions. An access point is analogous to a base station in a cellular communications network forming groups of mobile application components that are under its direct control. Access points may be connected to a fixed network, such as a company intranet or the Internet, and act as a portal allowing the components under its control to connect to the fixed network. In contrast, the ad hoc network model allows application components to communicate with each other without the aid of access points or a fixed network. Any application component can establish a direct communication relationship with any other application component without having to channel the transmission through an access point (provided there is no network partition). This allows application components to communicate

[1]The work described in this paper was partly supported by the Irish Higher Education Authority's Programme for Research in Third Level Institutions cycle 0 (1998-2001) and by the FET programme of the Commission of the EU under research contract IST-2000-26031 (CORTEX).

and collaborate in a spontaneous manner in the absence of a conventional fixed network.

Several middleware services utilizing the event-based communication model have been developed thus far by both industry [12] and academia [2], [8]. Most of these assume that the application components comprising an application are stationary and that a fixed network infrastructure is available to facilitate communication. They do not address the problems introduced by mobile application components and wireless network models, related to the dynamic reconfiguration of the network topology. Recently, some research has been done to support mobile computing in event-based middleware [7], [13]. However, this work has focussed on the infrastructure network model for wireless data communication assuming middleware components acting as event dispatchers being available. In this paper, we present STEAM (Scalable Timed Events And Mobility), an event-based middleware service that has been designed for the mobile computing domain. Specifically, it is intended for IEEE 802.11-based wireless local area networks (WLAN) utilizing the ad hoc network model. The next section introduces event-based middleware and provides a preliminary description of the differences between the traditional distributed computing domain that relies on fixed network infrastructure and the mobile computing domain. Section 3 outlines the design restrictions that are imposed on STEAM by the ad hoc network model. In section 4, we present our approach to overcome these restrictions and discuss how STEAM exploits a novel combination of three different types of event filter to address issues that are specific to the mobile computing domain and the ad hoc network model and their impact on the scalability of a system and the timely delivery of events. Finally, section 5 concludes this paper by summarizing our work and outlining the issues that remain open for future work.

2. Event-Based Middleware

An event system is an application that uses event-based middleware to allow the components that comprise the application to interact through *event notifications*. Event notifications, or simply events, contain data that represent a change to the state of the sending application component. They are propagated from the sending application components, called the producers, to the receiving application components, called the consumers. Events typically have a name and may have a set of typed parameters whose specific values describe the specific change to the producer's state. In order to receive events, event consumers have to *subscribe* to the instances of events in which they are interested; they are said to register interest in events. Once consumers have subscribed to events, they receive all subsequent events until they *unsubscribe* (de-register). An event system using event-based middleware, which is also called an *event service*, may consists of a potentially large number of application components, or entities, that produce and consume events. In conventional distributed event systems, entities are located on a number of physical machines that are interconnected by means of a fixed network infrastructure through which event-based communication takes place. In contrast, the STEAM event service is intended for the mobile computing domain where entities reside on mobile computing devices utilizing wireless networks to interact.

3. STEAM Design Issues

When designing event-based middleware or indeed middleware in general, the specific restrictions imposed by the environment for which the middleware is intended needs to be considered. This is particularly important when designing an event service for ad hoc networks, since the environment places especially strict limitations on the middleware due to the lack of any network infrastructure. Traditionally, event services exploit a range of services and protocols including lookup and naming services that are utilized by consuming entities to locate producing entities in order to route event subscriptions to them. Some event services, such as the CORBA notification service [12], use the service of intermediate middleware components for producing and consuming entities to communicate with each other. These services and intermediates must be accessible by the entities in the event system and are often implemented as independently running middleware components located in separate address spaces potentially on remote physical machines. Most significantly, utilizing such middleware components is not practical in an environment that is based on the ad hoc network model for a number of reasons. Middleware components cannot be located on separate physical machines as this approach assumes infrastructure that is inherently omitted in ad hoc environments. It can be argued that such middleware components may be co-located with the mobile entitles sharing the same physical machine. However, this does not overcome the most significant restriction of ad hoc environments, the limitation of the area that can be covered by mobile application components using wireless transmitters. Using a middleware component acting as intermediate requires all entities in a system being able to communicate with it at any given time. This is unlikely in an ad hoc environment as entitles may be distributed over a potentially large geographical area and thus are unlikely to be able to maintain a permanent communication link to the intermediate. In conclusion, the main restriction on the design of STEAM imposed by the ad hoc environment is the omission of application components that provide system-wide services.

4. The STEAM Event Service

In this section, we present our approach to overcome the restrictions imposed on event-based middleware by the mobile computing domain due to mobile application components, the wireless ad hoc network model, and the resulting limitations identified in the previous section. STEAM exploits a novel approach of combining three different types of event filter to address the dynamic aspect of the network topology. Our approach to event filtering supports the predictable behavior of the event service, which is essential for the scalability of a system and the timely delivery of events.

STEAM has been designed for mobile environments, specifically with the traffic management application domain in mind. In this domain, we envisage application scenarios that include a large number of entities representing real world objects using wireless technology and the ad hoc network model. Many of these entities may represent mobile objects including cars and ambulances; other entities may represent object with a fixed location, such as traffic signals and lights. All entities interact using event-based communication in order to exchange information on the current traffic situation. For example, a traffic signal may propagate a change to the speed limit due to road conditions to approaching cars. Another example may involve an ambulance disseminating its location to the cars in its vicinity for them to yield the right of way. Our work is motivated by the hypothesis that in this type of application scenario entities are most likely to interact once they are in close proximity. Reflected on event-based middleware, this means that the closer consumers are located to a producer the more likely they are to be interested in the events propagated by that producer. Significantly, this implies that events are valid within a certain geographical area surrounding a producer. An example scenario demonstrating such behavior would be a traffic light disseminating its status and cars being interested in receiving these events only if they are located within a certain range of the light. Our approach to propagating events within a certain area surrounding producers limits forwarding of event messages, and therefore reduces the usage of communication and computation resources, which are typically scarce in mobile environments. Furthermore, we expect the limitation of event forwarding to reduce the susceptibility of an event system to radio frequency interference. Subsequently, the behavior of communication connections between entities becomes more predictable, allowing the system to support reliable event delivery.

4.1. The Event Model

We have identified three distinct event models that may be implemented by an event service, namely peer to peer, mediator, and implicit [11]. Both peer to peer and mediator-based event models rely on system-wide services for entities to locate intermediate components or indeed each other. As argued in section 3, such approaches are not suitable for environments based on the ad hoc network model. Thus, the STEAM event service implements an implicit event model that allows consuming entities to subscribe to particular event types rather than at another entity or a mediator, without having to rely on system-wide services to locate entities or mediators, or on intermediate middleware components through which entities interact.

4.2. Proximity Group Communication

Group communication [5] has been recognized as a natural means to support event-based communication models [3]. Communication groups provide a one to many communication pattern that can be used by producers to propagate events to a group of subscribed consumers. Although there are other approaches to support message-oriented communication models, including distributed transactions, remote method invocation, and higher-level approaches such as workflow systems, group communication has been identified as the most suitable approach [3]. STEAM exploits a proximity-based group communication service [10] as the underlying means for entities to interact. Proximity groups have been identified as a useful communication paradigm for mobile applications utilizing wireless networks [10], based on the location of the application components. It allows mobile application components to discover each other once they are within the same geographical area using beacons. Significantly, this notion of proximity groups involves both geographical and functional aspects, i.e., the geographical area and the name of the group, to apply for group membership. An application component must firstly be located in the geographical area corresponding to the group and secondly be interested in the group in order to join. In contrast, classical group communication defines groups solely by their functional aspect. STEAM defines both the functional and the geographical aspect that specifies a proximity group. The functional aspect represents the common interest of producers and consumers based on the type of information that is propagated among them. The geographical aspect outlines the scope within which the information is valid, i.e., the limits within which it is propagated. Furthermore, STEAM exploits the message delivery semantics associated with proximity groups to provide end-to-end guarantees when delivering events.

4.3. Event Filtering

An event system may consist of a potentially large number of producers, all of which produce events that may contain different information. Therefore, the number of events propagated in an event-based system may be quite large. However, a particular consumer may only be interested in a subset of the events propagated in the system. *Event filters* provide a means to control the propagation of events. Ideally, filters enable a particular consumer to subscribe to the exact set of events that it is interested in receiving. Before events are propagated, they are matched against the filters and are only delivered to consumers that are interested in them, i.e., for which the matching produced a positive result. STEAM supports three different types of event filter, which are subject, proximity and content filter. The combination of these three types of event filter is specific to the mobile computing domain for which STEAM is designed.

Events propagated by STEAM consist of a name and a set of typed parameters. The name represent the type of an event classifying its structure and the parameters contain the values that describe a particular instance of an event. A common vocabulary is used by producers and consumers in order to agree on the name of a specific event type. A subject filter describes the particular type of the events in which a consumer is interested and hence is matched against event types. It is mapped onto a proximity group and corresponds to the proximity group's functional aspect. A proximity filter corresponds to the geographical aspect of a proximity group describing the geographical area within which events of a specific type are valid. Thus, a proximity filter specifies the scope within which events are disseminated. In principal, either a consuming or a producing entity may specify a proximity filter. However, we believe that in many application scenarios it is the producer that would specify the geographical area within which the generated events are valid. For example, a traffic light propagating events containing the status of its light to approaching cars defines its proximity filter based on the location of the next traffic light and on the local speed limit. STEAM allows producing entities to define their proximity filters, which are deployed when event types are announced. In summary, subject and proximity filters allow a consumer to express interest in a set of events based on their type and the geographical area in which they are propagated. Content filters contain a filter expression that can be matched against the values of the parameters of an event. They are specified using filter expressions that may contain equality, magnitude and range operators as well as ordering relations to describe the constraints of a consumer, similar to the semantics of content filter expressions described in [4]. Subject and proximity filters are applied on the producer side. Events are only propagated to a consumer if both filters match. In contrast, content filters are located at the consumer side and will be evaluated when an instance of an event is received to determine whether or not to deliver it to the application. This approach does not require consumers to pass content filters to producers when subscribing. Most significantly, they need not be forwarded to producers when a consumer changes its location from one geographical area to another. This results in a simple approach to maintaining the subscriptions and content filters in a system that accounts for the dynamic reconfiguration requirement of the system that arises due to the mobility of the entities.

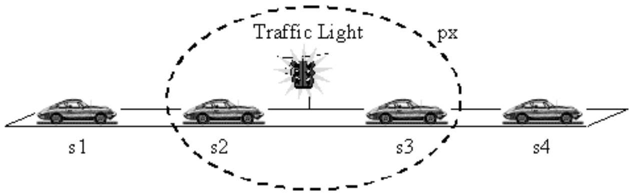
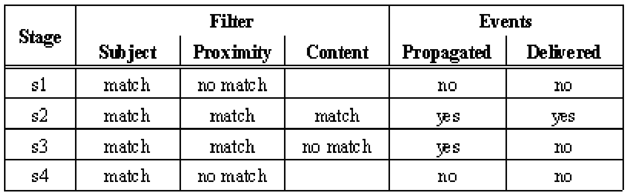


Figure 1. Traffic Light Application Scenario.

The traffic management application domain where (potentially driverless) vehicles are moving through a city interacting with each other and the environment offers a range of example application scenarios demonstrating the exploitation of subject, proximity, and content filters. Figure 1 depicts a traffic light application scenario in which a car is driving along a road acting as a consumer of events, thereby passing through the proximity px of a traffic light acting as event producer. The generated events are of type "traffic light" and their parameters contain the status of the light and its location. The car is shown at stages $s1$ to $s4$ of its journey and is assumed to have an associated subject, proximity, and content filter. The subject filter expresses interest in events of type "traffic light", the proximity filter defines px, and the content filter compares the car's previous and current position with the location of the traffic light stating that "events of the traffic light towards which I am driving" should be delivered. The results of matching events against each of these filters at the different stages of the car's journey and their effect on propagation and delivery of events are summarized in Figure 2. At stage $s1$ of its journey, the car subscribes to the traffic light events deploying the corresponding filters. All subsequent events produced by the traffic light will be matched against the car's filters and will be delivered to it if all three filters produce a positive match. Figure 2 shows that at stage $s1$ (after the car has subscribed) traffic light events are neither propagated nor delivered to the car as it is located outside the traffic light's proximity causing the proximity filter to produce a negative match. Traffic light events are propagated to the car at journey stages $s2$ and $s3$ where both the subject and the proximity filter match as the car has subscribed and is located within the traffic light's proximity. However, they are only delivered to the car's application at $s2$ as this is the only stage where the content filter matches as well. The content filter does not match at stage $s3$ since the car has passed the light's location. No events are propagated at stage $s4$ because the car has left the proximity of the traffic light.

When continuing its journey, the car is likely to enter the proximity of another traffic light, whose events it will receive if the content filter matches without explicitly issuing another subscription.



Stage	Filter			Events	
	Subject	Proximity	Content	Propagated	Delivered
s1	match	no match		no	no
s2	match	match	match	yes	yes
s3	match	match	no match	yes	no
s4	match	no match		no	no

Figure 2. Matching of Events Against Filters.

Scalability has been identified as one of the key distributed computing problems that remains open in message-oriented middleware [3]. Applications exploiting event-based middleware may be large, they may consist of a very large number of entities and subscriptions. The ability of such as system to easily cope with the addition of entities and subscriptions is essential. This is particularly crucial in event systems that are based on the ad hoc network model as their scale typically changes dynamically over time. For example, a traffic management application dealing with the entities representing the cars driving through a city must be able to handle the cars involved regardless of their number, which may range from relatively small during the night to very large during rush hours. STEAM addresses scalability in two ways. It enhances the ability of a system to grow by limiting the propagation of events in space hence reducing the need for events to be forwarded and by utilizing a combination of filters that allow events being matched efficiently. STEAM allows an application to outline the scope in space (the proximity) within which some information (the event type) is valid. As events only need to be forwarded within the limits of their respective proximity, adding an entity affects only the area in which the entity is located. A joining entity causes other entities in the same proximity to reconfigure, i.e., routing and subscription information need to be updated. However, it does not require entities outside the proximity to adjust. Using a combination of subject and content filters within the scope defined by a proximity filter enables a system to combine the advantages of both. Subject filters support the deployment of an optimal matching algorithm. A simple table lookup on the producer side based on the subject results in a constant time algorithm. Content filters support the deployment of filter expressions that can be matched against different parameters of an event. Applying content filters on the consumer side instead of the common approach of deploying them on the producer side results in the distribution of the matching load from a single producer to a number of consumers. Therefore, each consumer has to deal with a small number of content filters, e.g., potentially one per subject, compared to a producer having to match the potentially arbitrary large number of content filters of every subscribed consumer. This approach causes additional overhead due to the propagation of unwanted events to consumers, which are then discarded by the content filter. However, the additional overhead is minimal as the underlying proximity group communication mechanism uses an approach that is based on flooding the area of the proximity with messages at the network layer in order to provide reliable message delivery semantics.

STEAM allows applications to define delivery deadlines and to assign them to specific events. A dispatcher then exploits these deadlines to determine the time to deliver the corresponding events to the subscribed consumers. For the dispatcher to enforce the timely delivery of events the event service is required to behave in a predictable manner; it must support properties that enable predictions on behavior and duration of event propagation and delivery. The predictable behavior of STEAM needs to be enforced by the algorithm that matches events against potential filters, by the mechanism that propagates events on behalf of a producer to the subscribed consumers, and by the scheduler that delivers events to the application. Different scheduling algorithms for delivering events to the application in a deterministic manner have been proposed in [9] and predictable propagation of event messages are addressed in the underlying mechanism for proximity-based group communication [10]. Matching events against a potentially large number of subscriptions, i.e., filters, in a predictable manner is non trivial. Producers exploiting state of the art content-based filters are expected to maintain a very large number of subscriptions and thus need to compare the large number of corresponding filter expressions to the values of an event's parameters. Hence, it will be difficult to implement a matching algorithm that executes in a predictable manner due to the arbitrary number of filters and the arbitrary number of event parameters. In contrast, producers using STEAM's subject-based approach exploit a simple table lookup algorithm to retrieve the list of matching subscriptions, which results in a constant retrieving time. Moreover, the number of the deployed subject filters is expected to be substantially smaller than the number of content filters in a traditional approach as a single filter per subject is required. Our approach of exploiting content filters on the consumer-side faces similar issues as the traditional, producer-side approach. However, we expect our approach to include a substantially smaller number of filters, as we only need to consider the filters that are specific to a single consumer. A sub-linear algorithm suffices to retrieve the filters that correspond to a specific subject using a simple table lookup. Another algorithm is then used to match an event against the set of retrieved filters. We argue that the number of filters at this stage will be very small; there may potentially be a single filter per subject. Computing the match for a single filter is efficient; we propose imposing an upper bound on the computation time for the content filter for each specific subject that can be used to predict the time for matching events of that subject on both the producer and the consumer side.

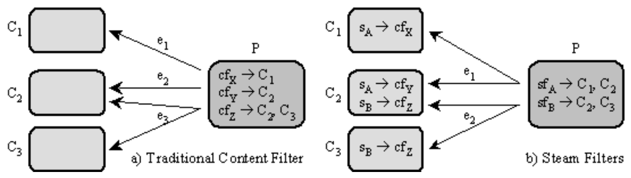


Figure 3. Content vs. STEAM Filters.

Figure 3 compares traditional, content-based filters shown in part a) with STEAM filters outlined in part b). In Figure 3a), a producer P matches an event e against each of its content filter cf to determine the list of consumers C to which to propagate e. For example, e_1 matches cf_X and is therefore propagated and delivered to C_1. Likewise, e_2 matches cf_Y and is delivered to C_2. e_3 matches cf_Z, a filter that has been deployed by both C_2 and C_3, and is delivered to both C_2 and C_3. In contrast, P in Figure 3b) holds a lookup table containing its subject filters sf, each of which relates to one of the types of event that may be produced. For example, e_1 of subject A is being propagated to the consumers C_1 and C_2 and e_2 of subject B is being propagated to the consumers C_2 and C_3. Once a consumer receives an event, it uses the event's subject as the key for accessing its lookup table to retrieve the relevant content filter cf. e is only delivered to the application if it matches cf. e_1 of subject A is only delivered to C_1's application if it matches cf_X and to C_2's application if it matches cf_Y. In summary, we argue that our approach of deploying subject and content filters on the producer and the consumer side respectively, allows a system to take advantage of the efficiency of subject filters and the expressiveness of content filters while supporting predicable algorithms for matching events and filters. Moreover, the computational load of matching events against filters is distributed between a producer and the subscribed consumers.

5. Conclusion and Future Work

We have outlined our work on STEAM, an event-based middleware service that has been designed for the mobile computing domain utilizing wireless local area networks and the ad hoc network model. We have outlined and discussed our design regarding the chosen event model and the underlying group communication mechanism and concluded that STEAM must omit application components providing system-wide services. We argue that in the envisaged traffic management application domain entities are more likely to interact once they are in close proximity and propose a novel approach of event filtering by combining three different types of event filter. This approach is well suited for systems that consist of a potentially large number of dynamically joining and leaving entities whose scale may change over time. Although we have addressed some of the fundamental issues arising when applying event-based middleware to the mobile computing domain, certain issues remain open for future work. Failed and temporary unavailable entities and connections are the norm rather that the exception in wireless networks compared to wired environments. A mechanism needs to be provided that anticipates entities and connections that may fail and subsequently handles partitioned proximities.

References

[1] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, and A. Hopper. Implementing a sentient computing system. *IEEE Comp.*, 34(8):50–56, 2001.

[2] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri. Generic support for distributed applications. *IEEE Computer*, 33(3):68–76, 2000.

[3] G. Banavar, T. Chandra, R. Strom, and D. Sturman. A case for message oriented middleware. In *DISC'99*, Bratislava, Slovak Republic, 1999.

[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):283–331, 2001.

[5] F. Cristian. Synchronous and asynchronous group communication. *Communications of the ACM*, 39(4), 1996.

[6] B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai. IEEE 802.11 wireless local area networks. *IEEE Communications Magazine*, pages 116–126, 1997.

[7] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE TSE*, 27(9):827–850, 2001.

[8] M. Haahr, R. Meier, P. Nixon, V. Cahill, and E. Jul. Filtering and scalability in the ECO distributed event model. In *PDSE/ICSE2000*, pages 83–95. Limerick, Ireland, 2000.

[9] T. Harrison, D. Levine, and D. Schmidt. The design and performance of a real-time CORBA event service. In *OOPSLA'97*, pages 184–200. Atlanta, Georgia, USA, 1997.

[10] M. O. Killijian, R. Cunningham, R. Meier, L. Mazare, and V. Cahill. Towards group communication for mobile participants. In *POMC01*, pages 75–82. Newport, RI, USA, 2001.

[11] R. Meier and V. Cahill. Taxonomy of distributed event-based programming systems. In *ICDCS/DEBS'02*. Vienna, 2002.

[12] Object Management Group. *CORBAservices: Common Object Services Specification - Notification Service Spec.* 2000.

[13] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with elvin4. In *AUUG2K*. Canberra, Australia, 2000.

[14] M. Weiser. Ubiquitous computing. *IEEE Hot Topics*, 26(10):71–72, 1993.