Reasoning about Time, Location, and Identity in Distributed Pervasive Computing

Sameer Adhikari and Umakishore Ramachandran
College Of Computing, Georgia Institute of Technology
801 Atlantic Drive, NW, Atlanta, GA 30332-0280, USA
sameera, rama@cc.gatech.edu

Abstract

To become reality pervasive computing, among other things, needs new kinds of middleware. In this paper we present an important middleware that helps pervasive computing applications reason about events using time, location, and identity information. The middleware consists of reasoning operations, event store, participation protocol, and communication model. The reasoning operations provide the means to navigate the three-dimensional event space with time, location, and identity axes. The event store acts as a repository of information that applications reason about. The participation protocol allows application components to dynamically interact with one another. The communication model allows the components to exchange information. We also evaluate our middleware by showing how applications can be developed using the system, by measuring the performance of the reasoning operations, and by studying the performance of an application pipeline.

I. Introduction

A key enabler of pervasive computing is the falling prices of computing devices and networking infrastructure. Still, many milestones need to be reached before pervasive computing becomes an integral part of our lives. An important piece is the middleware that allows developers to easily create interesting pervasive computing applications. Consider the following scenario.

A video camera used for boundary surveillance in a high-security compound captures a car parked nearby on successive days. A program determines the duration of the car's presence at the location to be a week long. The application then finds out that the car has been at various high-security locations over the past month. There are many things the application does next in parallel. One, it notifies guards, who have wireless handheld devices, to check the car. Two, it informs the applications in the other high-security compounds in the region to look out for the car. Three, it checks if this car was observed during some recent incident in the city. If it was, a police application then checks if there is anything suspicious about the car (reported stolen), or owner (criminal record).

Today this scenario is imaginary, as other possible applications in the domains of surveillance and traffic management. These applications, though seemingly different, have some common features. Components of the application are physically distributed, with all the attendant needs of distributed programming. Applications are distributed in time; behavior is influenced by live data as well as historical data. The components are spatially dynamic in that the participating entities may change their behavior based on location. The components are temporally dynamic in that

the participating entities change constantly over time or an entity may participate in discrete intervals rather than a continuous interval. The application components that are spread over time and location may have widely heterogeneous computation and communication capabilities. Fundamentally, these applications need to reason about events with respect to time, location, and identity in an integrated manner to control application behavior. Time, location, and identity refer, respectively, to the when, where, and who of events that drive the behavior of the application. Currently there is no middleware that allows pervasive applications to reason about events using time, location, and identity.

In this paper, we present the following. The need for applications to reason about time, location, and identity using some applications from surveillance and traffic management (Section II), and a summary of their characteristics (Section III). The system infrastructure to support the ability to reason about time, space, and identity by describing its architecture (Section IV), and implementation (Section V). The system has the following components: reasoning operations, event store, participation protocol, and communication model. The reasoning operations provide a rich set of APIs for navigating the three-dimensional space of space, time, and identity. The event store acts as a repository of the information that an application reasons about. The participation protocol allows the application components to initiate and maintain distributed and dynamic interactions with one another. The communication model allows the application components to exchange information necessary for the interaction. We evaluate the system along several dimensions in Section VI. First, we qualitatively show the ease of programming complex pervasive applications using our system. Second, we quantitatively evaluate the cost of the reasoning operations using a set micro-benchmarks. Third, we model an application (such as surveillance) using our system, and quantify the application level performance of our system. We present related work in Section VII and conclude in Section VIII.

II. Application Domain

In this section we talk about surveillance and traffic management applications. The present current work in these domains motivates the need for a middleware that helps applications to reason about time, location, and identity. The current work is limited in the sense that each project focuses on a specific reasoning instance. A middleware will be usable across a broad spectrum of applications, and help build comprehensive applications.

A. Surveillance

Various projects have looked at the automating different aspects of surveillance. In MPI-Video ([1]) the system allows a user to get data from disparate sources and interact with the data. An environment model that represents the dynamic state of the environment, helps an application developer to structure algorithms that determine the state of the environment. Davis et al ([2]) present work on surveillance of human activity: detecting independently moving objects from a moving ground camera, system to track multiple moving people using sequences taken from a stationary camera, classification of recovered motion into various activity classes. CyberARIES ([3]) is an agent-based system for collaborative, distributed surveillance system. Independent agents are in charge of individual sensor systems, and the agent collaborate with each other to reach a common decision about the environment. Ivanov et al ([4]) present a system for surveillance of human-car interactions. A tracker tracks objects and the tracking sequences are probabilistically classified according to a pre-determined set of discrete events. Multiple streams are correlated to reduce noise and uncertainty, and obtain a coherent interpretation.

The VSAM project ([5]) presents a multi-camera surveillance system. It detects people and vehicles, tracks them using cooperative sensors, determines their three-dimensional locations using a geospatial model, and presents the information to an operator using a graphical interface. Ellis and Black ([6]) present a system that uses multiple cameras to track objects. The track data is sent to a server that aligns temporally aligns the streams to compensate for the different processing rates, and integrates data from multiple streams to attain a common view. In BioSTORM ([7]) a system to quickly detect an epidemic resulting from a bio-terrorism agent is presented. The authors have developed a knowledge-based epidemic surveillance system that uses disparate clinical data sources.

All these projects reason about time, location, and identity under specific constraints. They are a long way off from being a part of end-to-end applications used for practical purposes such as homeland security ([8], [9]).

B. Traffic Management

The typical infrastructure for traffic control in a metro area can be described as follows. There are detectors on major roads that record various measures such as speed (of individual vehicles) and flow (number of vehicles in a unit time). The sensors are placed at fixed intervals on the roadways, and data arrive periodically at the control center. The control center also receives the current state of control devices such as signals at intersections, and message signs. In the control center, operators interpret the information to determine the traffic state. They try to detect problems, figure out their possible causes, and take action to solve or reduce the severity of the problems (congestion, accidents, stalled vehicles). Some actions they can take are change the timing of the signals, or change the signs to inform the people in the vehicles. The current systems with their dependence on human operators, inherently limits the number of roads under observation.

Some projects have tried to automate various aspects of traffic management to assist the operators. In the Traffic Sign Classifier ([10]) an in-car system to recognize traffic signs was developed. In ESCORT ([11]) a platform to integrate devices that provide traffic control at an intersection was developed. In Traffic Scene Analysis ([12]) the system detects stalls in video images of the traffic on a highway. Naumann and Rasche ([13]) present an approach to prevent collisions at an intersection by using a token-ring-based communication protocol among autonomous vehicles. Monitorix ([14]) presents a traffic surveillance system that integrates video analysis and agent technology to detect abnormal events in traffic. Molina et al ([15]) present a decision-support system for traffic control. It helps operators in detecting traffic problems and in choosing appropriate control actions.

As in surveillance there is no complete system that integrates these various technologies. A programming system that supports reasoning about time, location, and identity will assist in developing complete systems.

III. Characteristics

We can see some common characteristics of the application domains described in Section II. These characteristics specify the requirements that a programming system for these applications needs to meet.

Time-Dependent These applications reason about about events based on time. We may want to find out if someone was in the building in the past hour. Further, different applications will have different representations of time.

Location-Dependent These applications reason about about events based on location. We may want to know whether most of the accidents in an area happened during late afternoon. Location representations can vary based on the source system (GPS [16], Active Badge [17], Active Bat [18], Radar [19], Cricket [20], and Trip [21]).

Location dependence also demands interaction support that has two parts. One, an incoming participant should be able to determine the other participating entities in an application. Two, the participating entities should be able to communicate with each other.

Identity-Dependent User identity will also influence application behavior, and the notion of identity will be application-specific. For example, access to different office areas will depend on the authorization available to an employee.

The dependence of an application of time, location, and identity information drives the application behavior. The three aspects are not independent of each other; reasoning about one of them invariably involves reasoning about the others.

Dynamic Dynamism in the pervasive applications arises both from individual application entity behavior, and from variation among entities. The programming system should support this inherent dynamic behavior of participating entities in a natural manner.

Heterogeneous The components of these application are heterogeneous. For example in a surveillance application the sensors can be cameras, baggage scanners, and others. The computer systems running this system can range from desktops handling sensors to a cluster analyzing data.

Some of the requirements these application characteristics (viz. time-dependent, location-dependent, dynamic, fault-prone, heterogeneous) impose on a middleware are the following:

- Reasoning system that allows the applications to reason about events based on time, location, and identity, and allows the application behavior to adapt to events in the environment.
- Participation protocol to create a dynamic interaction among the application entities by allowing them to arbitrarily join and leave an application.
- Communication mechanism for the entities to communicate with each other over time and location.
- Heterogeneity support to allow various devices to participate in an application.

There are many other requirements of pervasive applications impose that we are not looking at currently. Some of them are mobility, disconnected-operation, naming, discovery, privacy and security

IV. Architecture

In this section we present the CReST system that allows an application to reason about time, space, and identity of events. The CReST system assumes a model of the universe shown in Figure 1. There is a set of participation servers and event stores, supporting a set of applications. Each application consists of a set of software entities working together. For example, in Figure 1 entities 1, 3, 4, 8, 9, and 10 participate in application 251. An entity can participate in more than one application (e.g. in Figure 1 entity 4 is a participant in three applications). The participation server helps a set of application entities to discover each other. Application entities can generate events, store them an the event store, and correlate events. The entities can communicate with each other directly, or indirectly through the event

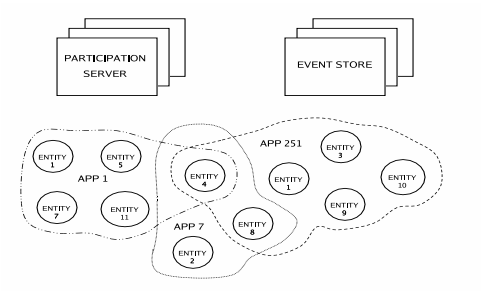

Fig. 1. Universe Model in CReST

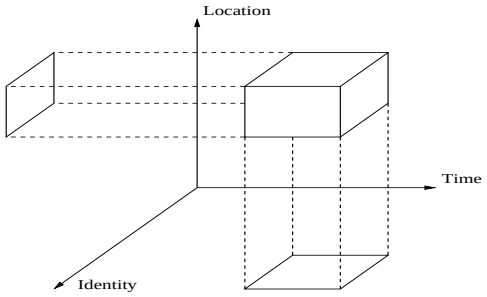

Fig. 2. Reasoning Visualization

store. The architecture has the following components: events, reasoning operations, event store, participation protocol, and communication model.

A. Events

An event is a 3-tuple and consists of time, location, and identity information. Time, location, and identity refer to the following: when did it take place, where did it take place, and who was involved in it. The time can be real (clock and calendar time) or virtual (some integer value). The location can be in different forms; GPS coordinates, application-specific notation. An identity can represent anything: real world object (person, place, thing, etc.), a virtual identity (e.g. a module that fuses data from multiple sensors), data used in an application.

An application can reason about multiple types of events. For example, if an application has two time types (TIME1, TIME2), one location type (LOCATION), and two identity types (IDENTITY1, IDENTITY2). It can can have multiple event types such as: EVENT1 = (TIME1, LOCATION, IDENTITY1), EVENT2 = (TIME1, LOCATION, IDENTITY2), and EVENT4 = (TIME2, LOCATION, IDENTITY2). Also, multiple applications can reason about the same event type.

B. Reasoning Operations

For reasoning about events using time, location, and identity information, the system provides a set of correlation operations. Figure 2 pictorially presents the correlation operations. An event definition in an application can be thought of as a three-dimensional space. (Each event instance is a point in the space.) Each of the event attributes (time, location, and identity) is an axis in the space. Each axis is only conceptually a line. Two factors prevent if from being a real line: one, a point on the axis can have multiple attributes; two, each axis is a collection of discrete rather than continuous values.

A correlate operation takes values in two event dimensions as arguments, and returns values in the third event dimension. The correlate operation works as follows:

- Determine the planar area defined by the values in the two input dimensions.
- Determine the cluster of points in the three-dimensional space that projects on this planar area.
- If there is a projection, return values in the third dimension from the cluster as a result of the operation.

An application can use the system-provided definitions for the correlation operations, or it can extend the operations. As an example consider time T, location L, and identity O. Correlate operations are of the form L [] = Correlate (T, O) and T [] = Correlate (O, L []).

C. Participation Protocol

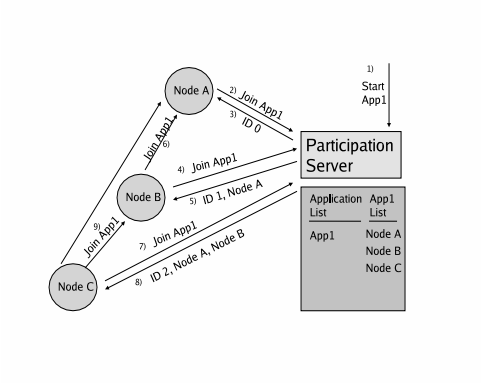

Fig. 3. Participation Protocol: Join

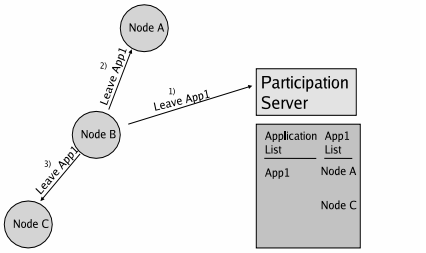

Fig. 4. Participation Protocol: Leave

In our participation protocol an entity explicitly joins or leaves an application. Initially an entity starts an application. It registers the application name with a participation server. Later on, when another entity wants to join the application, it first contacts the participation server. The participation server assigns an identifier to this entity and informs it of all the other entities in the application. Using this identifier the entity may contact the existing application entities, and informs them that it is joining the application. Figure 3 shows active join for three entities with the message sequences.

When an entity wants to leave the application, it informs the participation server and the other entities of its intent. Figure 4 shows a active leave for an entity among three participating entities. If a leaving entity did not inform another entity when it joined, it does not have to when it leaves. A entity can join multiple applications in parallel as there is no restriction. It can even join an application multiple times – if it wants to have multiple identities. The entities can join or leave an application in any sequence. This join and leave protocol helps create and maintain dynamic interaction among entities.

D. Event Store

The event store is a database that stores events for application to reason about later. When an application starts, it is assigned an event store by the participation-protocol server. This assignment holds for the life of the application. This global event store is in contrast to application entities storing events in a locally distributed manner. We chose the global approach because entities can join and leave arbitrarily, whereas the events have to be available all the time.

When an application entity stores an event it has to provide a unique identifier for each of the time, location, and identity instance for the event. The unique identifier has two parts: identifier assigned by the participation protocol to the entity (entity id) when it joins an application, and a locally unique integer generated by the entity itself (local id). This unique identifier allows an entity to store an event without having to synchronize with other entities in the application.

E. Communication Model

The system provide asynchronous communication between entities. To begin communication a receiver must notify a sender of its interest in receiving data and install a handler to receive data messages. Whenever a data message

arrives at the receiver, the sender-specific handler in invoked by the runtime. A receiver can install multiple handlers but cannot specify more than one handler for a specific sender. A sender starts sending messages after it receives the interest notification. Communication ends when the receiver notifies the sender that it is no longer interested in receiving data.

V. Implementation

The system is implemented as a runtime library that consists of the following components: participation protocol, communication system, event store schema, table creation, event storage, and reasoning operations. (Due to space constraints we do not describe the participation protocol and communication system.) The library is implemented in Java language because it is object-oriented and supports heterogeneity. The notion of an event as a tuple consisting of time, location, and identity components naturally maps to objects. We use PostgreSQL [22] database for our event stores.

A. Event Store Schema

The event store is a database that stores the event information. A global table that contains the event definitions is present in every event store. For example, Table I shows two event definitions, with EVENT class composed of TIME, LOCATION, and IDENTITY classes. Multiple event classes may have common component classes. For example, a class EVENT1 is composed of TIME LOCATION1, and IDENTITY. Each application-defined class (event, time, location, identity) maps to a table that stores the class instances. The table name is the same as the application-defined class name.

DEFINITIONS			
Event Name	Time Name	Location Name	Identity Name
EVENT	TIME	LOCATION	IDENTITY
EVENT1	TIME	LOCATION1	IDENTITY

TABLE I

Table to store event definitions

There is a two-level structure to the tables related to each event class. At the leaf nodes are three tables, one each, for the time, location, and identity class. Each table row represents a specific instance of the class in the table. At the root level is the event table. Each row consists of six columns, with two columns each, for the unique identifiers of the time, location, and identity values. The unique identifiers (entity id and local id) in the event table point to the entries in the leaf tables.

For example, consider an application where the time, location, and identity are represented by classes TIME, LOCATION, and IDENTITY respectively. A sample set of tables for these classes may look like as shown in Table III, Table IV, Table V, Table II.

Discussion In contrast to our approach of mapping objects to relations, object-oriented databases [23] provide an alternative to storing events. Some of the object-oriented databases are IRIS [24], ORION [25], GEMSTONE [26],

EVENT					
TIME		IDENTITY		LOCATION	
ENTITY ID	LOCAL ID	ENTITY ID	LOCAL ID	ENTITY ID	LOCAL ID
3	1	7	5	9	1
35	5	1	2	7	4

TABLE II

TABLE TO REPRESENT CLASS EVENT.

TIME		
ENTITY ID	LOCAL ID	Value
3	1	100
1	2	150
2	4	250

TABLE III

TABLE TO REPRESENT CLASS TIME.

OZ+ [27], ODYSSEY [28]. Table VI presents a comparison between object-oriented and relational databases. We chose the object to relation mapping because of the following reasons: higher performance, standard language, easy availability, and wide usage.

B. Table Creation

The system uses the Java interface mechanism for the application to specify the class attributes. For any class the application provides the names and types of the attributes. After the individual classes, the application informs the system about the composition of the event tuples. Based on all this information the system creates the tables in the

LOCATION					
ENTITY ID	LOCAL ID	Address	City	State	Country
10	2	320 X St.	Atlanta	GA	USA
5	5	120 Y Av.	Portland	OR	USA

TABLE IV

TABLE TO REPRESENT CLASS LOCATION.

IDENTITY						
ENTITY ID	LOCAL ID	First	Last	Eyes	Hair	Race
20	1	Sameer	Adhikari	Brown	Black	Asian
9	4	Arnab	Paul	Brown	Black	Asian

TABLE V

TABLE TO REPRESENT CLASS IDENTITY.

database. Any entity of the distributed application can call the create function. The application can ask the system to create the tables any time. There is only one condition: reasoning operations involving any class take place only after the corresponding table creation.

For example, in case of class TIME the application tells the system that it has an attribute Value whose type is Integer (Table III). It then tell the system that EVENT class consists of TIME, LOCATION and IDENTITY classes.

C. Event Storage

When an application invokes the store operation for an event the system does the following. It checks if the given time, location, and identity values already exist in the their respective tables. If they do not, it makes entries in the associated tables. If they already exist, it just extracts the unique identifiers. Next it checks the event table to see if the same entry already exists, using the unique identifiers it has from the individual tables. Only if this is a new instance, it makes an entry in the event table (pointing to the entries in the individual tables).

D. Reasoning Operations

Each reasoning operation maps to a multi-level query for the event database. Consider the case when an application correlates instances of TIME and LOCATION classes. The system queries the event-definition table to determine all the event classes that have the given TIME and LOCATION classes as components. By definition of an event class, the classes returned by the database will differ in the identity classes. For each event class the system does the following. First, it queries the event table using the unique identifiers (entity id and local id values) for the given TIME and LOCATION instances. Second, from the resulting events it extracts unique identifiers (entity id and local id values) to entries in IDENTITY. Third, using the unique identifiers values from the second step, the system queries the identity table to get the attribute value pairs. Fifth, it uses the attribute values to create instances of particular identity class. After, the system has processed all event classes, it returns the identity instances. Note that the set of returned identity instances can potentially belong to different classes.

VI. Evaluation

In this section we evaluate the CReST system in two ways. One, we qualitatively show the ease of programming using the system. Two, we quantitatively measure the system performance.

A. Programming using CReST

We revisit the scenario presented in the introduction (Section I). Consider an application that consist of the participating entities that represent the following: high security locations, guards. The entities use the participation protocol to join the application. The pseudocode for the application that implements the scenario may look as follows:

```
// Define variables.
CarIdentityClass Car; GuardIdentity Guards [];
DayClass CarTimes [], Week [7], Month [30];
```

OBJECT-ORIENTED DATABASES	RELATIONAL DATABASES
Persistent objects appear as normal objects.	Tuples represent the image of an object.
Persistent objects have complete object relationships.	Relationship between values in tables model object relationships.
Effi cient navigation between related objects via object references.	Navigation between related objects may require operations on multiple tables.
Objects of a class may scatter to different physical locations in a database, and slow down operations (query, update, etc.).	Objects of a class will be in one table, hence give higher performance for operations.
Data are tightly tied to the object design.	High level of data independence as tables are simple.
Can represent arbitrarily complex data.	Requires flattening of complex data into tables.
Extensible as new classes can inherit from exiting classes.	Harder to extend as there are limitations in inheriting tables.
No standard data manipulation language.	SQL is the standard data manipulation language.
Not easily available.	Easily available. Can even be freely downloaded.
Not widely used. Mostly a part of proprietary business applications.	Very widely used.

TABLE VI

TABLE COMPARING OBJECT-ORIENTED AND RELATIONAL DATABASES

SecurityLocationClass ThisSecurityLocation, OtherLocations [], HighSecurityLocations [];

// Initialize Variables.

Days = last seven days; Month = last 30 days;

HighSecurityLocations = high security locations in the surrounding region;

Guards = identities of guards in this high security location;

Car = car information from camera;

// Find out the duration of the car at current location.

CarTimes = Correlate (Car, ThisHighSecurityLocation);

if (CarTimes < Week) return;

// Check where has it been in the past one month.

OtherLocations = Correlate (Car, Month);

if (OtherLocations ≠ HighSecurityLocations) return;

// The car suspiciously has been to other high security locations in the past month.

Message = "Car" is suspicious;

// Send a message to some guards to check the car.

for (i = 0; i < Some number of Guards; i++) Send (Guard [i], Message);

// Send a message to other high security locations warning them about the suspicious behavior.

for (i = 0; i < Number of HighSecurityLocations; i++) Send (HighSecurityLocations [i], Message);

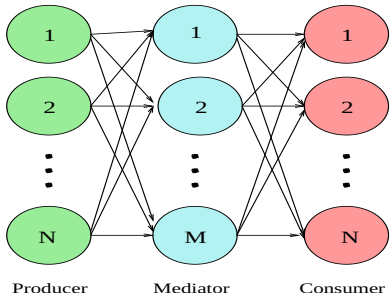

Fig. 5. Architecture of Survey Application

Next we describe the application (shown in Figure 5) that we have developed to measure the system performance. It consists of three classes of entities: producers, mediators, consumers. The communication topology is shown by the arrows connecting the various entities in Figure 5. All the producers, mediators, and consumers synchronize, using control messages, before the data transfer starts. In steady state, each producer sends data to all the mediators, the mediator creates a composite of the data it receives from all producers, and each consumer receives the composite from all the mediators. The number of producers, mediators, and consumers is variable.

Each producer entity does the following:

1) Join the application.
2) Synchronize with the mediators.
3) Sends the data it generates to all the mediators.
4) Leave the application.

Each mediator entity does the following:

1) For each producer, create a queue to store the incoming data.
2) Join the application.
3) Install a data-message handler to receive data that puts each incoming item in a producer-specific queue.
4) Synchronize with consumers and producers.
5) While there is data to be processed the mediator does the following.
    a) In the N^{th} cycle it picks up the N^{th} item from every queue and creates a composite.
    b) It performs a correlate operation and sends the composite to each consumer.
6) Sends a leave message to the participation server.

Each consumer entity does the following:

1) Join the application.
2) Installs a data-message handler to receive the incoming data from the mediators.
3) Synchronize with the mediators.
4) Receive all the incoming data.

5) Sends a leave message to the participation server.

The correlate operation just goes to the event store and returns without any match. The application pipeline we present is quite generic and applicable to various domains. The goal here is to study the generic application behavior on our system rather than focus on a specific application. For example, this prototype can easily be used for a surveillance scenario by inserting processing modules at the producer, mediator, and consumer. At the producer a tracker module can generate information to send to the mediator. At the mediator a module can use the correlate operations can do more meaningful analysis. At the consumer a module that displays video or plays audio can help a user. Further this application demonstrates that a complex application pipeline can be created with our system.

B. Performance

In this section we present the measurements of various CResT operations and an application pipeline that uses CReST operations. The experiments were run on nodes of a cluster consisting of 17 eight-way SMPs interconnected by Gigabit Ethernet. Each processor is a 550MHz Pentium III Xeon. Each node in the cluster has 4GB RAM and 18GB SCSI disk. The nodes run RedHat Linux 9.0.

Correlation Operations

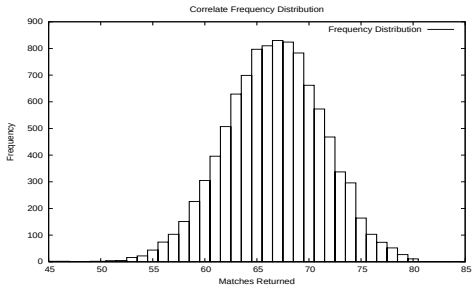

Fig. 6. Frequency distribution of correlate operations

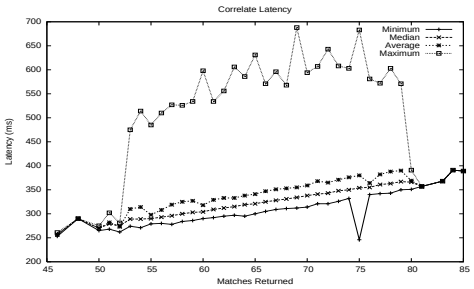

Fig. 7. Latency of correlate operations

To study correlation operations we have created an event store in the following manner. We have created the time, location and identity classes with each class having a single integer-value attribute. By varying the value of the integer-value attribute from one to 100 for each class, we are able to enumerate a million different events. For each event, with two-third probability we decided to store the event. Thus we end up with an event store of approximately 670,000 events. It takes more than seven hours to insert the events in the database.

The reason for choosing events in this manner is that we want a variable number of results when we execute the correlate operations. If we store all the million possible events, then every time we correlate an instance of one class (e.g. time) with an instance of another class (e.g. location), we would always get 100 instances of the third class (identity) as a result of the operation.

Next we perform the correlate operations. We perform 10,000 operations by enumerating all the possible combinations of 100 time instances and 100 location instances. Each correlate operation returns a certain number of identity instances. Figure 6 shows the frequency distribution of the number of return values for the correlation operations. Consider a

value Y on the y-axis and X on the x-axis. The pair (X, Y) means that Y out of the 10,000 correlate operations returns X identity instances. For example, 103 correlation operation returns 57 ind-entity instances. The correlation operations as a whole returns 46-85 ind-entity instances.

Figure 7 shows the latency of the correlate operations. The latency is measured as the time from when the correlate operation is issued by an application to the time when the system returns the results to the application. The x-axis refers to the number of identity instances returned by the correlate operation. As can be seen in Figure 6 multiple correlate operations may return the same number of results. Thus, the y-axis shows the minimum, median, average, and maximum time taken by correlate operations that return the same number of results. For example, when correlate operations returns 58 identity instances, the minimum time taken is 284 ms, the median is 300 ms, the average is 325 ms, and the maximum is 526 ms.

In terms of trend, all the lines show expected behavior. That is, the greater the number of returned results, the greater is the time taken for the operation. Except at the boundaries where there is only one correlate operation that returns a given number of results. (X = 46, 48, 81, 83, 85.) In these cases, the minimum, median, average, and maximum values are the same, and they distort the trends.

There are some other points to note about the readings. One, the time values suggest that the database is completely in-memory. The time taken to return the results for a correlate operation is very less compared to the typical disk-access latency. Two, the maximum time for a given number of results is about double the minimum, whereas the median and average values are close to the minimum. The reason for this wide variation may be that the measurements for the maximum time correspond to the time Java performs garbage collection.

Application Measurements

We use the application described in Section VI-A and shown in Figure 5 for this study. We vary the number of producers (consumers) from two to ten, and the number of mediators from one to five. (The number of consumers is always equal to the number of producers.) The number of producers (consumers) and mediators defines the instance of an application. The goal is to see how the correlate operations affect application behavior. The metric we measure is the bandwidth delivered at the consumers for each application instance. The bandwidth reading pertains to two scenarios: no correlate operation performed by the mediator, correlate operation performs by the mediator. In the correlation scenarios, each mediator performed a correlate operation before sending the composite to all the consumers. The correlate operation performed by the mediators can be thought of as a null operation. The inputs to the operation are chosen such that the operation just queries the database without returning any results. Further, the database is very small (very few events) to ensure that the query processing takes a very short time.

Figures 8, 9, 10, and 11 present the results of the experiments. Figure 8 shows the bandwidth as a function of the number of producers (consumers), when there are one and two mediators. The data are shown for both with and without correlate operations. For example, with seven producers (consumers) and one mediator, without correlation the bandwidth is 0.45 MBps, and with correlation it is 0.33 MBps. Figure 9 is the same as Figure 8, except that it deals with three and four mediators. For example, with six producers (consumers) and four mediators, without correlation the bandwidth is 1.37 MBps, and with correlation it is 0.59 MBps. Figure 10 presents the bandwidth as a function of the number of mediators, when there are five and six producers. The data are shown for both with and without correlate operations. For example, with three mediators and five producers (consumers), without correlation the bandwidth is

0.78 MBps, and with correlation it is 0.40 MBps. Figure 11 is same as Figure 10, except that it deals with seven and eight producers. For example, with five mediators and eight producers (consumers), without correlation the bandwidth is 1.50 MBps, and with correlation it is 0.72 MBps.

The obvious conclusion that can be drawn from these graphs is that adding a correlate operation in an application pipeline adds overhead and decreases bandwidth delivered at the consumers. In Figure 8 and Figure 9, as the number of producers (consumers) increases, so does the bandwidth when mediators perform correlation. The reason for this behavior is that a mediator performs one correlate operation before sending a composite to all the consumers. In essence, the cost of the correlate operation is amortized over more communication operations. The decreasing cost of correlation with increasing number of producers point to the convergence of the lines for data with and without correlation. In Figure 10 and Figure 11, as the number of mediators increases, so does the bandwidth when the mediator performs correlation. Even though, an increase in mediator numbers means more correlation, at the same time it also means greater parallelism in communication with consumers. Potentially when some mediators are performing correlations, others are sending data to the consumer. Further, the lines for bandwidth with correlation track the behavior the of corresponding line without correlation though with lesser bandwidth values.

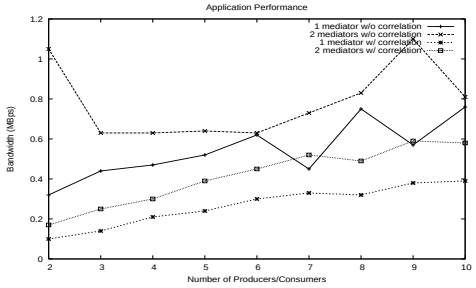

Fig. 8. Application bandwidth for 1 and 2 mediators

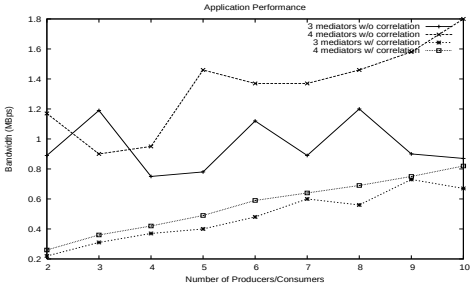

Fig. 9. Application bandwidth for 3 and 4 mediators

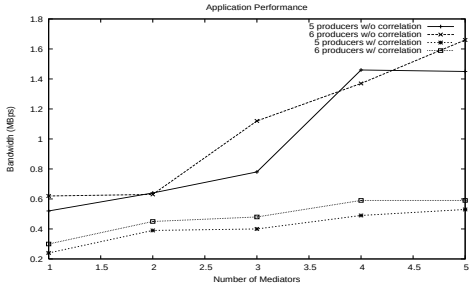

Fig. 10. Application bandwidth for 5 and 6 producers

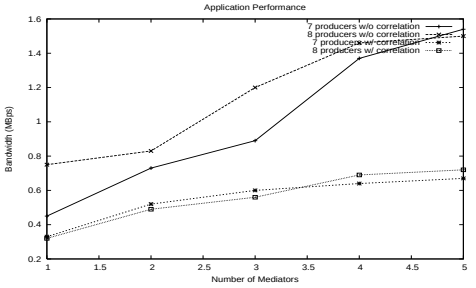

Fig. 11. Application bandwidth for 7 and 8 producers

VII. Related Work

In this section we discuss work that can potentially be used to build similar applications.

Pervasive Computing Middleware

The Context Toolkit ([29], [30]), is a system that provides a framework for building context aware applications. The framework provides the following support for context-aware applications: specification, handling, interpretation,

communication, storage, availability, and discovery. Context Toolkit aids applications by providing the means to separate the acquisition of context from its use. It focuses in the acquisition part but it leaves the use undefined. Our system in contrast does not handle the issue of acquisition, but it provides support to reason about the primary types of context information: time, location, and identity.

The relationship of our system to context toolkit can be thought of as a protocol stack, with the context toolkit the lower layer, and our system the higher layer. As an analogue consider networking applications. The TCP protocol is enough for distributed application components to communicate. Still, we have messaging protocols such as MPI and HTTP based on TCP. These protocols ease the task of communication for various applications. MPI helps parallel scientific applications, and HTTP helps Internet applications. Similarly our system helps pervasive computing applications to reason about time, location and identity.

The Active Spaces project ([31], [32]) presents the notion of an active space, an active space meta-operating system called Gaia OS, and an application framework to build active space-aware applications. Their focus is essentially in helping application adapt to the different environments when a user moves. They reason about time, location, and identity in a limited manner, but do not provide a system for general-purpose reasoning.

The OneWorld project ([33], [34]) also provides support to help pervasive applications adapt to dynamic environment. It provides services so that applications can be informed about the change in context, allow ad-hoc composition of services, and share information. How the application handles contextual change, composition, and sharing facilities is left to the applications. Our reasoning middleware can help the application in handling the facilities.

Apart from the system services, another difference between these projects (Context Toolkit, Active Spaces, One World) and our project is the target applications. They focus on personal applications (limited area, less users), whereas we focus on large scale distributed application (surveillance, traffic management).

Data Mining

Data mining ([35], [36], [37], [38]) is a process that allows users to discover patterns and relationships in data, using a variety of analyses, and to make predictions based on the discoveries. The basic steps of data mining for knowledge discovery are: define problem, build data mining database, explore data, prepare data for modeling, build model, evaluate model, and deploy model. Our work is similar to data mining in that both look to model data using some mathematical concepts. Our work differs from data mining in a couple of ways. First, we target pervasive computing applications, whereas data mining targets business transactions. Thus we try to find specific patterns among time, location and identities in an application, and data mining tries to find random patterns in data. Second, data mining is an offline process, while our system is an on-line system.

Programming Models

Linda [39], [40], [41], TSpaces [42], [43], and JavaSpaces [44] provide the tuple space model of building distributed systems. In the tuple-space model producers store tuples of attributes and values in a space. Consumers retrieve tuples they are interested in by providing a template to the space to filter the tuples. The tuple-space model of programming provides an annonymous and asynchronous communication between entities. Such systems provide fairly generic programming capabilities for data sharing and synchronization. In theory they can be used to build any pervasive computing application. In practice they lack specific help for the characteristics found in pervasive computing applications. For example they do not support an ability to reason about events using time, location, and identity.

VIII. Conclusion

In this paper we have presented an important middleware for pervasive computing applications. The middleware allows applications to reason about events using time, location, and identity information. We have evaluated the middleware by showing the ease of programming using the system, measuring the costs of the reasoning operations, and studying the performance of an application pipeline that uses our system.

There are may avenues for future work. One possible area is the correlation operations. Currently a correlate call returns all the results that match the input. We can try to return a subset of the complete results based on user criteria. We have to determine what criteria are important and how to specify them. For example, a criterion can be ten percent of the total events that are closest to the center of the cluster of matching events. Another possible area is application study. We can build an end-to-end application using our system, such as surveillance of an office building. Then study the behavior and performance of the application.

References

[1] J. Boyd, E. Hunter, P. Kelly, L. Tai, C. Phillips, and R. Jain, "Mpi-video: Infrastructure for dynamic environments," in *IEEE Conference on Multimedia Computing and Systems*, Austin, TX, June 1998.

[2] L. Davis, S. Fejes, D. Harwood, Y. Yacoob, I. Haratoglu, and M. J. Black, "Visual surveillance of human activity," in *Asian Conference Computer Vision*, Hong Kong, China, January 1998, pp. 267–274.

[3] C. Diehl, M. Saptharishi, J. H. II, and P. Khosla, "Collaborative surveillance using both fixed and mobile unattended ground sensors," in *SPIE's 44th Annual Meeting*, vol. 3713, July 1999, pp. 178–185.

[4] Y. Ivanov, C. Stauffer, A. Bobick, and W. Grimson, "Video surveillance of interactions," in *Conference on Computer Vision and Pattern Recognition and Workshop on Visual Surveillance*, Ft. Collins, CO, June 1999.

[5] R.Collins, A.Lipton, H.Fujiyoshi, and T.Kanade, "Algorithms for cooperative multi-sensor surveillance," *Proceedings of the IEEE*, vol. 89, no. 10, pp. 1456–1477, October 2001.

[6] T. Ellis and J. Black, "A multi-view surveillance system," in *IEE Intelligent Distributed Surveillance Systems*, London, UK, February 2003.

[7] D. Buckeridge, J. Graham, M. Connor, M. Choy, S. Tu, and M. Musen, "Knowledge-based bioterrorism surveillance," in *American Medical Informatics Symposium*, 2003, pp. 76–80.

[8] N. Smith and L. Messina, Eds., *Homeland Security: The Reference Shelf*. The H. W. Wilson Company, 2004, vol. 76, no. 1.

[9] J. Kayyem and R. Pangi, Eds., *First to Arrive: State and Local Responses to Terrorism: BCSIA Studies in International Security*. The MIT Press, 2003.

[10] L. Priese, J. Klieber, R. Lakmann, V. Rehrmann, and R. Schian, "New results on traffic sign recognition," in *IEEE Intelligent Vehicles Symposium*, Paris, October 1994, pp. 249–254.

[11] A. Savigni, F. Cunsolo, D. Micucci, and F. Tisato, "Escort: Towards integration in intersection control," in *Workshop on the International Foundation for Production Research on Management of Industrial Logistic Systems and 8th Meeting of the Euro Working Group Transportation - EWGT*, Rome, Italy, September 2000.

[12] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell, "Towards robust automatic traffic scene analysis in real-time," in *12th International Conference on Pattern Recognition*, Jerusalem, Israel, October 1994, pp. 126–131.

[13] R. Naumann and R. Rasche, "Intersection collison avoidance by means of decentralized security and communications management of autonomous vehicles," in *30th International Symposium on Automotive Technology and Automation*, Florence, Italy, June 1997.

[14] B. Abreu, L. Botelho, A. Cavallaro, D. Douxchamps, T. Ebrahimi, P. Figueiredo, B. Macq, B. Mory, L. Nunes, J. Orri, M. J. Trigueiros, and A. Violante, "Video-based mutli-agent traffic surveillance system," in *Intelligent Vehicles Conference and Dearborn and MI*, 2000.

[15] M. Molina, J. Hernandez, and J. Cuena, "A structure of problem-solving methods for real-time decision support in traffic control," *International Journal of Human-Computer Studies*, vol. 49, no. 4, October 1998.

[16] I. Getting, "The global positioning system," *IEEE Spectrum*, pp. 36–47, December 1993.

[17] R. Want, A. hopper, V. Falcao, and J. Gibbons, "The active badge location system," *ACM Transacctions on Information Systems*, vol. 10, no. 1, pp. 91–102, January 1992.

[18] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, "The anatomy of a context-aware application," in *Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking MOBICOM 1999*, August 1999, pp. 56–68.

[19] P. Bahl and V. Padmanabhan, "Radar: An in-building rf-based user location and tracking system," in *IEEE INFOCOM: The Conference in Computer Communications*, March 2000, pp. 775–784.

[20] N. Priyantha, A. Chakraborty, and H. Balakrishnan, "The cricket location support system," in *Sixth International Conference on Mobile Computing and Networking MOBICOMM 2000*, August 2000, pp. 32–43.

[21] D. de Ipina, P. Menonca, and A. Hopper, "Trip: a low-cost vision-based location system for ubiquitous computing," *Personal and Ubiquitous Computing Journal*, vol. 6, no. 3, May 2002.

[22] "Postgresql," http://www.postgresql.org.

[23] E. Bertino and L. Martino, *Object-Oriented Database Systems: Concepts and Architectures*. Addison-Wesley Publishing Company, 1993.

[24] D. H. Fishman, P. Lyngbaek, B. Mahbod, M. A. Neimat, T. Risch, M. C. Shan, W. K. Wilkinson, J. Annevelink, E.Chow, R. Conners, J. W. Davis, W. hasan, C. G. Hoch, W. Kent, and S. Leichner, *Object-Oriented Concepts and Databases and Applications*. ACM Press, 1989, ch. Overview of the Iris DBMS, pp. 219–250.

[25] W. Kim, N. Ballou, H. Chou, J. F. Garza, and D. Woelk, *Object-Oriented Concepts and Databases and Applications*. ACM Press, 1989, ch. Features of the ORION Object-Oriented Database System, pp. 251–283.

[26] R. Bretl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E. H. Williams, and M. Williams, *Object-Oriented Concepts and Databases and Applications*. ACM Press, 1989, ch. The GemStone Data Management System, pp. 283–308.

[27] S. P. Weiser and F. H. Lochovsky, *Object-Oriented Concepts and Databases and Applications*. ACM Press, 1989, ch. OZ+: An Object-Oriented Database System, pp. 309–340.

[28] J. Diederich and J. Milton, *Object-Oriented Concepts and Databases and Applications*. ACM Press, 1989, ch. Objects and Messages and Rules in Database Design, pp. 177–197.

[29] D. Salber, A. Dey, and G. Abowd, "The context toolkit: Aiding the development of context-enabled applications," in *Conference on Human Factors in Computer Systems*, 1999, pp. 434–441.

[30] A. Dey, D. Salber, and G. Abowd, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human Computer Interaction Journal*, vol. 16, no. 2-4, pp. 97–166, 2001.

[31] M. Roman, "An application framework for active space applications," Ph.D. dissertation, Computer Science, University of Illinois at Urban-Champaign, 2003.

[32] "Gaia: Active spaces for ubiquitous computing," http://choices.cs.uiuc.edu/gaia/.

[33] R. Grimm, "System support for pervasive applications," Ph.D. dissertation, Computer Science, University of Washington, December 2002.

[34] R. Grimm, J. Davis, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, , and D. Wetherall, "System support for pervasive applications," *ACM Transactions on Computer Systems*, 2004, (To appear).

[35] K. Thearling, "An introduction to data mining: Discovering hidden value in your data warehouse," http://www.thearling.com.

[36] ——, "An overview of data mining techniques," http://www.thearling.com.

[37] T. C. Corporation, "Introduction to data mining and knowledge discovery," http://www.twocrows.com.

[38] P. Adriaans and D. Zantinge, *Data Mining*. Addison Wesley Longman, 1996.

[39] D. Gelernter, "Generative communication in linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, 1985.

[40] S. Ahuja, N. Carriero, and D. Gelernter, "Linda and friends," *IEEE Computer*, vol. 19, no. 8, pp. 26–34, August 1986.

[41] N. Carriero and D. Gelernter, "Linda in context," *Communications of the ACM*, vol. 32, no. 4, 1989.

[42] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford, "Tspaces," *IBM Systems Journal*, August 1998.

[43] T. J. Lehman, A. Cozzi, Y. Xiong, J. Gottschalk, V. Vasudevan, S. Landis, P. Davis, B. Khavar, and P. Bowman, "Hitting the distributed computing sweet spot with tspaces," *Computer Networks*, vol. 32, pp. 457–472, 2001.

[44] E. Freeman, S. Hupfer, and K. Arnold, *JavaSpaces: Principles and Patterns and Practice*. Addison Wesley, August 1999.