IIP: An Event-Based Platform for ITS Applications

Shuo Ma, Ouri Wolfson
Department of Compute Science
University of Illinois at Chicago
Chicago, U.S.A.
{sma,wolfson}@cs.uic.edu

Jie Lin
Department of Civil and Materials Engineering
University of Illinois at Chicago
Chicago, U.S.A.
janelin@uic.edu

ABSTRACT
In this paper, we propose an event-based platform in support of developing Intelligent Transportation System (ITS) applications, especially applications compatible with the IntelliDrive^{SM} environment. Essentially, it combines two components, i.e. the IIP Cloud Component and the IIP Client Component. The IIP Cloud Component provides canonical publish/subscribe (pub/sub) functions to both traffic management facilities and mobile nodes. The IIP Client Component provides mobile nodes with pub/sub functions which allow them to communicate with the IIP Cloud Component as well as with each other. We show that by leveraging heterogeneous data sources and various communication mechanisms in the ITS environment, IIP supports a wide variety of ITS applications.

Categories and Subject Descriptors
H.3.4 [**Information Storage and Retrieval**]: Systems and software; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design;

General Terms
Design

Keywords
Event, ITS Information System, Event Schema Registry, Publish/Subscribe, Peer-to-Peer, Computational Transportation Science (CTS)

1. INTRODUCTION
ITS has garnered more and more attention in recent years. Authorities and affiliated organizations around the world have established programs to expedite the realization and deployment of ITS, e.g. eSafety initiative [5] and PReVENT initiative [6] funded by European Commission, VICS [7] in Japan, and TAIWAN iTS1 [4]. Among various ITS initiatives in US, *IntelliDrive^{SM}* is a most recent effort initiated by the U.S. Department of Transportation (U.S. DOT) that focuses on advancing connectivity among vehicles and roadway infrastructure in order to significantly improve safety and mobility of surface transportation systems [8]. IntelliDrive^{SM} operates in a hybrid environment consisting of the infrastructure network and hundreds and thousands of clients. Specifically, the infrastructure network connects fixed roadside sensors and management facilities, and includes the cellular phone network as well. Most of the clients refer to mobile nodes involving vehicles, such as passenger cars, transit buses, emergency vehicles, etc., and pedestrians and bicyclists equipped with wireless consumer devices. However, we also consider some static nodes, e.g. desktops, as clients. This paper assumes that each mobile node can communicate with each other as well as the infrastructure network via standard wireless technologies, e.g. DSRC and Wi-Fi. IntelliDrive^{SM} is anticipated to ultimately encompass a wide variety of ITS applications ranging from safety to environmental monitoring.

Envisioning that there are many common data management and communication elements shared by various prospective IntelliDrive^{SM} applications, in this paper we propose the *ITS Information Platform (IIP)* as a common data management and communication services platform facilitating and easing applications development. The motivation of IIP comes from the fact that there is a lack of a generic platform, which provides data management and communication support capability in a distributed, heterogeneous data environment like IntelliDrive^{SM}. As an example, consider how an application helps a driver who is interested in the current and expected traffic speed on the Kennedy Expressway receive the information. The application first identifies the relevant information that contains or may be used to infer traffic speed on the Kennedy Expressway. Traffic reports providing exact speeds are the ideal source. A picture or a video clip of the expressway may be useful as well. Special events announcements about sports events, roadwork, accidents, and extreme weather warnings are also relevant. The issue is that such information may exist in different forms, in a web post or personal text communication, on the Internet or a hand-held devices, etc. And the solution to locating the data dictates the approach to accessing the data.

Thus a common data management and communication services platform like IIP is of great need to ease the burden on the application developers and users from having to deal with issues such as finding the data discussed above. We argue that the publish/subscribe (pub/sub) paradigm [1] is an appropriate fundamental building block for IIP. Publish/Subscribe is an asynchronous communication paradigm that provides spatiotemporally loosely-coupled connection between communicating parties. Moreover, it is multicasting in nature since data producers can send the same data to multiple consumers with a single operation. As a result, the asynchrony

IWCTS'10, Nov 2, 2010, San Jose, CA, U.S.A.

and one-to-many characteristics make the publish/subscribe system more suitable than other communication paradigms, e.g. query-response and remote procedure call, for distributed information dissemination applications which are prevalent in the ITS environment. This paper describes the components, their functionality, and the architecture necessary for the pub/sub based IIP platform.

The remainder of this paper is structured as follows. Section 2 gives an overview of the architecture of IIP. In section 3, a key element of our proposal, namely the Event Schema Registry, is described. Section 4 discusses the Event Broker component, followed by related work in section 5. Finally conclusion and future work are given in section 6.

2. ARCHITECTURE of IIP

At the highest level, IIP is composed of two components, one residing on the cloud, i.e. the infrastructure network, which is referred as the *IIP Cloud Component*; and the other residing on the clients, which is referred as the *IIP Client Component*. The IIP Cloud Component provides canonical publish/subscribe functions to both traffic management facilities and mobile nodes. The IIP Client Component provides mobile nodes with pub/sub functions which allow them to communicate with the IIP Cloud Component as well as with each other.

Following the conventions in the pub/sub literature, we use the term *event* for any useful information and *publisher* and *subscriber* respectively for producer and consumer of the event in the rest of the paper. Figure 1 shows the overall architecture of IIP whose components are represented by dashed incarnadine boxes.

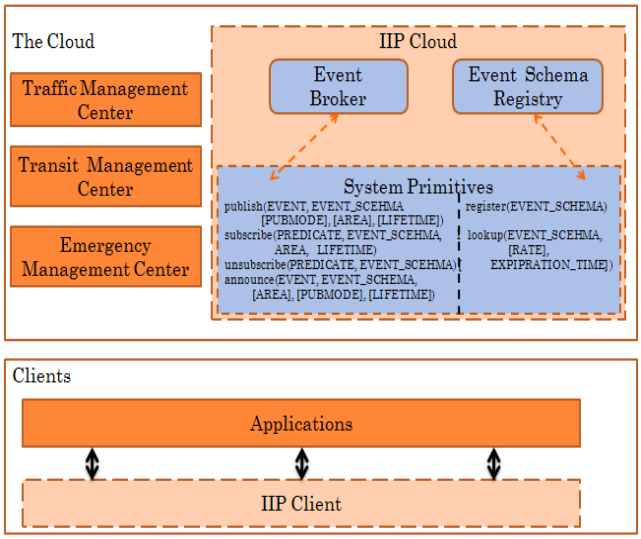

Figure 1. ITS Information Platform architecture

The IIP Cloud Component essentially consists of two building blocks, namely the *Event Broker* and the *Event Schema Registry (ESR)*. The Event Broker is a kernel function of large scale infrastructure-based publish/subscribe systems. It basically performs "store and forward" function to route events from publishers to subscribers. Upon the arrival of new events, the Event Broker sends the events to all the subscribers whose subscription matches the new event.

The Event Schema Registry provides a repository where publishers can find event schemas of interest. The *System Primitives* provide the open interface for applications running on either mobile nodes or management facilities to access the IIP Cloud Component. More detailed descriptions of the Event Schema Registry (Section 3) and the Event Broker (Section 4) will follow.

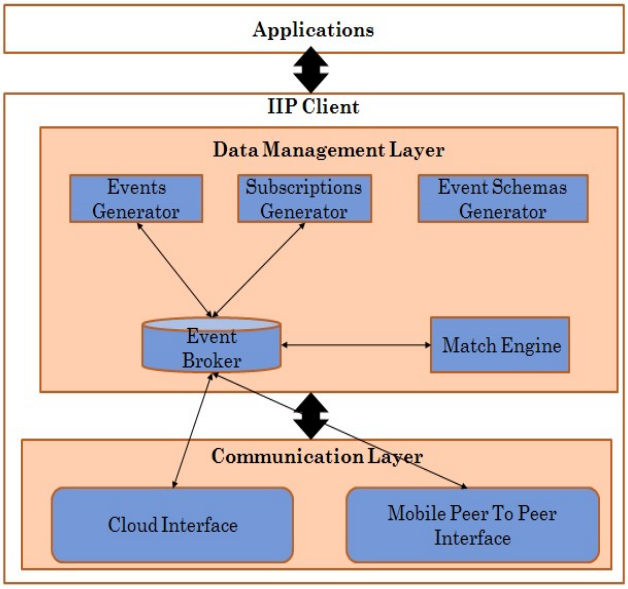

Figure 2. IIP Client architecture

Figure 2 shows the architecture of the IIP Client Component. It employs a two-layer structure. The upper level is the *Data Management Layer*. It provides applications with a uniform data abstraction and manipulation tool compatible with the IIP Cloud Component. Specifically, the *Event Schemas Generator* allows mobile nodes to produce event schemas to be registered to the ESR. The *Events Generator* and the *Subscriptions Generator* control the generation of events and subscriptions respectively. The Event Broker module uses the Match Engine to find matching pairs of publications and subscriptions, and it interfaces with the communication layer to receive and send publications and subscriptions generated at the client. Note that the IIP Client Component does not provide a schema discovery function, thus mobile nodes cannot learn event schemas of interest from anywhere else but the Event Schema Registry in the Cloud.

The lower level is the *Communication Layer*, which supports different communication paradigms. Currently two paradigms are considered. The *Cloud Interface* allows mobile nodes to access the IIP Cloud Component through Internet access points, e.g. Wi-Fi hotspots, cellular base stations or other road side units. Existing standard cellular system, e.g. GSM and GPRS, and works on Internet access protocols, e.g. fast Wi-Fi access protocols described in [24] are sufficient for realizing the Cloud Interface. Thus we do not discuss them further in the rest of the paper.

Notice that the Cloud Interface cannot support pub/sub function for purely mobile peer to peer applications in which the communication infrastructure does not exist, or is not fast enough. As an example, consider the application where the right-of-way is distributively maintained among vehicles, pedestrians and cyclists at a road intersection without traffic lights or human coordinators.

In such an application, mobile nodes approaching the intersection should coordinate with each other directly via short range high-bandwidth channels, since a cloud solution probably cannot guarantee the necessary response time.

Therefore we propose the *Mobile Peer to Peer (P2P) Interface* as a complementary communication method. It implements the pub/sub system purely via inter-node communication in a mobile P2P network (vehicles, smart-phones and other pedestrian/bicyclist devices), regardless of whether or not the mobile P2P network is connected. In other words, we assume that the Data Management Layer on IIP Client is not concerned with connectivity issues in the Mobile P2P network, and the communication layer will employ the appropriate Delay Tolerant protocols to overcome connectivity problems in the this network.

3. THE Event Schema Registry

The Event Schema Registry is implemented as a central directory which enables publishers to register their event schemas and provides schema search functionality to subscribers. From an application's point of view, it provides the following two essential primitives (optional parameters are enclosed by brackets):

- *register(EVENT_SCHEMA)*
- *lookup(EVENT_SCHEMA, [RATE], [EXPIRATION_TIME])*

3.1 The *register* Primitive

Each publisher registers an event schema with the Event Schema Registry by invoking the *register* primitive. The *EVENT_SCHEMA* parameter is formally represented in the format of SCHEMA_NAME(ATTRIBUTES), where SCHEMA_NAME and ATTRIBUTES represents the name and the set of attributes of the event to be registered by the publisher. Each attribute can be a primitive data type, such as a string, a numeric field, a spatiotemporal data structure such as a point, region, hour, day, or a complex data structure, such as an XML document. For a complete example of a schema, consider: "BUS LOCATION" with attributes (ROUTE_NO, RUN_NO, TIMESTAMP, LOCATION-COORDINATES).

3.2 The *lookup* Primitive

Each subscriber discovers the event schemas of interest from the Event Schema Registry by invoking the *lookup* primitive. The *EVENT_SCHEMA* parameter has the same format and semantics as in the *register* primitive. We treat each call to the *lookup* primitive as an instantaneous rather than a continuous query. Consequently, the Event Schema Registry saves considerable memory space by not keeping track of the history of received lookup calls.

Each call to the *lookup* primitive from an application is passed to the IIP Client Component, which in turn sends the call to the IIP Cloud via the Cloud Interface. The Event Schema Registry returns to the IIP Client Component a list of all event schemas that match the *EVENT_SCEHMA* parameter of the call. Here the matching is performed using both schema matching techniques [15] and ontology matching techniques [26].

Prior to displaying the returned list of event schemas from the ESR to the application, the IIP Client Component ranks the schemas based on some measures, such as reputation of the publisher of the event schema, if a reputation system is implemented; or based on the degree to which the lookup and register schemas match. Then the subscribers choose schemas on which they would like to write subscriptions.

The *RATE* parameter is introduced such that subscribers are constantly kept posted about new schemas of interest registered with the ESR. To understand why, consider the following scenario. Suppose a driver is interested to traffic speed information on his way to home. So s/he invokes a call to the *lookup* primitive, where the *EVENT_SCHEMA* parameter equals "traffic speed" TRAFFIC_SPEED (FLOW, ROAD_NAME, TIME). Further suppose s/he receives the returned event schema with attributes SPEED (TRAFFIC_FLOW, LOCATION, TIMESTAMP) from the ESR. Assume that later a new schema VELOCITY (FLOW, ROADLINK_ID, TIME) is registered, and this schema also matches the driver's interest. However, the driver has no way of knowing about the new registration, unless s/he invokes the lookup primitive with the same parameter again later. Clearly, some kind of "update" service is desirable from the users' perspectives and that is why the parameter RATE is introduced. Here is how it works. For each call C with the *lookup* primitive, the IIP Client retransmits C to the ESR at the frequency indicated by the *RATE* parameter given in C. Thus, the user receives updated information on relevant schemas periodically. The retransmission occurs until *EXPIRATION_TIME*. If *RATE* and *EXPIRATION_TIME* are omitted, the lookup occurs only once.

4. THE Event Broker

The Event Broker serves as the message "bridge" connecting publishers and subscribers. It provides the following primitives:

- *publish(EVENT, EVENT_SCHEMA, [PUBMODE], [AREA],[LIFETIME])*
- *subscribe(PREDICATE, EVENT_SCHEMA, [PUBMODE], [AREA], [LIFETIME])*
- *unsubscribe(PREDICATE, EVENT_SCHEMA)*
- *announce(EVENT, EVENT_SCHEMA, [AREA], [PUBMODE],[LIFETIME])*

Next we first give precise definitions for all primitive parameters and explain the semantics of each primitive, and then we discuss how the Event Broker is implemented.

4.1 The *publish* Primitive

Each node invokes the *publish* primitive when it needs to publish an event. The EVENT parameter represents an instance record of the event schema given by the EVENT_SCHEMA parameter.

4.1.1 The PUBMODE Parameter

Given a publish call from an application, the IIP Client Component may need a mechanism to intelligently decide which action to take, i.e., to deliver the publication to the IIP Cloud Component via the Cloud Interface or to disseminate the publication using the Mobile Peer to Peer Interface, or both. Thus the *PUBMODE* parameter is introduced as the basis on which the IIP Client Component makes such a decision. In case the *PUBMODE* parameter is absent, the IIP Client Component simply disseminates via both interfaces.

Specifically, the value of *PUBMODE* can be (i) *mobile*, meaning publishing events through the embedded Mobile Peer to Peer Interface, i.e. disseminating events among peers; (ii) *cloud*,

meaning publishing events to the IIP Cloud Component via the embedded Cloud Interface, i.e. transmitting events to the Event Broker; or (iii) *mixed*, meaning publishing events via both mobile and cloud interfaces. For an example of publishers using different values for *PUBMODE*, consider the application described in [25] where private cars disseminate events reporting the witnessing of a fleeing car driven by some criminals. Patrolling police cars receive such events. In this case, the private cars are using *mobile* mode to publish events via the free unlicensed spectrum. And the police cars may use *cloud* mode to report the collected events to the headquarters where a chase plan is made.

Notice that the value of the *PUBMODE* parameter makes no indication about the mobility status of the publisher. For example, a publisher who uses the *mobile P2P* mode may be a moving probe car or a fixed roadside sensor. On the other hand, some publishing modes may not make sense provided the mobility status of the publisher. For instance, it is meaningless for a desktop user sitting in the office to publish using the *mobile P2P* mode.

4.1.2 The AREA and LIFETIME Parameters

The *AREA* parameter specifies the region where the intended recipients of the published events are located. By intended recipients, we refer to the anticipated receivers of the events from the publisher's point of view. For example, consider the Emergency Electronic Brake Light (EEBL) application, where a car disseminates an event to upstream vehicles when it detects that the driver suddenly brakes hardly. In this case, the value of the *AREA* parameter is the rectangle between the current location L of the car, and a certain point on the same lane behind L.

Notice that we expect that most of the time the *AREA* parameter is used by the Mobile Peer to Peer Interface, i.e. when the *PUBMODE* parameter has a value of *mobile* or *mixed*. In other words, any implementation of the Mobile Peer to Peer Interface is supposed to support geocasting, i.e. delivering events to a group of nodes within a certain geographical area. However, the AREA parameter may also be used when *PUBMODE* equals *cloud*.

Another observation is that nodes outside of the *AREA* may also receive the events, but they are only brokers who relay the events, but not display them to their users. For example, in the EEBL application, vehicles traveling in the opposite direction may relay the emergency-brake event, but they will not display this event to the driver.

The IIP Client Component will provide a Graphical User Interface (GUI) for the purpose of specifying *AREA*. The GUI will display the real time road network around the node, which typically consists of nearby parallel lanes in both directions. And it provides shapes, such as circle, rectangle etc., that are used to select areas on the displayed road network. For an example the "here I am" application will use a circle centered at the vehicle as the AREA.

The *LIFETIME* parameter indicates how long the event is to be valid in the network. An invalid event will be discarded by all its carriers.

PUBMODE, *AREA* and *LIFETIME* are all optional parameters to the *publish* primitive because they are simply "performance hints". In other words, they affect performance, but not the semantics of the application. For example, if the AREA is omitted from the EEBL publication, then event will travel far unnecessarily. But the subscription can specify that semantically, it is only interested in EEBL events that are generated up to 2 miles ahead. Similarly, if *PUBMODE* is omitted, the event will be published via both interfaces, although it will have no subscribers on one of them.

4.2 The *subscribe/unsubscribe* Primitive

Each node invokes the *subscribe* primitive to receive events of interest. The *AREA* parameter and the *LIFETIME* parameter have the similar semantics as their counterparts in the *publish* primitive. However, here they affect the semantics of the subscribe primitive. For example, in the aforementioned EEBL scenario, a subscriber cannot express that s/he is only interested in hard brake events published in last 30 seconds by vehicles within 1 mile ahead, without the *AREA* and the *LIFETIME* parameter.

At any time, a subscriber can call the *unsubscribe* primitive to revoke a prior subscription.

4.2.1 The PREDICATE Parameter

The *PREDICATE* parameter is a Boolean combination of conditions on the values of the requested event attributes. For example, PRECIPITATION > 10mm indicates interest in WEATHER events in which the PRECIPITATION attribute has a higher value than 10mm.

A PREDICATE is either (i) an *Event Filter Predicate* or (ii) an *Event Pattern Predicate*.

An Event Filter Predicate is a conjunction of multiple predicates over a single event. Each predicate is a binary function, returning either TRUE or FALSE. An Event Filter Predicate is matched against individual events, and a match is found if an event makes the evaluation of the predicate TRUE. For example, consider a driver that subscribes to weather forecast events which are published every 15 minutes. The attributes of the weather events are (TIME_INTERVAL, TEMPERATURE, WINDSTRENGTH, PRECIPITATION, ULTRAVIOLET_INDEX). An example of an Event Filter Predicate is "notify me if rain with precipitation of over 100 millimeters is coming within an hour". The predicate will be evaluated on each individual weather event to filter out all events whose TIME_INTERVAL and PRECIPITATION values do not satisfy the above conditions.

In contrast to an Event Filter Predicate, an Event Pattern Predicate is often defined and evaluated across multiple events of the same event schema. For instance, again consider a driver that subscribes to the aforementioned weather forecast events the predicate: "Notify me if the temperature drops 10 degrees within 30 minutes". This is an example of an Event Pattern Predicate since it can only be evaluated on a sequence of events. Another example of an Event Pattern Predicate in the above context could be "notify me when the temperature reported by the weather forecast events has monotonically increased for two hours".

We utilize the language described in [3] to formally define the Event Pattern Predicates. The language essentially consists of a set of formally defined language operators, such as projection, union, conditional sequence, iteration, etc. Given the defined operators and expressions, matching an Event Pattern Predicate

against an event stream can be done using extended finite state automata. Another possible language for defining Event Pattern Predicates is FTL [28].

4.3 The *announce* Primitive

The *publish* and *(un)subscribe* primitives provide the basic semantics of a pub/sub system. More primitives are required to construct various applications. An additional primitive we propose is *announce*. All parameters in the *announce* primitive have the same meaning as their counterparts in the *publish* primitive. The difference between the *announce* and the *publish* primitives lies in that announced events will be delivered to all subscribers located in the *AREA* regardless of whether or not they have subscriptions that match the events.

The *announce* primitive is used by emergency notification applications, such as delivering warning, evacuation, or rescue notification events to drivers in some affected urban area, or to drivers of specific types of vehicles (e.g. ambulances).

4.4 Implementation of the Event Broker

Recall that we define three different modes, namely *mobile*, *cloud* and *mixed*, for the *PUBMODE* parameter. This means that for each event schema, there are three modes to publish the events of a schema, and three modes to subscribe to events of the schema. Therefore, given an event schema, there are 9 different scenarios in terms of how the publications and subscriptions are transmitted. For each of these 9 scenarios, the matching between events and subscriptions can be implemented in IIP by two types of Event Brokers. One type that matches mobile P2P subscriptions with mobile P2P publications, and another type that matches Cloud subscriptions with Cloud publications.

The most straightforward way to implement the Cloud Event Broker uses the centralized approach, like the Event Schema Registry does. However, publication and subscription rates are usually much higher than event schema registration rate. Thus the single server implementation is probably not reliable or feasible. A more promising approach is to implement the Event Broker as a set of autonomous, interconnected distributed servers. The servers form a flat overlay network and they forward events and subscriptions among themselves and ensure that each event is matched against each active subscription in the overlay network. A possible forwarding algorithm is described in [9].

There are many alternatives to implement the mobile P2P Event Broker. For example, each node stores its subscriptions locally, and only publications are disseminated in a P2P fashion. A match occurs when the publication reaches the local subscription. Even for this simple strategy, many rules, such as which peers can be brokers of which events, and when the brokers retransmit received events, needs to be specified in order to optimize the system performance and efficacy. Observe that the mobile P2P Event Broker is a best-effort type of algorithm which is not guaranteed to discover all the existing matches. This limitation is inherent in the mobile P2P (i.e. VANET) dissemination. The fact that safety applications often employ this mode makes the problem acute. However, solving this problem is beyond the scope of this paper.

5. RELATED WORK

The present proposal of the Event Schema Registry is related to work on service discovery. Currently existing service discovery protocols generally fall into two categories in terms of the way in which service information is managed. One approach is to broadcast service information through the entire network either using a gossip algorithm or maintaining a multicast tree. Salutation [13] by IBM and Universal Plug and Play (UPnP) [14] by Microsoft are representative members of this category. We believe such an approach suffers from a high cost for broadcasting or tree construction and maintenance in highly mobile networks, and thus is not suitable for IIP. An alternative approach taken by other protocols such as Universal Description and Discovery and Integration (UDDI) [21] for web services and Jini [22] is to maintain a central directory to store service information. The Event Schema Registry component of IIP also takes a central-directory approach, and can indeed use a web service system for implementation.

The paper is also relevant to the pub/sub literature. There are basically two types of publish/subscribe systems, namely topic based and content based [2]. In topic based systems, users express interests by simply joining a group defined by a central subject. Whereas content based systems provide much more flexibility in expressing users' interests by allowing users to specify predicates over a set of attributes. As a result, arbitrary queries over the events content can be easily posed by users.

The pub/sub system in IIP employs a content-based approach. In contrast to traditional pub/sub systems [16, 10, 11] which are cloud-based, IIP supports an arbitrary combination of mobile peer-to-peer and cloud publications and subscriptions. In terms of purely mobile P2P implementations, a few pub/sub systems have been proposed. For example, [19] proposes a publish/subscribe implementation for MANETs. In the implementation, subscriptions are only deployed locally due to the fact that the paths utilized by the subscription forwarding strategy in server overlay networks quickly become stale in a mobile environment. When receiving an event from a neighbor, a mobile node matches the event to its own subscriptions and broadcasts the event, as long as it is still valid given the current location and time.

Triantafillou and Aekaterinidis [20] present a different approach to support P2P applications via building a pub/sub middleware over a structured P2P network such as Chord [23]. But that solution is again restricted to a static P2P network.

In summary, these existing systems are relevant and their concepts can be used in the distributed implementation of the IIP sub/sub system.

6. CONCLUSION AND FUTURE WORK

The development of ITS applications imposes the need for a platform which provides generic data management and communication functionalities. To meet the need, this paper proposed IIP as a way to facilitate the development of a variety of prospective ITS applications. We outlined the architecture of IIP and discussed the implementation of its two major components, namely, service discovery and publish/subscribe.

Several open issues require further exploration. In the paper, we do not propose any specific mobile peer to peer protocol for the

Communication Layer of the IIP Client Component. Although some existing research is concerned with this issue [12, 17], there still remain many interesting problems when attempting to apply those approaches to the ITS environment. For instance, in order to expedite the information dissemination in the mobile P2P network, in additional to short range communication methods, wide area wireless networks, e.g. the cellular network, can be utilized. More specifically, to improve performance both publications and subscriptions can be disseminated. When a subscription and publication are matched at a node, since the net id of the subscriber is available, the event can be directly delivered to the subscriber via the cellular network. Thus, a subscriber may receive the event either via the mobile P2P network, or via the cellular network, whichever comes first. A similar approach was taken in [27] for matching meta-data and queries in a vehicular network.

Also, how to make the pub/sub protocol adapt to various combinations of applications and network conditions remains an open question.

7. REFERENCES

[1] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A. The many faces of publish/subscribe. *ACM Comput. Surv.2003*.

[2] Huang, Y. and Garcia-Molina, H. Publish/subscribe in a mobile environment. *Wirel. Netw.* 10, 643-652, 2004.

[3] Demers, A., Gehrke, J., Hong, M., et al. Towards Expressive Publish/Subscribe Systems. In EDBT'06.

[4] Lee Tsu-Tian. Research on intelligent transportation systems in Taiwan. *CCC '08.*

[5] The eSafety Initiative. http://ec.europa.eu/information_society/activities/esafety/index_en.htm.

[6] The integrated Project PReVENT. http://www.prevent-ip.org/en/home.htm.

[7] Vehicle Information and Communication System. http://www.vics.or.jp/english/vics/index.html.

[8] IntelliDrive – Safer, Smarter, Greener. http://www.intellidriveusa.org/.

[9] Carzaniga, A., Rosenblum, D. S., and Wolf, A. L. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.* 2001.

[10] Zhao, Y., Sturman, D., and Bhola, S. Subscription propagation in highly-available publish/subscribe middleware. *Middleware Conference'04*

[11] Sivaharan, T.,Blair, G., and Coulson, G. GREEN: A Configurable and Re-configurable Publish-Subscribe *CoopIS/DOA/ODBASE' 04.*

[12] Rezende, C. G., Rocha, S. and Loureiro, F. Publish/subscribe architecture for mobile ad hoc networks. *SAC '08.*

[13] Salutation Architecture Specification, Version 2.0c, Salutation Consortium, June, 1999.

[14] UPnP Device Architecture 1.0, UpnP Forum, Dec. 2003.

[15] Giunchiglia, F., Yatskevich, M., and Shvaiko, P. Semantic matching: algorithms and implementation. In *Journal on Data Semantic*s'07.

[16] Fiege, L., Mühl, G., and Gärtner, F. C. A modular approach to build structured event-based systems. *SAC '02.*

[17] Huang, Y. & Garcia-Molina, H. Publish/Subscribe Tree Construction in Wireless Ad-Hoc Networks. *MDM '03*.

[18] Xu, B., Ouksel, A., and Wolfson, O. Opportunistic resource exchange in inter-vehicle ad-hoc networks. *MDM'04.*

[19] Meier, R., Cahill, V. STEAM: Event-Based Middleware for Wireless Ad Hoc Networks. *ICDCS'02*.

[20] Triantafillou, P. and Aekaterinidis, I. Content-based Publish/ Subscribe over Structured P2P Networks. *DEBS'03.*

[21] UDDI Version 2.04 API Specification, OASIS standard, July 2002.

[22] Jini Technology Core Platform Specification, v. 2.0. Sun Microsystems, June, 2003.

[23] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM'01*.

[24] Eriksson, J., Balakrishnan, H., and Madden, S. 2008. Cabernet: vehicular content delivery using WiFi. In Proceedings of *MobiCom '08*.

[25] Lee, U., Zhou, B., Gerla, M. et al. Mobeyes: smart mobs for urban monitoring with a vehicular sensor network. *Wireless Communications, IEEE* , vol.13, no.5, pp.52-57, 2006.

[26] Euzenat, J., Loup, D., Touzani, M., and Valtchev, P. Ontology Alignment with OLA. *EON*'04.

[27] Wolfson, O. and Xu, B. "Multimedia Traffic Information in Vehicular Networks". ACMGIS' 09.

[28] Sistla, A. P. and Wolfson, O. Temporal Triggers in Active Databases. IEEE Trans. on Knowl. and Data Eng. 1995.