Towards Semantic Services for Sensor-Rich Information Systems

Jie Liu
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
Email: liuj@microsoft.com

Feng Zhao
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
Email: zhao@microsoft.com

Abstract— **This paper describes the architecture and programming model of a semantic-service-oriented sensor information system platform. We argue that the key to enabling scalable sensor information access is to define an ontology and associated sensor information hierarchy for interpretation of raw data streams. The ontological abstraction allows a sensing system to optimize its resource utilization in collecting, storing, and processing data. We describe the SONGS architecture that uses an automatic service planning to convert declarative user queries into a service composition graph, and performs compile-time and run-time optimizations for resource-aware execution of the service composite in a sensor network, building on the sensor information hierarchy. We motivate and demonstrate the SONGS platform using a parking garage example.**

I. INTRODUCTION

Networked sensing promises to drastically change the way people interact with the environment by providing rich, real-time information about the physical world. To be usable by ordinary users and to support multiple applications simultaneously, these systems must be easy to deploy, program and manage, and interoperate with each other as well as with the existing IT infrastructure. We begin with the following use scenario.

Liz is a site manager at the high-rise CEDAR office building in downtown B city. As one of their major customers has just moved to another site, she wants to evaluate the idea of opening part of the parking space underneath the building to the general public. To help her to reach a decision, she wants to collect vehicle arrival and departure statistics in the parking lot for a period of two weeks. Pablo is a security officer for the CEDAR building. He is investigating complaints from people that there are cars driving extremely fast in certain parts of the garage. He wants to take pictures of those cars and issue warnings to offending drivers. Cameo is a local law enforcement agent in B city. He is in charge of installing chemical sensors at strategic locations throughout the city for terrorism detection. He has installed some of them in the CEDAR building garage, and wants a notification whenever a vehicle carrying certain chemical elements is detected. Although they are from different organizations, Liz, Pablo, and Cameo all plan to use several generic wireless sensor networks co-located in the garage and augment it with special purpose event detections as needed. These sensor networks were gradually deployed over time and are managed by different organizations. In addition, certain information, such as car images, may be privacy sensitive that should not leave organization boundaries.

The scenario illustrated above in the CEDAR building is an example of *sensor infrastructure*. We envision that a large-scale sensor infrastructure is likely to be deployed by multiple vendors incrementally. The system may consist of both mobile and stationary nodes with varying capabilities. Sub-systems from different vendors should interoperate easily. They must be integrated seamlessly into the rest of the IT infrastructure and must provide intuitive interfaces for remote and novice users. There will be multiple, concurrent clients exercising different functionalities of the system for different purposes. Although resources may not be the most stringent constraint, the system has to be self-monitored and resource-aware. It must have a certain level of autonomy to decide on the best use of available resources to fulfill multiple users' concurrent uncoordinated requests.

Systems like these are not readily supported by the existing networked embedded system technologies developed by the sensor network research community. Most sensing systems today are designed for either data collection purposes or domain-specific applications. In the data collection case, a sensor network is regarded as data collection and aggregation fiber, and the interpretation of raw data is left to end users. This is fine for scientific applications where the domain is fairly narrow and scientists prefer to play with the data to test different hypotheses about the data, as for example in the experiments on Great Duck Island [7] and in James Reserve [11]. The database approach (e.g., TinyDB [6]) is a nice example of how SQL like operators can provide a limited but useful interface for users to interact with the raw sensor data. The IrisNet project [3] goes a step further with the introduction of data schema for sensor data, to permit data query over an Internet scale infrastructure. In these systems, however, the data interpretation programs are written by users and there is little reuse of the application software components. In the application-specific case, as for example in detection [2], [8], tracking [14], and pursuer/evader games [12], low level sensor signals are converted into high-level events for the system or users to react on, but the system architecture is closed and the event semantics is hard-wired into an application. What is needed is an open architecture that allows multiple ordinary users to post a diverse set of tasks on demand.

One major barrier to the wider adoption of sensor net systems by novice end users lies in the lack of an information framework with which one can ask what information is to be desired and how the information is to be extracted from raw and noisy sensor data. There are often more than one way to obtain a piece of information, and one has to understand the information equivalency relations over the rich variety of sensor data. In this paper, we argue that an open sensor-rich information system requires a semantic information hierarchy and a set of semantic services to process and reason about sensor data, moving beyond just protocol agreement and data format conversion. To quantify the semantic information contained in sensor data and to capture the relations between various semantic entities such as objects and events in the physical world, we need to define an ontology for a variety of commonly encountered sensor data. We must provide a set of transformations, or *semantic services*, that can incrementally extract new semantic information from lower level data.

For example, one sensor network in the CEDAR building can be a set of infrared break beams that is used to detect moving objects. A pair of beam breaking and un-breaking events (called a pulse) indicates an object passing through. The raw sensor data can be hard to interpret by ordinary users. But, by using a sequence of these sensors, one may distinguish whether the passing by object is a human or a vehicle. By estimating the vehicle's speed and length, one may further infer whether it is a car or a truck. Alternatively, a magnetometer can detect a vehicle through sensing the magnetic field disturbance the moving vehicle generates. One may also distinguish a car from a truck by inferring the amount of metal the object has. Of course, this requires even more domain knowledge of magnetic fields.

Now, if the system has access to a common ontology that defines concepts such as "mobile object", "vehicle", "car", "truck" and their relations, as well as properties such as "speed", "length", "direction", "metal", "pulse", and "magnetic signal", one may encapsulate sophisticated domain knowledges and signal processing algorithms into computational components, or services, that can be reused. Each service simply transforms input events into output events, adding new semantic information as necessary. For example, a `SpeedEstimation` service may take a sequence of pulses and produce the speed of the object.

Using the ontology, the end users can specify what she/he needs to compute, say "detecting a red car passing the garage entrance", without explicitly referring to low-level sensor signals such as break-beam events or camera images; the system can take care of instantiating the requisite semantic services, when wired together, will produce the required semantic events, i.e., the red car events. The system may even choose among alternate detection methods to satisfy contextual or resource constraints.

The rest of the paper devotes to derive the framework of using semantic services as a programming model for sensor-rich information systems. Section II describes a system architecture for sensor infrastructures. Section III presents the semantic service model. Section IV discusses an implementation of the semantic service programming model, named SONGS[1]. Finally, Section V shows how the CEDAR building example can be built using semantic services and SONGS.

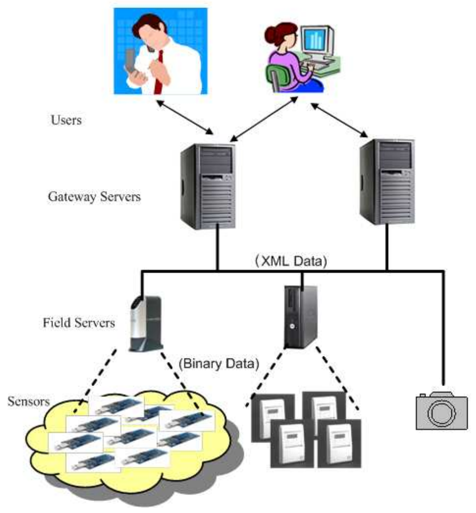

Fig. 1. A hierarchical architecture for sensor information system

II. AN OPEN, HIERARCHICAL SYSTEM ARCHITECTURE

We envision a hierarchical architecture for sensor infrastructures, consisting of *sensors*, *field servers*, and *gateway servers*, as illustrated in Figure 1. Note that the separation of these roles is logical rather than physical. Multiple logical entities may co-exist on a same physical node. Or, a single logical entity can be distributed or replicated among multiple physical nodes.

At the bottom level, various sensor nodes are capable of gathering data from the physical environment and communicate among each other or to higher level nodes through wireless or wired network. The energy, computation, and communication resources at these nodes may vary. The amount of data sensors generate also varies, for example, a couple of bytes per minute from small battery powered wireless sensor motes to mega bytes per second from video cameras.

The next level entities in the hierarchy are field servers. A field server directly connects to a set of sensors. Since these roles are logical, the link between sensors and field servers may not be a network. For example, an Internet-ready web camera can be viewed as a bundle of a camera sensor and a field server, although this field server may only provides limited service to the outside such as unprocessed image sequences. A field server converts sensor data, which are usually in platform-specific format into something that is open and directly usable, such as XML data format or standard image encoding. As a server, it may be capable of

[1]SONGS stands for Service-Oriented Network proGramming of Sensors

processing sensor data in response to specific user tasks. It may provide only aggregated data to reduce communication resource utilization, as well as, if necessary, protect privacy.

Users interact with a sensor infrastructure through gateway servers where user requests and sensor information flow through. A gateway server can be a web server, which allows users to query real-time sensor information through a browser. It may contain a planner that converts user requests into tasks that can be executed on a set of field servers. It may employ data-base servers that archive history information to be fused together with real-time information.

Either field servers or gateway servers can be at the organizational boundary. Since field servers are closer to the sensor nodes, exposing them to trusted partners results in a more flexible and resource optimal architecture. On the other hand, since there are usually a smaller number of gateway servers than field server, using only gateway servers as public access point is easier to manage.

This paper focuses on the discussion of software architectures and programming models for gateway servers and field servers. We treat sensors as simple data collection front end. For example, a field server can turn on and off a sensor node, and may set its sampling rate.

III. SEMANTIC SERVICES

Semantic services operate on semantic streams, where each semantic stream is defined in a domain-specific ontology.

A. Ontology

The notion of ontology used in this paper is to capture the information about physical entities sensors are trying to sense and their relations. Having a common ontology for each application domain helps unify information presentation and permits software and information reuse. Defining a sensor information ontology is not the focus of this paper. It is suffice to understand them as a common information hierarchy describing the data (streams) that can be provided by a sensor information system.

For example, using a UML class diagram, the left part of Figure 2 describes a simple information inheritance hierarchy for the vehicle detection scenario in the CEDAR building example. Notice that classes are used here to capture data types. That is, a class represents a concept in the ontology and the data fields of the class represent the possible properties that the conceptual object may have. We explicitly introduce `unknown` as a possible value for properties to allow reasonings on partial information. For example, the diagram captures that cars and trucks are sub-classes of vehicles, which are sub-class of mobile objects, as well as metal objects. The notion of `unknown` is particularly useful for semantic conversion, that not every property field of an object, say vehicle, needs to be filled before certain detection can be performed. For example, to distinguish a vehicle from a person, one can test if the object is a metal object or not, which uses the structural information defined in the ontology, or one can test if the length of the object exceed a threshold, which exploits differences in object property values to classify objects.

The right half of Figure 2 shows pulses and MagSignals as signals. A signal is a time-tagged set of measurements. The diamond symbol in the diagram shows a collection relationship. For example, a SignalGroup is a collection of Signals.

1) Uncertainty: Sensor data are noisy, due to the nature of physical environment. Uncertainties in sensor data may come from mis-calibrated devices, sensor positioning and orientation, background noise, thermo-noise, communication error, and so on. These uncertainties will eventually propagate through data processing to high-level information. Capturing data uncertainty and its effect on the information equivalency in the ontology is very important since it constrains what data can be used as input to a service and what are the different ways to produce an output. Many sensing applications can tolerate a certain degree of noise and uncertainties.

The purpose of modeling uncertainties is to enable the analysis of uncertainty propagation when higher level information is extracted. How to represent information uncertainty is domain specific, and is a wide open issue. Sensors may come with noise-level specifications in their datasheet, indicating the uncertainty in the raw sensor data. False positive and false negative rates are usually used to characterize event detection performance, assuming certain noise model at the input. A belief state, i.e., a probability distribution of estimated variable over a set of possible values, may be natural for estimation.

2) TEDS and SensorML: There are several industrial standards that aim at unifying sensor interfaces and data formats. The IEEE 1451 standard family [1] uses Transducer Electronic Data Sheet (TEDS) to capture sensor parameters, such as transducer identification, calibration, correction data, and manufacture-related information. A TEDS is a text table completely focused on describing the capability and characteristics of individual sensors. It does not capture any derived semantics from sensor data, nor does it capture how the sensor is used, such as the location, age, or condition of the sensor.

The Sensor Modeling Language (SensorML [10]) has recently been proposed by Open Geospatial Consortium (OGC). The goal of SensorML is to describe the geometric, dynamic, and observational properties of dynamic sensors. SensorML goes beyond just describing individual sensors. The model puts sensors in a context, in most cases, geospatial observation. In this specific domain, the language allows sensor providers to describe in situ what a sensor can observe, with what accuracy, etc. The language also introduces the notion of virtual sensor as a group of physical sensors that provide abstract sensor measurement. For example, the breakbeam and magnetometer sensors collectively monitor cars as “car sensors”.

Sensor information ontology is different from both TEDS and SensorML in the sense that it captures semantic entities and relations of events and objects beyond what sensors can directly provide. The ontology can build on top of the schemas provided by SensorML, but the sensor data format and data semantics is only a small part of a domain-specific ontology.

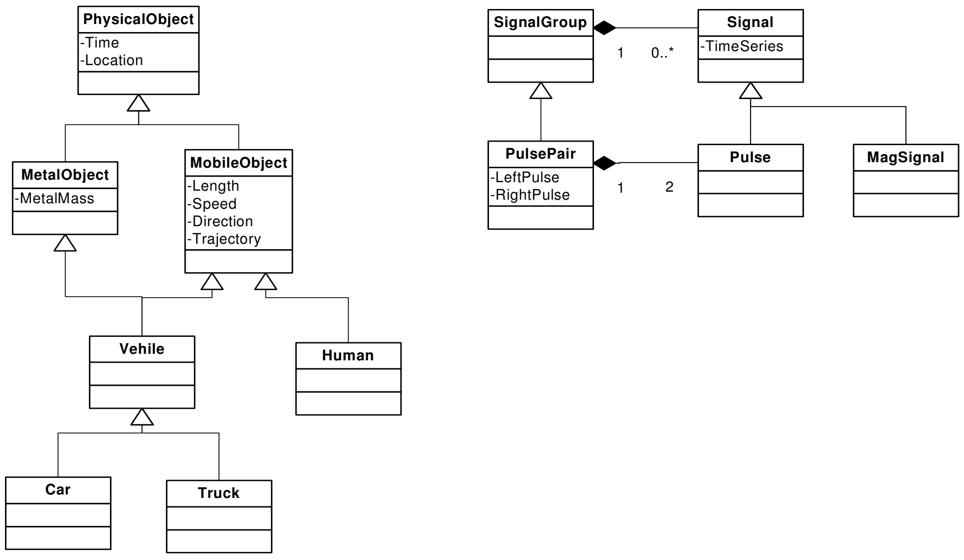

Fig. 2. A UML representation for a vehicle/human detection ontology.

For example, in the information hierarchy shown in Figure 2, the majority of the concepts are about the application domain. To some extent, in a traffic sensing system, being able to detect a vehicle and estimate its behavior is what users care about. The fact that it is using beam sensors or magnetometers is just an implementation artifact.

Having common ontology for application domains allows users to use high-level sensor-rich information as first class objects in their applications. For example, a user can directly query a sensor network using events such as "images of speeding vehicles at the entrance of the garage." Or, a high-level event can be used to trigger reactions in an event-driven programming model. For example, a programmer may write:

```
ON Vehicle.Speed > 15 DO {
   take a picture of Vehicle...
}
```

B. Semantic Services

With the help of sensor information ontology, one can design computational components that convert between elements in the information structure. For example, a truck detector may output truck events from the length estimation of a mobile object, regardless of how the length estimation is obtained. We call these components *semantic services* (or *services* in short), since their primary role is to extract new semantic information from existing data streams.

Pragmatically, semantic services can be implemented by software components with ports, similar to UML action model [9] and actor-oriented design [4]. A port can be input or output. Data streams communicated among these ports carry specific semantics as defined in a domain-specific ontology, which we call *semantic streams*. A service has a single execution method, which consumes a number of events from the input ports, and produces a number of events at the output ports.

Note that a semantic service can either pass through the input events with additional semantic annotation, or can produce completely new data streams. In the latter case, the input semantic stream terminates at the service. A pass-through service only constrains what additional properties the output event can have. It does not affect the properties the input event carries. Thus, the pass-through services are easy to reuse.

The essence of semantic services is that they perform semantic transformations rather than simply data transformations. That is, the semantics of the input and output streams have to be different and must be expressed explicitly in the sensor information ontology. This makes semantic services rather coarse-grained software components. In particular, a service may not be a centralized component that exists only as one piece. It can be implemented on distributed nodes but logically be one service. There may also be many alternatives of implementing the same services, such as using different sensing modality or signal processing algorithms. These semantically equivalent services provide flexibility for users to select particular implementation, but also give rise to the complexity in making trade-offs. These trade-offs are usually due to the non-computational aspects of services.

Non-Computational Aspects of Services: Semantic inference can be viewed as the computational aspect of services, in the sense that they compute new data semantics from existing ones. There are also non-computational aspects of services that relate to the resource utilization, system configuration requirements, performance guarantees, and information qualities. These non-computational aspects determine how many services can be executed and how many users can be supported at the same time; how the resources in the network are used; what quality of services, in terms of e.g. end-to-end latency, can an application deliver; what are the uncertainty measures for given inputs; and so on. Since sensor infrastructures are designed as open systems that dynamically accepts user tasks, it is crucial to optimally use the available resources to serve as many clients as possible.

How to model non-computational properties of services is an open research question, especially because of the heterogeneity of devices and communication links. A component may deliver a different quality of service if executed on a different platform. Since the system resources may change during the course of a long running sensor information application, the execution of services must dynamically adapt to resource variations to preserve quality requirements.

C. Service Composition

Services provide a layer of abstraction for building sensor information applications. Services are composed by connecting their input and output ports with compatible semantics. Conceptually, the execution of a service is event-driven. That is, a service waits for input events at its input ports, when an expected set of events present, it react to them by consuming the events, and produces a new set of events at its output ports. The connections between input and output ports of different services have publish-subscribe semantics. When multiple input ports connect to the same output port, each of them receives a copy of the output event.

Figure 3 shows a composition of a set of semantic services for detecting vehicles via length estimation using two infrared beam sensors located a distance *d* apart. The semantics of each event stream is labeled. The `MotionEstimator` service converts a `PulsePair` semantic stream into a `MobileObject` stream with `Speed` and `Length` properties. The `VehicleClassifier` further derives a stream of `Vehicles` if the lengths of the mobile objects are larger than a threshold.

The benefit of abstracting services as semantic conversions on a common ontology includes the possibility of automatically compose services to satisfy a semantic query and the possibility of adapting to resource changes by switching to different services producing the same output semantic streams. In the automatic service composition model, the users do not need to compose the services manually to create a sensor information application. They can simply issue a query that request data streams with desired semantics to the gateway server. A query planner can search through all possible services and sensor configurations for the best service composition that can answer the user queries.

IV. SONGS: A SERVICE-ORIENTED PROGRAMMING MODEL

SONGS is an implementation of semantic service frameworks for sensor infrastructures. SONGS is built on top of a hierarchical system architecture as described in section II. In SONGS, the field servers provide the library and manage the execution of semantic service. The gateway servers accept user tasks (in the form of semantic queries) and derive optimal service composition plans. These plans, in the form of service composition graphs, are sent to one or more field servers for execution. So, by using SONGS, the users do not have to construct their own service compositions, such that they can interact with sensors through high-level semantic queries. Another benefit of using automatic planning at the gateway server is the possibility of combining multiple uncoordinated user tasks. For example, in the CEDAR building scenario, all three tasks contain detecting vehicles in the parking garage. In addition, Pablo's task specifically asks for the speed estimation of the vehicles. A resource optimal plan should use vehicle detection through speed and length classification to answer all three tasks, rather than, for example, using speed estimation for Pablo but using magnetometers for Liz.

Figure 4 illustrates an instantiation of SONGS. We discuss the key components in this flow in the following sections.

A. Service description

In SONGS, services are implemented as software components on the .NET framework. A service description is a "rich" interface for services that describes the data semantics required for the services to execute, as well as the new semantics the service creates at its output ports. The input semantics is called the *pre-condition*of the service; while the output semantics is called the *post-condition* of the service. Services can be parameterized. These parameters can be used to build constraints that the service must operate within.

Sensors themselves are treated as services with only output semantics, which abstracts the data that they can produce. For example,
`sensor(breakSensor,[[25, 43, 0],[26, 60, 1]])`
defines a break-beam sensor that covers range [[25, 43, 0],[26, 60, 1]], where the range is a pair of [x, y, z] coordinates that defines a cube in the 3-D space.

The following is a definition of a break service that takes break-beam sensor input and produces pulse events.

```
service(breakService(Region),
   needs(sensor(breakSensor,Region)),
   creates(Pulse(X),detected(X,T,Region))
).
```

The `service` keyword indicates that this is a service with name breakService, which is parameterized by a region. It requires a sensor of type breakSensor as its input, and inherits the region defined by the sensor. As output X, it creates events of type Pulse, and further asserts that the event is detected in the particular region, at a particular time.

Services can have constraints that quantify its non-computational properties. For example, a camera service is described as:

```
service(triggerablePhotoService(Y,Delta,Region),
   cstrnbl(Delta),
   needs(sensor(camera,Region),
         detected(Y,T2,_ )),
   creates(photo(X), detected(X,T,Region),
         triggered(X,Y), delay(T,T2,Delta))
).
```

where `Delta` is a constrainable variable that indicates the delay from the time it is triggered to the time a photo is output. This service needs a camera sensor and has input Y

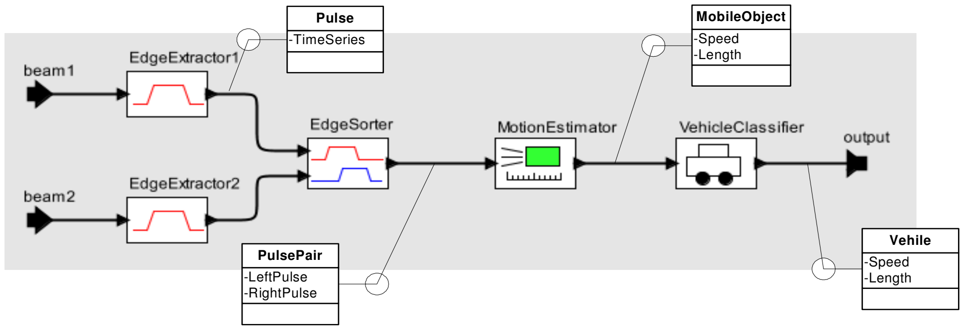

Fig. 3. A vehicle detection application composed from semantic services.

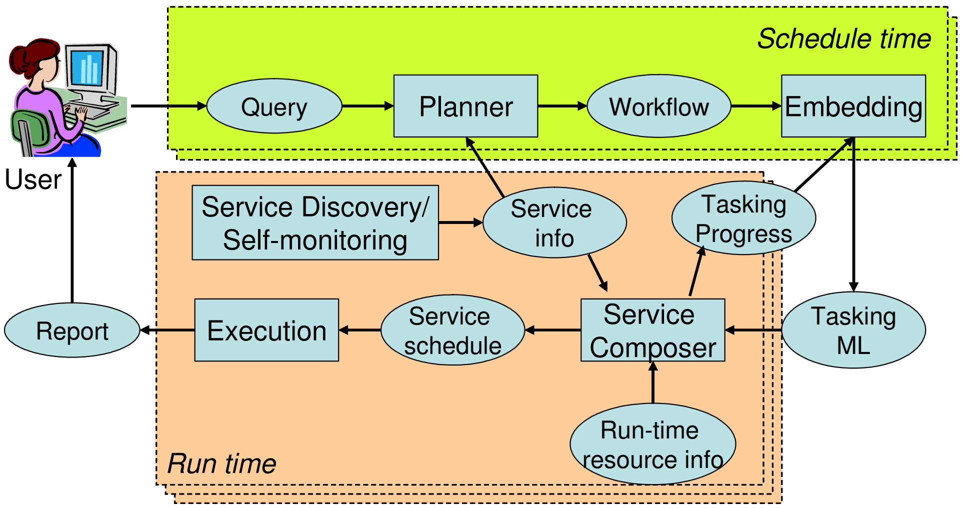

Fig. 4. A illustration of how to SONGS architecture.

as the trigger. It assets that the output X is triggered by Y with an additional delay `Delta`.

The existence of services depends on both software installation and system status. A field server can monitor connectivity, life-time, and reliabilities of the sensors and itself, and modify the service descriptions accordingly. The service descriptions are used by service planning to build service composition graphs on demand.

B. *Service planning*

When a user issues a query to SONGS, the query is first decomposed by the planner into a set of services forming a service composition graph or workflow. In SONGS, the service planner is implemented as a logic inference engine with a constraint programming extension CLP-R.

Service descriptions are converted by the service planner into a set of logic rules with constraints. For example, a sensor description is simply a fact. The `needs` and `creates` clauses in a service definition are conjunctions of first-order logic predicates. A user query is also converted into a set of predicates. The inference engine then performs a backward chaining in order to "prove" the use query. The steps that are used to prove the query is the service composition graph.

For example, the service composition graph in Figure 3 can be viewed as a proof for "vehicle streams at region R". A `VehicleClassifier` can produce a `Vehicle` stream, given a `MobileOject` stream with `Speed` and `Length` properties. A `MotionEstimator` service can provide `MobileObject` if it gets `PulsePair` in that region, and so on.

A detailed discussion of the planner can be found in [13]. It is useful to point out that when proving the user query, the planner tries to reuse existing streams if it has already been instantiated by existing services or queries. By doing so, the planner can co-optimize multiple user queries by using a minimum set of sensors and services.

The planner has a constraint solver that differentiate otherwise equivalent service compositions. Constraints can come from service descriptions or user queries. For example, a sensor may have a confidence level of certain event detections depending on its sensing modality and location. Sensor fusion services may specify how its output confidence is related to the input confidence. A user may request that the event be detected with confidence higher than, say 90%. The service planner can then derive which set of sensors to pick and which set of services to use, in order to satisfy all the constraints. If there are parameters of services as parts of the constraints, the constraint solving process can also derive the values for these parameters. If the constraints are too stringent that no feasible plan can be found, the planner can also output partial plans. The partial plan suggests where to add new sensors or new services.

C. Service embedding

The output of service planners are logical service dependency graphs. An extra step called *service embedding* is needed to assign each service to a particular node in the network.

The service embedder first converts a service dependency graph into an abstract syntax graph, similar to the one shown in Figure 3. The abstract syntax graph describes components, ports, and relations that connect ports. All of them can be annotated with properties. This is the intermediate representation for service composition that a chain of optimization tools can operate on.

The service embedding process has to take a number of factors into account, such as field server load, network connectivity and reliability, network bandwidth and latency, and power utilization. Since not all platform dependent information is available to the planner, some of these trade-offs have to be performed after the planning. Thus the boundary of service planning and service embedding has to be carefully examined, to avoid ruling out alternative service compositions too early. One possibility is to have the service planner to output all possible service compositions that satisfy static constraints, and have the service embedder to pick the best option among them based on platform-specific information.

Furthermore, due to platform resource variations during the course of long running executions, the service embedding results may only serve as an initial embedding subject to run-time adaptation.

D. Service runtime

Partitioned service composition graph is then injected into the network in the form of a tasking description language — the micro-server tasking markup language, or MSTML. This description is accepted by a service scheduler running on each node. In order to control memory consumption, not all services are preloaded. They are created only upon request. The service scheduler is also responsible for checking all other services running on the same node to determine whether part of a new task can be achieved using parts of other tasks. After this optimization step, the task is admitted to execute. Because some tasks are instantiated on-demand, service schedulers may negotiate with the planner to iteratively achieve a feasible and optimal service allocation.

Runtime services may have constraint specification carried from the planning and embedding process. The service execution engine can monitor these constraints. If, due to resources change in the network or migration of physical phenomena (such as in vehicle tracking), an existing service assignment is no longer available, the runtime system can first try to fix the assignment locally, and if it fails, request the planner for a re-plan. Part of the local fixing strategy could be to migrate services to other nodes. When tasks terminate, the execution engine is also responsible to clean up parts of the task that is not shared by other tasks.

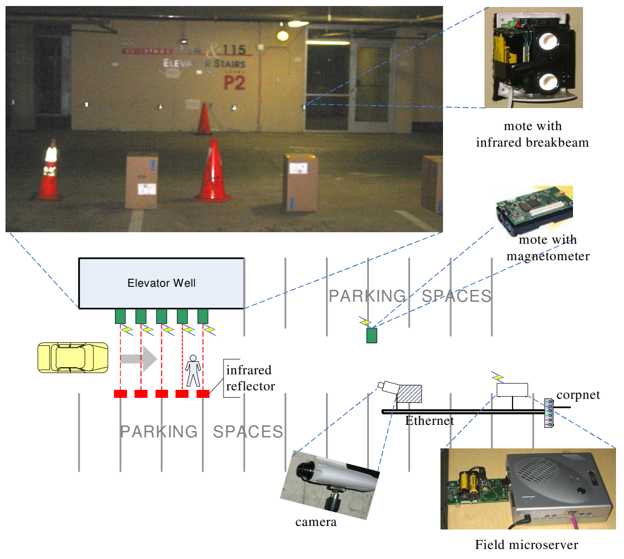

Fig. 5. A parking garage deployment of a sensor infrastructure.

We should point out that it is sometimes useful to carry event semantics all the way from planning stage to service runtime. This can be particularly useful if multiple gateway servers task a set of field servers with no coordination. Having the semantic information at runtime can facilitate low-overhead run-time service sharing and optimization [5].

V. PARKING GARAGE SCENARIO REVISITED

We have built a small-scale sensor infrastructure in a parking garage that emulates the CEDAR building scenario described in section I. As shown in Figure 5, an array of 5 infrared beam sensors is deployed to detect vehicles. Each consecutive pair of beam sensors can be used to estimate the speed and length of the vehicle.

These speed estimations can be averaged together to improve accuracy. A magnetometer is also placed near the beam sensor array as another modality to detect vehicles. A web camera points to the vehicle passing area. The camera can be triggered to take a picture. Another magnetometer (`Mag2`) is used to simulate the chemical sensor. Mag2 can be triggered by an event to collect a periodically sampled continuous signal.

Three queries are sent to the service planner:

- Liz: collecting vehicle arriving time histogram.
- Pablo: taking pictures of speeding vehicles.
- Cameo: taking the magnetometer signature for every vehicle.

The output of the planner, after converted to an abstract syntax graph is visualized in Figure 6[2]. Here, the beam sensors, in stead of the magnetometer, are used for vehicle detection because the speed and length estimation results can be shared among multiple tasks. Two consecutive speed estimations are averaged to achieve the confidence level required by Pablo.

[2]The Ptolemy II interface used here is only for illustration purpose

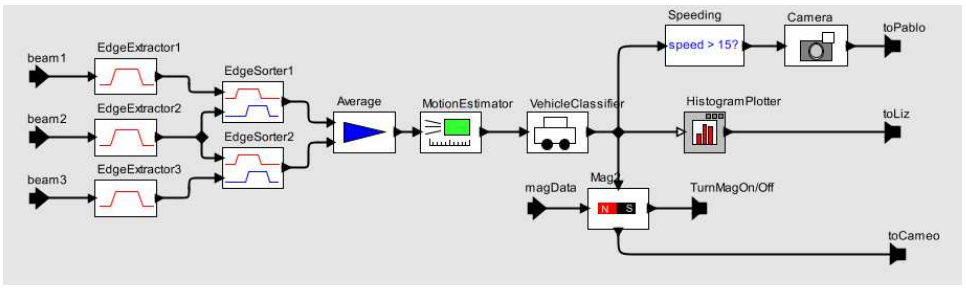

Fig. 6. An illustration of a service composition graph that fulfills all three queries from Liz, Pablo, and Cameo.

In this simple example, there are only two field servers: one connecting to all the motes and the other is the web server interface for the web camera. The service embedding is straightforward, with all the services, except the `triggeredPhotoService`, assigned to the field server.

VI. CONCLUSIONS

Sensor-rich information systems integrate physical information with the digital world. This paper motivates the need for common ontologies that capture the hierarchy and equivalency of information that can be gathered and inferred from sensors. As shown in the parking garage scenario, multiple end users can interact with a sensor-rich information system simultaneously. Using a common semantic model, they can query the system for high-level events without dealing with raw signals. Real-time data are gathered and processed on demand to fulfill all user requests using minimum resources. The semantic model also permits semantic services to be reused, which is particularly important since processing noisy sensor data requires strong domain knowledge.

We presented the SONGS programming model, which uses automatic planning to find the best service compositions from user specified semantic requirements. Service plans are mapped to a subset of network nodes to provide the optimal quality of service. This way, the users do not need to program each indivitual node, but interact with the entire sensor information system as a whole.

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge the contribution of Kamin Whitehouse, Elaine Cheong, Prabal Dutta, and Alec Woo to the semantic service programming model and the parking garage demo.

REFERENCES

[1] *1451.2: A Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*, 1997.

[2] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. G. Gouda, Y. ri Choi, T. Herman, S. S. Kulkarni, U. Arumugam, M. Nesterenko, and M. M. A. Vora, "A line in the sand: a wireless sensor network for target detection, classification, and tracking," *Computer Networks*, vol. 46, no. 5, pp. 605–634, 2004.

[3] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, "Irisnet: An architecture for a world-wide sensor web," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 22–33, 2003. [Online]. Available: http://www.intel-iris.net/papers/pervasive-03.pdf

[4] E. A. Lee, S. Neuendorffer, and M. J. Wirthlin, "Actor-oriented design of embedded hardware and software systems," *Journal of Circuits, Systems, and Computers*, to appear 2003.

[5] J. Liu, E. Cheong, and F. Zhao, "Semantics-based optimization across uncoordinated tasks in networked embedded systems," in *Proc. of the 5th ACM Conference on Embedded Software (EMSOFT 2005), Jersey City, NJ*, Septmber 2005.

[6] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The design of an acquisitional query processor for sensor networks," in *Proc. of the 2003 ACM SIGMOD international conference on Management of Data, San Diego, CA*, June 2003, pp. 491 – 502.

[7] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc. of Intl. Workshop on Wireless Sensor Networks and Applications (WSNA 04), Atlanta, GA*, September 2002.

[8] M. Maroti, G. Simon, A. Ledeczi, and J. Sztipanovits, "Shooter localization in urban terrain," *IEEE Computer*, vol. 37, no. 8, pp. 60–61, 2004.

[9] Object Management Group, "OMG unified modeling language specification (action semantics)," November 2002, oMG Document #ptc/02-01-09.

[10] Open Geospatial Consortium, Inc., *Sensor Model Language (SensorML) for In-situ and Remote Sensors (v1.0.0 beta)*, 2004, doc# 04-019r2.

[11] M. Rahimi, R. Pon, W. Kaiser, G. Sukhatme, D. Estrin, and M. Srivastava, "Adaptive sampling for environmental robotics," in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, April 2004, pp. 3537 – 3544.

[12] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler, "Design and implementation of a sensor network system for vehicle tracking and autonomous interception," in *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN 2005)*, January 2005, pp. 93–107.

[13] K. Whitehouse, F. Zhao, and J. Liu, "Semantic Streams: a framework for declarative queries and automatic data interpretation," Microsoft Research, 1 Microsoft Way, Redmond, WA 98052, Tech. Rep. MSR-TR-2005-45, April 2005.

[14] F. Zhao, J. Shin, and J. Reich, "Information-driven dynamic sensor collaboration for tracking applications," *IEEE Signal Processing*, vol. 19, no. 2, pp. 61–72, March 2002.