Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications

René Meier and Vinny Cahill

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

Abstract. Middleware supporting event-based communication is widely recognized as being well suited to mobile applications since it naturally accommodates a dynamically changing population of interacting entities and the dynamic reconfiguration of the connections between them.
STEAM is an event-based middleware designed for use in ad hoc networks. STEAM differs from other event-based middleware in that its architecture does not rely on the presence of any separate infrastructure, event notification filters are distributed, and filtering may be applied to functional and non-functional attributes. In particular, filters may be applied to either the subject or the content of an event notification, or to non-functional attributes, such as location and time. Filters may be used to define geographical areas within which event notifications are valid, thereby bounding the propagation of these notifications. Such proximity-based filtering represents a natural way to filter events of interest in mobile applications.
This paper describes the architecture and implementation of STEAM and its use of proximity-based filtering. In particular, we show how proximity-based filtering can be used to reduce the number of events delivered to collaborative mobile applications.

1 INTRODUCTION

Middleware supporting event-based communication [1] is widely recognized as being well suited to addressing the requirements of mobile applications [2, 3]. Event-based middleware naturally accommodates a dynamically changing population of components and is particularly useful in wireless networks, where communication relationships among application components are typically dynamically reconfigured during the lifetimes of the components.

Existing research on event-based middleware for wireless networks has mainly focused on what may be termed *nomadic applications*. These applications are characterized by fact that mobile nodes make use of the wireless network primarily to connect to a fixed network infrastructure, such as the Internet, but may suffer periods of disconnection while moving between points of connectivity. Such applications typically employ *infrastructure networks* [4]. As a result, most of this work has concentrated on handling disconnection while entities move from one access point to another. In contrast, we focus on *collaborative applications* characterized by the fact

that mobile nodes use the wireless network to interact with other mobile nodes that have come together at some common location. Although these applications may use infrastructure networks, they will often use *ad hoc networks* [4] to support communication without the need for a separate infrastructure. Consequently, this collaborative style of application allows loosely coupled components to communicate and collaborate in a spontaneous manner.

In this paper, we present the architecture and implementation of STEAM (Scalable Timed Events And Mobility), an event-based middleware for mobile computing, and outline how its features address the functional and non-functional requirements of such collaborative applications with a special emphasis on support for the use of ad hoc networks.

STEAM has been designed for IEEE 802.11b-based, wireless local area networks and is intended for applications that include a large number of highly mobile application components typically distributed over a large geographical area. Unanticipated interaction between nearby components is supported, enabling a component to dynamically establish connections to other components within its current vicinity. This allows components representing real world objects currently located within the same geographical area to deliver events at the location where they are relevant.

We envisage STEAM being utilized by collaborative applications in various domains including indoor and outdoor smart environments, augmented reality, and traffic management. In a traffic management application scenario, application components may represent mobile objects including cars, buses, fire engines, and ambulances as well as objects with a fixed location, such as traffic signals and lights. When within close proximity, such components may interact using STEAM in order to exchange information on the current traffic situation. As a simple example, an ambulance might disseminate its location to the vehicles traveling in front of it in order to have them yield the right of way. In general, inter-vehicle communication may contribute to better driver awareness of nearby hazards and is likely to lead to safer driving.

The STEAM event-based middleware has a number of important differences from other event services that support mobility [1, 2, 5, 6]:

- STEAM assumes an ad hoc network model supporting very dynamic coupling between application components.
- The architecture of STEAM is inherently distributed. The middleware is exclusively collocated with the application components and does not rely on the presence of any infrastructure.
- Application components are location aware. Geographical location information is provided by a location service and used to deliver events at the specific location where they are relevant.
- Distributed event notification filtering. Event notifications may be filtered at both the producer and the consumer side or may be filtered implicitly. Filters may be applied to functional and non-functional attributes associated with an event notification including subject, content, and geographical location.

The STEAM middleware is fully distributed over the same physical machines as the components that comprise a collaborative application. This implies that the middleware located on every machine has identical capabilities allowing its

components either to initiate or respond to communication. STEAM's architecture contains neither centralized components, such as lookup and naming services, nor the kind of intermediate components that are used by other event services to propagate event notifications from event producers to event consumers [1, 2, 6-8]. Generally, dedicated machines that are part of the event service infrastructure are used to host such components in order to ensure that they are accessible to all application components in a system at any time. However, this approach is impractical in ad hoc environments due to lack of infrastructure and the possibility of network partition.

STEAM supports distributed event notification filters that may be applied to the functional and non-functional attributes of an event notification. Functional attributes include the subject and content of an event notification, whereas non-functional attributes may include context, such as the geographical location of a component, time, and the Quality of Service (QoS) available from the network. Combining distributed event notification filters, which may be applied on both the producer and the consumer side, enables a subscriber to describe the exact subset of event notifications in which it is interested, exploiting multiple criteria, such as meaning, geographic location, and time. For example, filters that are applied to location information allow application components to interact based on their current location; an event producer may define a geographical area within which certain event notifications are valid thereby bounding the area within which these event notifications are propagated. STEAM provides location filters, called proximity filters, that differ from traditional filters in that they are not inherently located on either the producer or the consumer side. Producers and consumers may both apply location filters to determine whether their current location is within the geographical scope of certain event notifications. STEAM exploits group communication, which has been recognized as a natural means to support event-based communication [9], as the underlying mechanism for components to interact. However, STEAM's approach differs from the traditional approach in that it utilizes a group communication mechanism based on proximity [10] enabling the mapping of location filters describing geographical scope to proximity groups.

We argue that the STEAM architecture and our approach to distributed event notification filtering helps to improve system scalability by omitting centralized components and by bounding the propagation of subscription information and event notifications. This reduces the use of communication and computation resources, which are typically scarce in mobile environments. In general, distributed event filtering limits the number of filters being applied at a particular location and balances the computational load of filter matching between the physical machines in a system. The number of producer side filters is independent of the potentially large number of subscribers and the number of consumer side filters depends solely on the number of local subscribers. As a result, filter evaluation time can be bounded for the events disseminated in a particular scope.

The reminder of this paper is structured as follows: Section 2 surveys related work. Section 3 focuses on the programming model supported by STEAM. Section 4 presents the STEAM architecture including the main middleware components and the employed communications model. Section 5 presents our initial evaluation of STEAM, which demonstrates how distributed event filtering reduces the number of

events delivered to an application. Section 6 concludes this paper by summarizing our work and outlining the issues that remain open for future work.

2 RELATED WORK

Middleware supporting event-based communication has been developed by both industry [7, 11] and academia [1, 2, 8, 12]. Most such middleware assumes that the components comprising an application are stationary and that a fixed network infrastructure is available. Existing research on event-based middleware for mobile computing has mainly focused on supporting nomadic applications using wireless data communication based on the infrastructure network model [1, 2, 5, 6]. Relatively little work has been done to address the distinct requirements, due to the lack of infrastructure, associated with supporting event-based communication in ad hoc networks. Application components using the ad hoc network cannot rely on the aid of access points when discovering peers in order to establish connections to them. Event messages can neither be routed through access points nor rely on the presence of intermediate components that may apply event filters or enforce non-functional attributes such as ordering policies and delivery deadlines.

In the remainder of this section, we introduce various event models that offer mobility support and briefly outline their respective architectures. JEDI [2] allows nomadic application components to produce or consume events by connecting to a logically centralized event dispatcher that comprises the event service infrastructure and has global knowledge of all subscription requests and events. JEDI provides a distributed implementation of the event dispatcher consisting of a set of dispatching servers that are interconnected through a fixed network. Nomadic entities may move using the `moveOut` and `moveIn` operations. The `moveOut` operation disconnects the entity from its current dispatching server, allowing it move to another location and then to use the `moveIn` operator to reconnect to another dispatching server at a later time. The dispatching server buffers all relevant information while an entity is disconnected and forwards it upon reconnection. Mobile Push [5] proposes a similar approach to supporting nomadic application components in which entities do not use the event service while moving. In addition, it supports mobile application components accessing the event service infrastructure through wireless connections while moving. However, both approaches rely on the presence of a separate event service infrastructure. Elvin [6] implements support for mobility through the use of a proxy server that maintains a permanent connection to the event server on behalf of nomadic client components. The proxy server stores events while a client is temporarily disconnected until the client reconnects. The proxy server allows clients to specify a time to live for each subscription to prevent large numbers of events being stored indefinitely. Clients must explicitly connect to a proxy server using a URL and must reconnect to the same proxy server each time.

Although these middleware services support mobility, their main goal is to handle disconnection while an entity moves from one access point to another. In contrast, STEAM accommodates a changing set of collaborative entities coming together at a

location and supports communication between these entities without relying on a separate event service infrastructure.

3 STEAM ARCHITECTURE

The design of the STEAM architecture is motivated by the hypothesis that there are applications in which mobile components are more likely to interact once they are in close proximity. This means that the closer event consumers are located to a producer the more likely they are to be interested in the events that it produces. Significantly, this implies that events are relevant within a certain geographical area surrounding a producer. For example, in augmented reality games players are interested in the status of game objects or indeed other players, only when they are within close proximity. An example from the traffic management domain might be a crashed car disseminating an accident notification. Approaching vehicles are interested in receiving these events only when located within a certain range of the car.

3.1 Using Event Types and Proximities

STEAM implements an implicit event model [13] that allows event producers to publish events of a specific *event type* and consumers to subscribe to events of particular event types. Producers may publish events of several event types and consumers may subscribe to one or more event types.

To facilitate the kind of location-aware application described above, STEAM supports a programming model that allows producers to bound the range within which their events are relevant. Producers *announce* the type of event they intend to *raise* together with the geographical area, called the *proximity*, within which events of this type are to be disseminated. Such an announcement associates a specific event type with a certain proximity and implicitly bounds event propagation. Consumers receive events only if they reside inside a proximity in which events of this type are raised.

Producers may define proximities independently of their physical transmission range with the underlying group communication system routing event messages from producer to consumer using a multi-hop protocol. Proximities may be of arbitrary shape and may be defined as nested and overlapping areas. Nesting allows a large proximity to contain a smaller proximity subdividing the large area. Fig. 1 depicts two overlapping proximities of different shape and illustrates that multiple consuming and producing entities may reside inside a proximity. These proximities have been associated with events of *type A* and *type B* respectively. Consequently, consumers handling these event types receive events if they reside inside the appropriate proximity. Note that entities located inside these areas handling other event types will not affect the propagation of these events. An example of overlapping proximities might include a car disseminating an accident notification within the vicinity of a traffic light propagating its status to approaching vehicles.

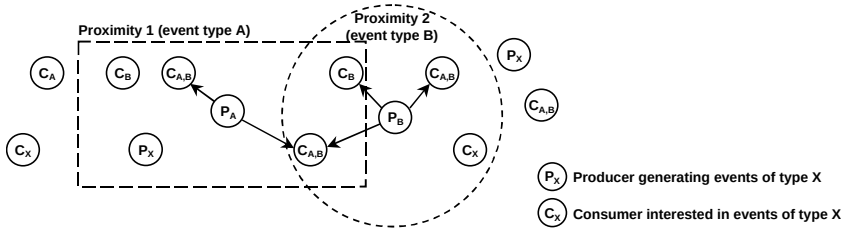

Fig. 1. Disseminating events using event types and proximities

3.2 Supporting Mobility

STEAM has been designed to support applications in which application components can be either stationary or mobile and interact based on their geographical location. This implies that the STEAM middleware as well as the entities hosted by a particular machine are aware of their geographical location at any given time. STEAM includes a location service that uses sensor data to compute the current geographical location of its host machine and entities. To suit outdoor applications, for example in the traffic management domain, STEAM exploits a version of the location service that uses a GPS satellite receiver to provide latitude and longitude coordinates.

In addition to supporting stationary and mobile entities STEAM allows proximities to be either stationary or mobile. A stationary proximity is attached to a fixed point in space whereas a mobile proximity is mapped to a moving position represented by the location of a specific mobile producer. Hence, a mobile proximity moves with the location of the producer to which it has been attached. This implies that mobile consumers and producers may be moving with a mobile proximity. For example, a group of vehicles heading in the same direction may cooperate to form a platoon in order to reduce their consumption of fuel. These vehicles might interact using a proximity that has been defined by the leading vehicle. Such a proximity might be attached to the position of the leader moving with its location.

3.3 Subscribing to Event Types

Consumers must subscribe to event types in order to have the middleware deliver subsequent events to them if they are located inside any proximity where events of this type are raised until they unsubscribe. A consumer may move from one proximity to another without re-issuing a subscription when entering the new proximity. Thus, subscriptions are persistent and will be applied transparently by the middleware every time a subscriber enters a new proximity. This implies that a subscription to a specific event type applies to all proximities handling these events even though the subscriber may only receive a subset of these events at any time. A single subscription may result in events of a particular event type raised by different producers in multiple proximities being delivered. Hence, the set of events received by a subscriber at a certain time depends on its movements as well as on the movements of producers and proximities.

3.4 Defining Event Types

Applications define event types to specify the functional and non-functional attributes of the events they intend to disseminate. Fig. 2 illustrates that a STEAM event type consists of *subject* and *content* representing its functional attributes, as well as of a self-describing *attribute list* representing its non-functional attributes. The subject defines the name of a specific event type and the content defines the names and types of a set of associated parameters. STEAM event instances are defined in a similar manner by specifying a subject, content parameter values, and attribute list. Producers and consumers must use a common vocabulary defined by the application to agree on the name of an event type. An event type and an event instance that have the same subject must have an identical content structure, i.e., the set of parameter names and types must be consistent. This approach allows an application to associate non-functional attributes to events of the same type as well as to individual event instances. An event type may include attributes such as proximity and ordering semantics, whereas event instance attributes may include event priority, temporal validity, and delivery deadline. As described in more detail below, distributed event filters may be applied to the subject, content, and attribute list defined by either an event type or an event instance.

STEAM event type = {subject, content_name_type, attribute_list}
STEAM event instance = {subject, content_value, attribute_list}

Fig. 2. STEAM event type and instance definition

3.5 Applying Event Notification Filters

STEAM supports three different event filters, namely *subject filters*, *content filters*, and *proximity filters*. These filters may be combined and a particular event is only delivered to a consumer if all filters match. Subject filters match the subject of events allowing a consumer to specify the event type in which it is interested. Content filters contain a filter expression that can be matched against the values of the parameters of an event. Content filters are specified using filter expressions describing the constraints of a specific consumer. These filter expressions may contain equality, magnitude and range operators as well as ordering relations. They may include variable, consumer local information such as the consumer's location. Proximity filters are location filters that define the geographic scope within which events are relevant and correspond to the proximity attribute associated with an event type.

4 COMMUNICATIONS ARCHITECTURE

The design of the STEAM communications architecture is motivated by our approach of bounding the scope within which certain information is valid and by the characteristics of the underlying wireless network. We employ a transport mechanism based on group communication and use a multicast protocol to route messages between the participants.

4.1 Using Proximity Groups

Group communication [14] has been recognized as a natural means to support event-based communication models [9]. Groups provide a one to many communication pattern that can be used by producers to propagate events to a group of subscribed consumers. STEAM exploits a *Proximity-based Group Communication Service* (PGCS) [10] as the underlying means for entities to interact. Proximity groups have been designed to support mobile applications using wireless local area networks [10]. 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, i.e., a group is identified by both geographical and functional aspects. 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, whereas the geographical aspect outlines the bounded scope within which the information is valid. Hence, STEAM maps subject and proximity to the functional and geographical aspect of proximity groups respectively. Furthermore, proximity groups can be either absolute or relative. An absolute proximity group is geographically fixed; it is attached to a fixed point in space. In contrast, a relative proximity group is attached to a moving point represented by a specific mobile node.

4.2 Locating Proximity Groups

Instead of requiring a naming service to locate entities that wish to interact, STEAM provides a discovery service that uses beacons to discover proximities. Once a proximity has been discovered, the associated events will be delivered to subscribers that are located inside the proximity. This service is also responsible for mapping discovered proximities to subscriptions and to the underlying proximity-based communication groups. Hence, it causes the middleware to join a proximity group of interest, i.e., for which it has either a subscription or an announcement, once the host machine is within the associated geographical scope and to leave the proximity group upon departure from the scope.

4.3 Mapping to Proximity Groups

Mapping announcements and subscriptions to groups requires a means to uniquely identify a group as well as for consuming and producing entities to retrieve the identifier of the specific group that disseminates certain events. Unique group identifiers are traditionally generated either statically or using global knowledge by means of a centralized lookup service. STEAM implements an addressing scheme in which identifiers representing groups can be computed from subject and proximity pairs. Each combination of subject and proximity (shape, dimension, and location) is unique throughout a system. The description of such a pair is used as stimulus for a hashing algorithm to dynamically generate identifiers using node local rather than

global knowledge. Upon discovery of a proximity and the associated subject, producing and consuming entities compute the corresponding group identifier if the subject is of interest. This scheme allows entities to subsequently use these identifiers to join groups in which relevant events are disseminated. Moreover, it prevents entities that are not interested in certain events from joining irrelevant groups and as a result, from receiving unwanted events even though they might reside within the proximity of the group.

4.4 Mapping to Ad Hoc Networks

STEAM allows entities to define geographical scopes independently of the physical transmission range of these wireless transmitters. Consequently, STEAM supports multi-hop event dissemination in which nodes residing within the boundaries of a proximity forward event messages. Members of the corresponding multicast group recognize the identifiers of these event messages and subsequently deliver them. Nodes residing outside a proximity will discard event messages that they cannot associate with any proximity known to them.

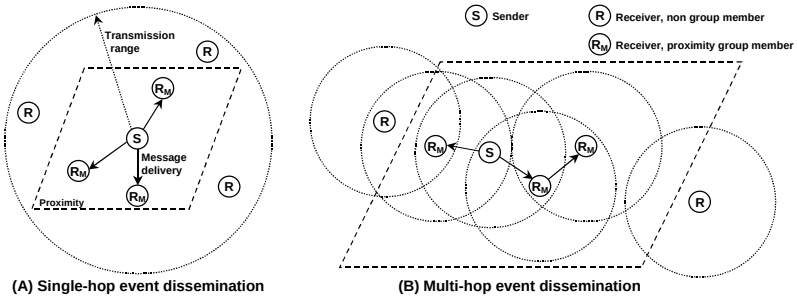

Fig. 3. Event dissemination scenarios

Fig. 3 (A) outlines a single-hop event propagation scenario where the transmission range of the sender covers the entire scope of the proximity. Event messages are propagated within the transmission range and member nodes will deliver them. **Fig. 3** (B) shows a multi-hop event propagation scenario in which the proximity exceeds the transmission range of the sender. The maximum number of hops a message may travel to reach any member of the group is bounded by the proximity.

5 EXPERIMENTAL RESULTS

This section presents our initial evaluation of STEAM, which demonstrates how bounding the propagation range of event notifications using proximity-based filtering can be used to reduce the number of events delivered to an application. A prototypical application scenario from the traffic management domain has been implemented for this experiment. The scenario simulates the interaction between vehicles passing through an intersection and the intersection's traffic light disseminating its light

status. The scenario is modeled according to the intersection of North Circular Road (NCR) and Prussia Street (PST) located in Dublin's inner city. It is based on real data, which has been provided by Dublin City Council, describing vehicle movements and light status at the intersection over a period of 24 hours.

Fig. 4 (A) illustrates the intersection and outlines how the traffic flow can be broken up into two distinct phases. The intersection comprises two *approaches*; approach one describes the traffic flows arriving from east and west whereas approach two describes the traffic flows arriving from north and south. Approach one comprises three lanes and approach two comprises of four lanes. The traffic light for both approaches is located in the center of the intersection at the stated latitude and longitude. The *cycle* time defines the duration for the two approaches to complete their respective sequences of light changes. The proportion of the cycle length that is assigned to one particular approach is called the *split*. The split between the phase of approach one and two is 45% to 55%. The intersection data was acquired over a period of 24 hours starting on the 3rd of December 2002 at 6 pm. It consists of a sequence of records, each describing a cycle duration and the number of vehicles passing through the intersection on each individual lane during the cycle.

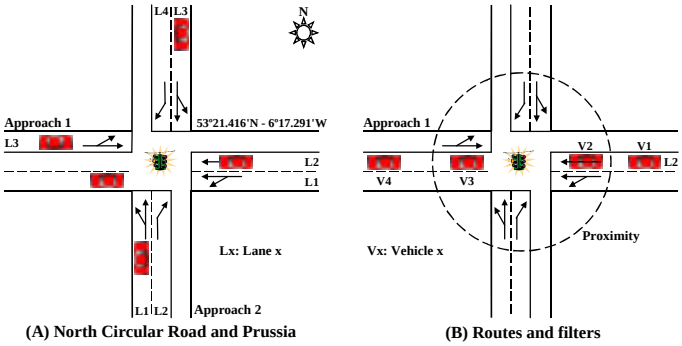

Fig. 4. Modeling the intersection

The experiment includes three notebook computers placed 5 meters apart communicating through wireless ad hoc connections, each equipped with a Lucent Orinoco Gold WiFi PCMCIA card. One machine hosts the traffic light and the other two the vehicles arriving on approach one and two respectively. The traffic light raises an event every second for each approach to disseminate the light status, approach name, and light location. Vehicles approach the intersection in their respective lanes at an average speed of 25 miles per hour (the intersection is located in a 30 miles per hour zone). Each vehicle follows a pre-defined route according to its approach lane simulated by its location service. **Fig. 4** (B) depicts an example route of a vehicle in lane two of approach one. The available intersection data does not describe the behavior of an approaching vehicle in terms of queuing; it only indicates the number of vehicles passing the intersection during a green light sequence. Hence, vehicles are modeled to reflect this behavior arriving at the intersection in time to pass the light during a green light sequence.

Fig. 4 (B) also illustrates the use of distributed filters in the simulation. The traffic light announces events of type "Traffic Light" and the associated proximity, which

defines the radius of the area of interest surrounding the traffic light. The radius has been set to 40 meters to allow for vehicle breaking distance (16 meters) and update rate (once per second) of the location service. This radius guarantees that an approaching vehicle receives at least two events before having to decide whether or not to stop at the light. Vehicles subscribe to "Traffic Light" events and define a content filter that matches events on their approach when they are moving towards the traffic light. This combination of filters causes vehicles one and four to discard "Traffic Light" events as they reside outside the scope of the proximity. Even though vehicles two and three are inside the proximity, only vehicle two will deliver "Traffic Light" events to its application. The content filter of vehicle three prevents event delivery since the vehicle is moving away from the traffic light.

This experiment comprises two runs using the same stimuli. The first run applies distributed events filters as described above whereas the second run lacks any filters assuming the communication range of the traffic light's wireless transmitter to limit event dissemination. We assume the radio transmission range in the modeled urban environment to be 200 meters.

	Approach 1			Approach 2			
	Lane 1	Lane 2	Lane 3	Lane 1	Lane 2	Lane 3	Lane 4
Number of Vehicles	6652	3320	3728	3038	1383	2802	1135
No Event Filter	469945	235093	264557	210114	95743	193931	78730
Distributed Event Filter	24139	11905	13553	9436	4488	8805	3543
Relative Decrease	94.9%	94.9%	94.9%	95.5%	95.3%	95.5%	95.5%

Table 1. Results of the experiment

Table 1 summarizes the number of vehicles passing through the intersection on each lane as well as the total number of events delivered to these vehicles in each experiment. It also includes the relative decrease of delivered events between the two experiments. The data in **Table 1** shows a substantial reduction, averaging at around 95%, in the number of events delivered to the vehicles when applying distributed event filters. This is hardly surprising considering the bounding of the propagation range and the content filter discarding events once a vehicle has passed the traffic light. Applying distributed event filters in experiment one causes additional overhead due to proximity discovery. The traffic light announces its proximity by propagating beacons within the proximity area. Vehicles discover the proximity upon entering the area delivering a single beacon message to the middleware. Assuming this being equivalent to delivering an additional event per vehicle would reduce the relative decrease by approximately 1.4%. However, the overall number of events delivered would still substantially decrease, on average by over 93%.

6 CONCLUSIONS

We have described the architecture and implementation of STEAM, an event-based middleware for collaborative, location aware applications using wireless local area networks. STEAM differs from other event-based middleware in that its architecture

does not rely on the presence of any infrastructure, event notification filtering may be distributed and may be applied to functional and non-functional attributes of an event notification. Our initial evaluation of STEAM shows that using proximity-based filtering to bound the propagation range of event notifications reduces the number of events delivered to an application. We plan to further evaluate our work using a variety of prototypical application scenario implementations.

Acknowledgements. 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).

The authors would like to thank Dublin City Council for providing the traffic data that made our evaluation of STEAM possible.

References

[1] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri, "Generic Support for Distributed Applications," IEEE Computer, vol. 33, pp. 68-76, 2000.

[2] G. Cugola, E. D. Nitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and its Application to the Development of the OPSS WFMS," IEEE Transactions on Software Engineering (TSE), vol. 27, pp. 827-850, 2001.

[3] R. Meier, "Communication Paradigms for Mobile Computing," ACM SIGMOBILE Mobile Computing and Communications Review (MC2R), vol. 6, pp. 56-58, 2002.

[4] B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai, "IEEE 802.11 Wireless Local Area Networks," IEEE Communications Magazine, pp. 116-126, 1997.

[5] I. Podnar, M. Hauswirth, and M. Jazayeri, "Mobile Push:Delivering Content to Mobile Users," in Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02). Vienna, Austria, 2002, pp. 563-570.

[6] P. Sutton, R. Arkins, and B. Segall, "Supporting Disconnectedness – Transparent Information Delivery for Mobile and Invisible Computing," in Proceedings of IEEE CCGrid 2001. Brisbane, Australia: 2001, pp. 277-285.

[7] Object Management Group, CORBAservices: Common Object Services Specification - Notification Service Specification, Version 1.0: Object Management Group, 2000.

[8] A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," ACM Trans. on Computer Systems, vol. 19, pp. 283 - 331, 2001.

[9] G. Banavar, T. Chandra, R. Strom, and D. Sturman, "A Case for Message Oriented Middleware," presented at Proceedings of the 13th International Symposium on DIStributed Computing (DISC'99), Bratislava, Slovak Republic, 1999.

[10] M. O. Killijian, R. Cunningham, R. Meier, L. Mazare, and V. Cahill, "Towards Group Communication for Mobile Participants," in Proceedings of Principles of Mobile Computing (POMC'2001). Newport, Rhode Island, USA, 2001, pp. 75-82.

[11] Sun Microsystems Inc., Java Distributed Event Specification. 1998.

[12] M. Haahr, R. Meier, P. Nixon, V. Cahill, and E. Jul, "Filtering and Scalability in the ECO Distributed Event Model," in Proc. of the 5th Int. Symposium on Software Engineering for Parallel and Distributed Systems (ICSE/PDSE2000). Limerick, Ireland. 2000, pp. 83-95.

[13] R. Meier and V. Cahill, "Taxonomy of Distributed Event-Based Programming Systems," in Proceedings of ICDCS/DEBS'02. Vienna, Austria, 2002, pp. 585-588.

[14] F. Cristian, "Synchronous and Asynchronous Group Communication," Communications of the ACM, vol. 39, pp. 88-97, 1996.