Weave: An Architecture for Tailoring Urban Sensing Applications across Multiple Sensor Fabrics

Vinod Kulathumani, Mukundan Sridharan, Rajiv Ramnath, Anish Arora
Deptartment of Computer Science and Engineering, The Ohio State University
{vinodkri, sridhara, ramnath, anish}@cse.ohio-state.edu

***Abstract*—A characteristic of urban sensing is that applications need to compose the operations and fuse information from sensing fabrics extant in the environment, to meet requirements for which the fabrics were not designed a priori. We propose an architecture, WEAVE, that allows tailoring of applications across one or more urban sensing fabrics. Key to the architecture is use of standard application programming interfaces (including vertical or domain dependent ones). We propose such interfaces for sensing fabrics that support the class of search applications, and show how these interfaces are used to compose example applications in operation scenarios that we have implemented at the Ohio State University.**

I. Introduction

The domains of urban sensing applications are many and varied, ranging from security to entertainment. Unlike for example scientific applications where an "optimal" sensor network may be designed and then deployed, the urban environment consists of loosely-coupled (or even un-coupled) sensor fabrics that are already "out there". We characterize the urban sensing problem as one where, in response to a specific requirement (such as tracking and subsequent capture of an assailant), an application needs to be developed where (a) the operations of varied, independent sensor networks are integrated and (b) the information from these networks suitably retrieved, analyzed and fused to meet the application, either while an event is in progress or post facto after the event has taken place. Also, the exact sequence or pattern of events may not be known to completely automate a client application. In other words, often a human may (or even, must) exist in the loop to interpret outputs from a given fabric and involve followup fabric actions based on the outputs. [1]

In this paper, we discuss issues relevant to the architecture of urban sensing applications, and present key elements of an architecture, which we call *Weave*. In this architecture, we view individual sensor networks as programmable fabrics that can be leased by multiple high-level applications. To be used as such a fabric, these loosely coupled networks are trusted by their clients, conform to a shared information model, and provide services with standard interfaces.

This paper is organized as follows: In Section 2, we describe our concept of operations for urban sensing applications and use them to motivate architectural requirements. In Section 3, we outline the *Weave* architecture. We then, in Section 4, describe the standard application programming interfaces (API) for a class of search applications. Next, in Section 5, we present example search applications that are composed using the *Weave* APIs. The applications are chosen from operation scenarios that we have implemented at the Ohio State University (OSU): The first is related to an urban surveillance scenario where multiple sensor fabrics are used to detect and track suspicious persons entering a building. The second and third are location service applications for objects or people of interest within a building in the same fabric or across multiple sensing fabrics. In Section 6, we present related work, and finally, in Section 7, we discuss future work and make concluding remarks.

[1]This human intervention and inferencing requirement is often due to privacy regulations that require that automated inferencing, such as by using data-mining techniques, may not be made without legal approval of probable cause.

II. Example Application Scenarios

To motivate our characterization of urban sensing and identify requirements for our architecture, let us consider concepts of operations and specific application scenarios in two subdomains of urban sensing: campus surveillance and social networking.

Campus Surveillance ConOps: In collaboration with domain specialists at the Air Force Research Laboratory, the Homeland Security Program at OSU and the Ohio Association of Chiefs of Police, we identified the following representative scenarios for this domain.

Scenario: Suspicious persons need to be tracked across a campus, including through buildings. Monitoring outdoors is via a network of building-mounted video cameras and indoors is via motion sensor and camera network fabrics deployed inside buildings. To support low power operation and mitigate concerns about citizen privacy, the indoor network should not be activated for all moving persons, but be cued instead to operate when the outdoor video camera network tags some activity as suspicious and its tracking of people suggests that they are entering a building.

Social Networking ConOps: Motivated by scenarios where users have expressed interest in locating friends and finding out about objects of interest within a locality, we have implemented PeopleNet, a network of mobile stations, as well as multiple static sensor networks at OSU.

Each mobile unit in a PeopleNet fabric consists of a cell phone connected to a (Intel-developed, 802.15.4 radio equipped) psimote. Mobile units connect to a central access point in the building that they are in, as well as a set of infrastructure nodes that aid in localizing the mobile units. We are presently deploying an instance of PeopleNet within the CSE building and another instance is planned for the Recreation and Physical

Activity Center (RPAC) that is a few blocks away.

We have also designed and implemented 3 fabrics of static PIR sensors, (1) one of these is deployed within the CSE building to assist with indoor tracking and monitoring status of conference rooms, (2) a second one to be deployed in the RPAC to monitor availability of squash courts, and (3) a third one to be deployed at the nearby Oxley's Cafe that monitors the queue at the cafe. Apart from these PIR sensing fabrics, there is also an Elevator sensing fabric designed using low power radios that monitors the position and status of elevators inside the CSE building.

Fig. 1 shows each mobile unit of the PeopleNet fabric that consists of a cell phone integrated with a $802.15.4$ equipped psi-mote. Fig. 2 shows the deployment within the CSE building corresponding to the PeopleNet sensing fabric and the PIR sensing fabric. In the Peoplenet fabric there are two types of nodes, infrastructure nodes and mobile nodes. Each floor consists of a set of infrastructure nodes (shown by green dots) to assist with localization and programming of the mobile nodes. The mobile nodes are shown by blue dots.

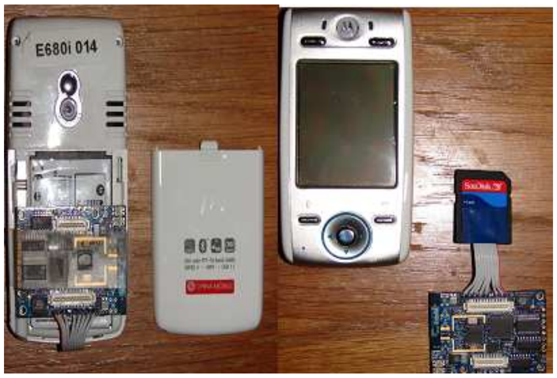

Fig. 1. A mobile unit in PeopleNet

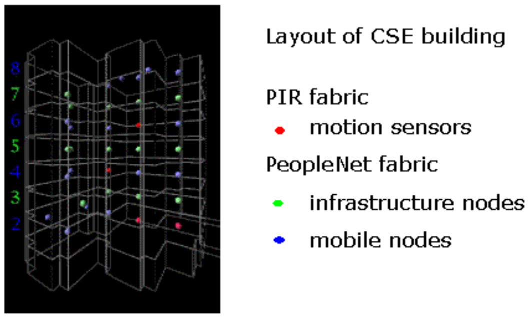

Fig. 2. Layout of CSE building with PIR fabric and PeopleNet fabric

Scenarios: Representative application scenarios in this case consist of users issuing ad-hoc locality specific queries such as "Is Mukundan already at the RPAC?", "Is any squash court in RPAC empty?", "Where is the elevator?", or "How long is the queue at Oxley?" via their PeopleNet mobile devices. Resolving these queries would typically require routing the query to the central point in the building where the user is and somehow accessing the desired data from the sensor fabric in question.

Common to these conops are the following architecture-relevant elements:

- ◇ *No "best" sensor*: Different sensor fabrics may suit for different environments (e.g., indoor and outdoor) even if the objects to be sensed are identical. The right detectors to deploy may not be known until a high-probability threat has been identified or an event occurs. Following the threat or attack and as part of the response, sensor fabrics may be re-targeted and re-purposed accordingly.
- ◇ *"Dual-use" fabrics*: Information and intelligence may be needed at multiple layers of the safety response system, i.e., to local, state and federal law and safety enforcement, each investigating situations with different objects and at different scales. In other words, fabrics should have the ability to support multiple clients operating independently.
- ◇ *Resource management*: The architecture must support the different fabrics to cope with their resource constraints, while still supporting their ability to federate.
- ◇ *Human-in-the-loop*: A primary mode of sensor fusion is human-driven, i.e., sensor streams are typically fed back to a command console and cross-stream inferences made by humans. Thus, support is desirable for letting authorized personnel readily access and search data sensed by the fabric, and conversely, for letting people add human "sensor" information such as "beat cops" reporting on suspicious activity, informants providing tips, etc. to databases that can be fused with data automatically collected via fabrics.

III. The *Weave* Architecture

Weave has four key architectural elements: Sensing fabric, Object and Client and an Access manager.

Sensing fabric: A fabric is an independent, decoupled, network, capable of sensing, storing, and communicating some physical phenomena. Examples include networks of motion sensors, surveillance cameras, netted microphones, and so on. A fabric need not in general share state or cooperate with others fabrics, as a result of policy or of platform ability.

A fabric offers its services to clients via an application programming interface (API). Some fabric services may be generic, others may be tailored to the application domain(s) that the fabrics support. We refer to the latter set as vertical services. Standardizing vertical services is desirable, so that applications can be readily composed and ported across fabrics geared to support a particular application domain. In general, a fabric need not make guarantees about its quality of service, delivering its results based only on a "best-effort".

Later, in Section 4, given the context of the "search" application scenarios described in Section 2, we detail a proposed standard API for the services offered by sensing fabrics that support search.

Object: A fabric enables measurement of physical phenomena to detect and classify physical objects such as humans, vehicles, weapons and explosives. Each type of object is characterized as a predicate on physical phenomena; all objects that satisfy the corresponding predicate belong to an object type. Objects may be identified (or associated) by the fabric or

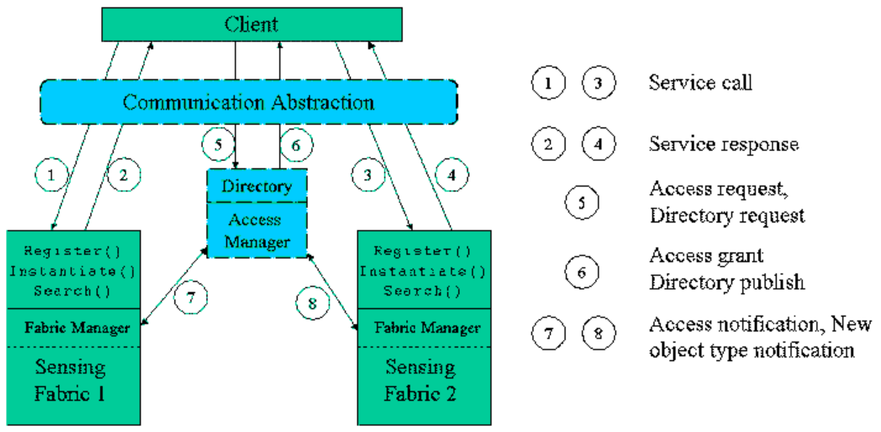
Client
Communication Abstraction
Directory
Access Manager
Register()
Instantiate()
Search()
Fabric Manager
Sensing Fabric 1
Sensing Fabric 2
1 3 Service call
2 4 Service response
5 Access request, Directory request
6 Access grant Directory publish
7 8 Access notification, New object type notification

Fig. 3. *Weave* Architecture

remain anonymous (or non-associated). Some objects may be known a priori to the fabric (for example, "blue force" objects that advertise their presence to the network or permanent elements in the environment such as "Elevator 2" or "Squash court B"). Other objects are not known a priori, but are detected and then associated by the fabric. Objects have type-specific attributes, such as location and velocity, associated with them.

Associated with each object type is a *detector*. Detectors may be implemented by the fabric designer or by clients. In order to load and run a detector on the fabric, a client needs to access some fabric services (such as the Instantiate service, which is discussed in Section 4).

Client: A client is an application that can use the services provided by one or more sensing fabrics. We assume a client can access a fabric, but how it does so (i.e. the medium for connectivity) is outside the scope of the architecture. Likewise, we assume that the client knows the services offered by a fabric through an *access manager* dscribed next, but how it discovers or looks up this information is outside scope.

A client may choose to trust the fabric, or it may not (preferring instead to validate the fabric-provided information by cross-referencing it with information provided by other fabrics or sources). The access rights of a client may vary from fabric to fabric. Moreover, if a client has the rights to instantiate new detector capabilities within a fabric, it may choose to make these shareable with other clients. We do not elaborate on how security and trust between clients and fabrics is realized.

To begin use of a fabric, a client registers with the fabric. It may then choose to instantiate detectors on the fabric. It would then invoke fabric services: In the case of search services, the client may selectively search for particular object types, specific objects, or objects in certain segments of the fabric. Operation invocations could be one-time, time-bounded or persistent. A client may therefore terminate time-bounded and persistent invocations.

Access manager: We assume that the client knows the services offered by a fabric through a directory maintained by an *access manager*. How the client discovers or looks up this information is outside the scope of this architecture. An access manager may manage multiple fabrics if they belong to the same administrative domain. The access manager maintains a directory of object types supported by fabrics and their semantics. The access manager also grants access permissions to a client for using a fabric. Fig. 3 shows the *Weave* architecture.

IV. Vertical API for Search

In this section, we describe the standard APIs provided by urban sensing fabrics that support application scenarios stated in Section 2. These fabrics support two generic APIs *Register* and *Instantiate*, and a vertical API *Search* specific to the "search" application domain. We propose the following specification for *Register*, *Instantiate* and *Search* APIs.

Register: To use a sensing fabric, a client first registers with the fabric and is provided a network handle which is then used in subsequent invocations of the services offered by the fabric. The $Register$ service is a generic fabric service that handles the security and authentication aspects for the fabric.

```
FabricSession Register(
Fabric F, Client C)
```

The Fabric data structure contains the name of the fabric being accessed. The Client data structure contains credentials such as id and password assigned to the client by the fabric administrator. A FabricSession is returned to the client. This may contain an expiration date and will encapsulate access rights for the client on the fabric.

Instantiate: The $Instantiate$ service is a generic fabric service that provides the client the capability to program the fabric with its own detector. Instantiate is invoked with a FabricSession and the Detector to be downloaded to the fabric. Note that in order to create a Detector, the user has to be aware of the internal API of the fabric.

```
Boolean Instantiate(
FabricSession, Detector, Shareable)
```

A detector that is implemented by a client may be made available to another client through the Shareable parameter.

Search: Once a client has registered, it can use $Search$ in order to find objects in the fabric .

```
SearchSession Search(
FabricSession, ObjectType, Object,
Locations, Parameters, Persistence,
Duration, Periodicity, SearchListener)
```

The return value of $Search$ specifies whether the call is a success or a failure. If the call fails, SearchSession is NULL. The result of the search on the other hand is returned through a separate call on the SearchSession to map to the actual Search call. The result of a search contains the status of all objects that match the search. SearchListener is a function specified by the client which the $Search$ service can call back to return the results of the search.

We now discuss the parameters of the Search API.

- ◇ *ObjectType*: This specifies the object type for which the $Search$ is invoked. The fabric uses this to invoke the appropriate detector.
- ◇ *Object*: If the client is interested in searching for a specific object of the type, this can be specified by providing the identifier of the particular instance, $Object$ (which must have been returned as part of a prior $Search$ if the object is non cooperative). If the object is one of the cooperative types, then the id is known a priori to the client.
- ◇ *Locations*: If the fabric allows selective execution of the detector on a section of the network, the Locations field specifies those sections
- ◇ *Parameters*: These are the set of parameters exposed by the fabric for a particular object type (i.e. the corresponding Detector).
- ◇ *Persistence*: This parameter specifies whether $Search$ has been invoked as a one shot or a persistent operation.
- ◇ *Duration*: This field specifies the interval over which objects that match the $Search$ should be returned. The duration of a $Search$ can be DURATIONPAST, which means that the $Search$ is for objects detected in the past. Note that the fabric may not implement any history capability, in which case, $Search$ will fail if called with DURATIONPAST.
- ◇ *Periodicity*: A persistent $Search$ invocation corresponds to tracking the status of an object. The semantics of the persistent $Search$ invocation could be either that a new result is returned every time the status of an object changes or it could be that the status of an object is reported periodically at the interval specified by Periodicity.
- ◇ *SearchListener*: This field specifies the listener function initiated by the Client which the $Search$ service can call back to return the results of the search.

We do not require that a fabric implement all possible parameters. However, the fabric must return a "failed" result if the parameters supplied are not supported.

The status of an object contains the current values of the attributes associated with the object. One example of an object attribute is location. There can also be other attributes associated with objects. For example, if a fabric implements a detector for object type "Elevator", the attributes of an object could be location and the direction of motion. Each call back to a SearchListener function will contain one record of an object containing the value of all its attributes.

$Search$ can be thus invoked in the following modes of operation *in combination*:

- ◇ *Find*: One may invoke $Search$ to simply find out if an object exists.
- ◇ *Search temporally*: Objects could be searched for a period of time that spans the future or the past. Thus $Search$ can be used for tracking an object or a set of objects.
- ◇ *Search spatially*: $Search$ may be invoked to only look for objects at a specified location or all the locations.

As part of implementation of the sensing fabric, $Search$ may internally be supported by services for association and power management. By an association service, we mean identifying new objects of a certain type, assigning an identifier to them and associating subsequent detections of an object with previously existing identifiers in the fabric. Power management services may be used to selectively activate or deactivate detectors within the sensing fabric. For example, a detector may be activated only when the first $Search$ for that object type has been invoked. On the other hand the semantics of a fabric could be to keep detectors always active irrespective of whether a $Search$ has been issued. Such fabrics could, for instance, support $Search$ over a past duration. Similarly, if $Search$ is invoked to track a particular object, power management services can selectively activate the detectors in the vicinity of the current location of the object.

Terminate: Corresponding to each of the $Register$, $Search$ and $Instantiate$ operations is a destructor $Terminate$. A persistent $Search$ that has been invoked or a detector that has been instantiated or a new network handle can be revoked using the $Terminate$ API. Terminating an active detector removes all $Search$ invocations on that object type.

V. Illustrating Search Applications Composed Using *Weave*

In this section, we describe $Search$ applications related to the urban surveillance and social networking scenarios of Section 2 that we have composed based on the *Weave* architecture.

A. Urban Surveillance Scenario

We monitor suspicious activity using our building-mounted video camera outdoor fabric and camera and motion sensor indoor fabric. The outdoor video camera fabric is instantiated with detectors to detect unusual events, such as people getting out of illegally stopped cars, atypical individual motion, crowd formation, or repeatedly circulating vehicles. Any person(s) tagged as suspicious and headed towards a building lead to cueing of the indoor sensor fabrics. The following example showcases tracking of people getting out of an illegally stopped car.

The client registers itself to the building camera fabric.

```
FabricSession_m =
Register(OutdoorCameraFabric,
Client)
```

The client is assumed to be aware of the object types detected by the fabric. In this case, it invokes $Search$ on object type 'StoppedCar'.

```
Result_m = Search(FabricSession m,
StoppedCar, NULL, LOCATIONALL,
Parameters, PERSISTENT,
DURATIONFOREVER, 0, SearchListener)
```

$ObjectType$ refers to the 'StoppedCar' detector, activated in the entire fabric, while $Parameters$ could be settings (such as semantics for a stopped car) exposed by the underlying detector. $Search$ will return every new instance of a stopped car with a different id. $Periodicity$ is set to 0 to indicate that a result should be returned only when a new stopped vehicle is detected.

The client also registers itself with the indoor photo camera fabric. Upon detection of a suspicious stopped car, it issues a $Search$ to the indoor motion triggered photo camera fabric, as follows. $Duration$ is set to the next 30 seconds (during which the object is likely to enter the building) and results are returned at a periodicity of 1 second. $Locations$ is set to the region near the entrance of the building.

```
FabricSession_ca = Register(
camera_network, Client)
```

```
Result_ca = Search(FabricSession_ca,
NULL, NULL, Locations, PERSISTENT
30, 1, SearchListener)
```

Fig. 4(a) shows a snapshot of a person getting out of a car stopped at a curb outside the CSE building (3 video cameras cover this particular area) and heading towards the CSE building with a bag. After this detection, the indoor camera fabric is activated by the client. Fig. 4(b), shows a photograph from the indoor motion-triggered camera fabric, of the same person tagged as suspicious, entering the CSE building with the bag. (These snapshots were taken during a trial run of the above scenario at OSU staged by AFRL as part of a layered sensing experiment that included airborne sensors in addition to the fabrics described in this example).

In addition, the indoor motion sensor fabric can track any suspicious person entering the building. The $Search$ issued to the motion sensor fabric using a session handle $FabricSession\_mo$ is shown below.

```
Result_mo = Search(
FabricSession_mo, Human, NULL,
Locations, PERSISTENT, 10, 1,
SearchListener)
```

The object type is set to Human for which a detector is implemented in the fabric. $Object$ is set to NULL because the object type is non advertised and an identifier has not been assigned. $Duration$ is set to the next 10 seconds when the object is likely to enter the building and the results are returned at a periodicity of 1 second. $Locations$ is set to the region near the entrance of the building.

If the suspicious person enters the building, a new object instance is created by the motion sensing fabric and returned to the client. The client then terminates the previously issued $Search$ operation and invokes a new $Search$ with the returned id.

```
Result_mo = Search
(FabricSession_mo, Human,
"SuspiciousPerson", LOCATIONALL,
PERSISTENT, DURATIONFOREVER, 5,
SearchListener)
```

Note that in the previous call, "SuspiciousPerson" is actually an instance of the object type 'Human' returned by the motion sensing fabric. The first invokation of $Search$ is performed by specifying the parameter as NULL as the id of any individual obejcts is not known. This invokation of Search returns every new instance of object type HUMAN with a separate identifier. When any person enters the building,"SuspiciousPerson" is actually the identifier assigned to this object. The previous $Search$ is terminated and a new $Search$ is invoked with this identifier to track the person that just entered. If the fabric cannot clearly associate an id with an object all the way, it tracks all candidate objects.

B. Social Networking Scenarios

We now describe Weave-based application associated with two locality specific query scenarios, that are composed to originate from the PeopleNet mobile fabric and to be realized using other sensing fabrics mentioned in Section 2.

Example 1 (Where is Vinod?): Client Anish in the CSE building, who is scheduled to play squash with Vinod in RPAC, wishes to find out if Vinod is still in the CSE building or has already reached the RPAC. In this case, the supporting application to determine if Vinod is in the CSE building or in RPAC connects to the instances of the PeopleNet networks in CSE and in RPAC, and invokes the $Search$ for the id associated with Vinod's mobile device (the application is assume to know the id) in both networks. The PeopleNet fabric supports Search operations on cooperative objects, and can provide a more precise location as well.

The following is the sequence of *Search* invocations in the client application. (We omit *Register* operations.)

```
Result_cse = Search(
FabricSession_cse, ADVERTISED, Vinod,
LOCATIONALL, ONCE, NOW,
NULL, SearchListener)
```

```
Result_rpac = Search(
FabricSession_rpac, Advertised, Vinod,
```

(a) Person getting out of stopped car, tagged as suspect

(b) Picture of person tagged as suspicious, entering CSE building

Fig. 4. Campus surveillance scenario

```
LOCATIONALL, ONCE, NOW,
NULL, SearchListener)
```

$ObjectType$ is set to $ADVERTISED$ and Vinod is the object specified. The search is invoked as a one shot operation, and Periodicity is set to NULL.

Fig. 5 shows localization of mobile units in PeopleNet in an early trial run, using a combination of static anchor nodes and collaboration among mobile units themselves. This sort of service supports the response to client issued *Search* on self advertised objects.

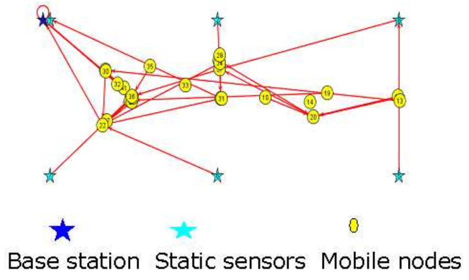

Fig. 5. Localization of Mobile Units in PeopleNet

Example 2 (Is there an empty squash court?): Anish then wishes to find out if any squash court in RPAC is free. The PIR sensing fabric in the RPAC building implements a detector for object type 'SquashCourt'. The detector returns one of two values, $full$ or $empty$, for every instance of squash court. These are cooperative objects that belong to the sensing fabric itself. Each squash court itself has an id associated with it and the application to answer Anish's query can invoke a $Search$ operation on an individual court also.

```
Result_pir = Search(
FabricSession_cse_rpac, SquashCourt,
NULL, LOCATIONALL, ONCE, NOW,
NULL, SearchListener)
```

$Search$ is invoked in this example to return the status of all squash courts by setting the id to NULL.

Note that in both examples, we have abstracted from the client the issue of the underlying communication. In other words, the client is oblivious to communication network that ensures connection with the RPAC PeopleNet fabric and the RPAC PIR fabric and that ensures retrieval of information from these fabrics .

VI. Related Work

Two notable aspects of our proposed Weave architecture are (i) it simplifies design of applications across heterogeneous sensor networks that are independently managed and (ii) it views a sensor fabric as providing an extensible database of queryable objects. SenseWeb [1] and GENI [2] are related to (i) in that they propose architectures for heterogeneous networks. TinyDB [3], SemanticStreams [4] and ASAP [5] are frameworks related to (ii).

SenseWeb: SenseWeb, an architecture proposed by Microsoft Research, allows sensing applications to be developed which use contributing sensor networks from across the globe. A (presently centralized) engine provides a uniform set of APIs, to which applications and sensor networks from different domains connect to subscribe/visualize and publish data respectively.

Weave, in contrast, is distributed in that applications directly access individual sensor network fabrics. The fabrics provide applications with the services themselves, as opposed to simply publishing data, unlike SenseWeb where application process data acquired from the engine. And the fabric APIs need not all be identical.

GENI: GENI proposes a framework for experimentation across different types of networks, including wireless subnets and sensor networks. GENI has a notion of user services, services available for researchers or users of a particular network to access its underlying resources. The services that we propose may indeed be viewed as an instance of GENI user service; for instance, they allow users to instantiate their own programs on the underlying network resources and to access the associated results.

Note that Weave services also allow users to access data generated by other network services or other user-supplied programs. This contrasts with the default GENI virtualization requirement that different user programs be isolated from each other. We assume that some users may opt-in to share the object data resulting from their programs with other users. (It is up to the latter to decide whether or not to trust the data.) For instance, a user can instantiate a network with a new detector for objects of type car and another user can query the network for objects of that type. In this sense multiple users of a network are not isolated from each other.

TinyDB: TinyDB is a query processing system for extracting data from within a sensor network. Users specify data requirements as TinyDB queries, the data is then extracted from the network using appropriate aggregation and filtering mechanisms.

Weave also views each sensor fabric as a database, but its queries (which can be TinyDB-like) are posed on objects, which are semantic values exposed by the fabric potentially via user supplied programs, as opposed to raw sensor data. Weave allows composing applications such as tracking across different networks, which are much more complex than those with TinyDB. TinyDB does not focus on extracting data from multiple independent networks.

SemanticStreams: SemanticStreams provides a framework for describing and composing applications on semantic values inferred over sensor data, such as person, car or truck. A primary contribution is an interpreter for concurrent high-level applications executing on a single network that optimizes the design optimum design of underlying network services while satisfying the requirements of all the applications.

By contrast, we expect that user services be already implemented for the sensor fabric and available to different applications. At run time, upon invocation by an application, a user service is allowed to optimize its operation but this is not a goal of the fabric design. Our goal is to provide a standard set of APIs that allow applications to be tailored across different sensor fabrics.

ASAP: ASAP focuses on optimizing urban sensor network applications based on priority and situation awareness. The focus of that paper is on designing an ASAP agent, which implements a query provided by the user on the underlying sensor networks based on the knowledge of underlying network interfaces in an optimized manner. This is complementary to our architecture, where we focus on a standardized interface for sensing fabrics so that composing applications is facilitated.

Other related work: In [6], the authors propose 4 architectural requirements for urban participatory sensing, namely in-network verification of location and time context, provisions for operating on physical context based on sensor readings, enabling selective sharing of information and services for naming, dissemination and aggregation. In this paper, we have proposed an architecture that allows aggregation of data in space and time and allows individuals to coordinate activities and thus partially address their requirements on sharing of information and aggregation.

Handling location is a basic requirement for the *Weave* architecture (as for most pervasive computing systems). [7] provides a comprehensive taxonomy of location that will be of great use in structuring extensions to the handling of this concept in *Weave*, such as the Locations parameter of Search. Urban sensing applications are a subset of the pervasive computing applications considered in [8] which presents a taxonomy for characterizing, and providing a controlled vocabulary for thinking about, such applications. Essentially this taxonomy will provide additional architectural requirements to *Weave* as we develop it further.

VII. Conclusions and Future Work

We have presented the *Weave* architecture for composing applications that use across one or more urban sensing fabrics. The architecture leans towards the use of standard APIs for sensing fabrics. Some of these are generic to all urban sensing application domains, while the rest are vertical APIs specific to an application domain. Standardizing even the vertical services is desirable so that applications can be readily composed for and ported across fabrics that support a particular application domain. In this paper, we detailed an API to support the class of search applications that we have encountered in diverse setting. We have validated our $Search$ API by showing its use in the composition of sample applications in operation scenarios that we have implemented at OSU.

While sensing fabrics for urban sensing applications belong to one class of fabrics, similar vertical APIs can be standardized for other classes of sensing fabrics. An example is that of testbed fabrics, and we have in other work been outlining a vertical API for this class.

We plan to build upon this work in three directions. The first is to design and implement efficient lower level services necessary to support the fabric model. Design and implementation of these services is a current focus. Examples include power management services that efficiently manage resources across requirements of multiple clients (including user supplied detectors) and scheduling services that ensure fairness and security.

The second direction of work is security. In our implementation of Kansei [9], a static wireless sensor network testbed at Ohio State University, we have the following security features: (1) client authentication with a trusted manager, and (2) running each client program on a different wireless channel to prevent interference. We are currently working on other security features such as detection and prevention of jamming and forming a restricted set of hardware APIs that can be accessed by client code to operate on the network.

Implicit in the *Weave* standard is a standardization of the entities within the fabric, namely, Objects, and their attributes, namely, Identity and Location. The Location attribute also served as a relationship between the object and the fabric. Building on this, a third direction is towards a framework for knowledge-based urban sensing. We plan to develop a richer ontology for urban sensing that builds up from physical phenomena, sensing signatures, detectors, and richer classes of objects. Our goal is to then use this ontology to enable queries with richer semantics, as well as model-driven application monitoring, management, composition and generation.

VIII. Acknowledgements

We thank the students and faculty of the newly formed Institute for Sensing Systems at OSU and Dr. Todd Stewart, Director of OSUs Homeland Security Program for several insightful discussions. We also acknowledge the help provided by the Air Force Research Laboratories in Dayton, and Brett Gerke, Ohio Association of Chiefs of Police.

References

[1] A. Kansal, S. Nath, J. Liu, and F. Zhao, "SenseWeb: An Infrastructure for Shared Sensing," *IEEE Multimedia*, vol. 14, no. 4, pp. 8–13, 2007.

[2] "Global Environment for Network Innovations: GENI System Overview," Prepared by GENI Project Office, Document ID: GENI-SE-SY-SO-01.1. [Online]. Available: http://www.geni.net/doc/GENISysOvrvw1.1.pdf

[3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An Acqusitional Query Processing System for Sensor Networks," *ACM Transactions on Distributed Systems*, vol. 30, no. 1, pp. 122–173, 2005.

[4] K. Whitehouse, F. Zhao, and J. Liu, "Semantic Streams: A Framework for Composable Semantic Interpretation of Sensor Data," in *EWSN*, 2006, pp. 5–20.

[5] J. Shin, R. Kumar, D. Mohapatra, U. Ramachandran, and M. Ammar, "ASAP: A Camera Sensor Network for Situation Awareness," in *International Conference On Principles Of Distributed Systems (OPODIS)*, 2007.

[6] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory sensing," in *World Sensor Web Workshop, ACM Sensys* , 2006.

[7] S. Dobson, "Where's Waldo? - or - A taxonomy for thinking about location in pervasive computing," Trinity College Dublin, Tech. Rep. TCD-CS-2004-05, 2004.

[8] K. Dombroviak and R. Ramnath, "A Taxonomy for Pervasive and Ubiquitous Computing Applications," in *Symposium for Applied Computing*, 2007.

[9] A. Arora, E. Ertin, R. Ramnath, M. Nesterenko, and W. Leal, "Kansei: A high fidelity sensing testbed," *IEEE Internet Computing*, vol. 10, no. 2, pp. 35–47, March 2006.