Location-Aware Event-Based Middleware: A Paradigm for Collaborative Mobile Applications?

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

***Abstract*—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. Relatively little work has been done to accommodate collaborative applications that often use ad hoc networks.**

This paper describes a novel kind of event-based middleware, called STEAM, that has been 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 and event notification 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 collaborative mobile applications.

I. INTRODUCTION

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 [1-3].

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 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, 5-7]:

- 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, 6-9]. 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 especially 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 [10], 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 [11] enabling the mapping of location filters describing geographical scope to proximity groups.

The reminder of this paper presents the STEAM programming model and architecture including the main middleware components and the employed communications model.

II. 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.

A. Using Event Types and Proximities

STEAM implements an implicit event model [12] 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 underling 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.

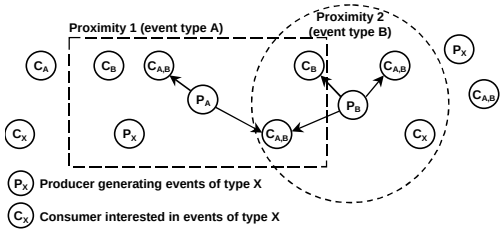

Fig. 1. Disseminating events using event types and proximities.

B. Supporting Location Awareness and 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. Mobile application components may move within the scope of a specific proximity and may move between proximities. 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.

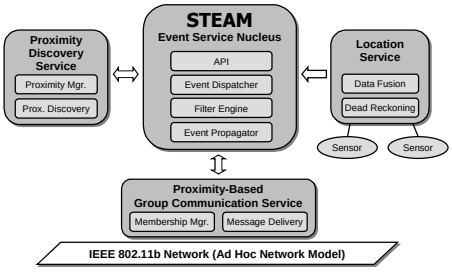

Fig. 2. STEAM architecture overview.

As shown in Fig. 2, STEAM includes a *Location Service* (LS) that uses sensor data to compute the current geographical location of its host machine and subsequently provides this location information to the middleware and to event producers and consumers running on that machine. To suit outdoor applications, for example in the traffic management domain, STEAM exploits a version of the LS 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.

C. 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.

In summary, STEAM's approach to proximity-based event dissemination allows consumers to specify the event types in which they are interested and producers to define the scope within which events of a specific type are relevant. In principal, either a consuming or a producing entity may define a proximity. However, we believe that in many application scenarios it is the producer that would bound the scope within which its events are relevant. For example, a traffic light propagating its status to approaching vehicles defines its proximity based on the location of the next traffic light and on the local speed limit.

D. Defining Event Types

Applications define event types to specify the functional and non-functional attributes of the events they intend to disseminate. Fig. 3 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. Furthermore, Fig. 3 outlines that a STEAM event instance is 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 and to individual event instances. An event type may include attributes such as proximity, ordering semantics, and QoS requirements, 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. 3. STEAM event type and instance definition.

E. Appying 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.

III. 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.

A. Using Proximity Groups

Group communication [13] has been recognized as a natural means to support event-based communication models [10]. Groups provide a one to many communication pattern that can be used by producers to propagate events to a group of subscribed consumers. As shown in Fig. 2, STEAM exploits a *Proximity-based Group Communication Service* (PGCS) [11] as the underlying means for entities to interact. Proximity groups have been designed to support mobile applications using wireless local area networks [11]. 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. This notion of absolute and relative proximity groups serves as the basis for the stationary and mobile scopes supported by STEAM's programming model.

B. Locating Proximity Groups

Instead of requiring a naming service to locate entities that wish to interact, STEAM provides a *Proximity Discovery Service* (PDS) 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. As shows in Fig. 2, each node runs a PDS.

The PDS is also responsible for mapping discovered proximities to subscriptions and to the underlying proximity-based communication groups. Hence, the PDS 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.

C. 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.

D. 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.

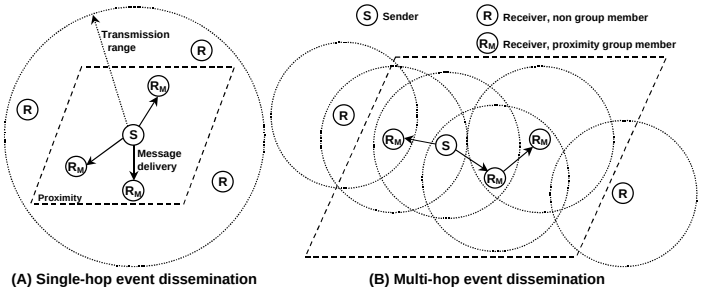

Fig. 4. Event dissemination scenarios.

Fig. 4 (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. 4 (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.

IV. CONCLUSIONS

We have described the architecture 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. Functional attributes include the subject and content of an event notification, whereas non-functional attributes may include context, such as geographical location of a component. STEAM's approach of exploiting location information allows components currently located within the same geographical area to deliver event notifications at the location where they are relevant.

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).

REFERENCES

[1] 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.

[2] Y. Huang and H. Garcia-Molina, "Publish/Subscribe in a Mobile Environment," in *Proceedings of the Second ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDe'01)*. Santa Barbara, CA, USA, 2001, pp. 27-34.

[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 the IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001)*. Brisbane, Australia: IEEE CS Press, 2001, pp. 277-285.

[7] 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.

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

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

[10] 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.

[11] 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.

[12] R. Meier and V. Cahill, "Taxonomy of Distributed Event-Based Programming Systems," in *Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02)*. Vienna, Austria, 2002, pp. 585-588.

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