Composing Semantic Services in Open Sensor-Rich Environments

Jie Liu and Feng Zhao

Microsoft Research
One Microsoft Way
Redmond, WA 98052
{liuj,zhao}@microsoftc.com

Abstract

Networked sensing promises to drastically change the way people interact with their environments by providing rich contextual information in real time. Major challenges remain on how concurrent users program and control such environments at the application level. This paper summarizes our research efforts in automatically composing semantics services to fulfill declarative user queries in resource efficient ways. We also describe an example software platform, MSR Sense, which supports service abstractions, composition, and execution.

Consider a scenario where Jane is at an airport. She wants to find a restaurant with healthy food and short waiting time on the way to her terminal. The airport can provide such services through a web site that takes Jane's location and flight number information and returns a list of restaurants fulfilling her requirements. For that system, restaurant listings are in the airport's own database, restaurant reviews can be obtained over the Internet through web services, but the crowdedness of the restaurants along Jane's path has to be derived in real time. Depending on the types of the restaurants, the number of customers in them may be estimated by cameras, acoustic sensors, or entrance motion detectors. These sensors may be installed for other purposes, such as security or employee training. But for Jane, they are services that are open to her and she can use them to meet her own goals. Furthermore, while she is using these services, security guards, store employees and other customers can share the same sensing infrastructures with their own quality of service preferences. We call this kind of environments *open*

sensor-rich environments (OSRE), where rich sensor information is accessible to generic users and can be seamlessly integrated with each other.

OSREs are not readily supported by the existing networked embedded system technologies developed by the wireless networking and sensor network research communities. Most sensor networks today are deployed for a single purpose. They are designed for either data collection [8] or a domain-specific application (e.g. [6],[9],[10]). Many research projects assume that the sensors are homogeneous but the network formation is ad hoc. They focus on providing basic network services in the system, such as time synchronization, node localization, topology control, and security. In OSRE, many kinds of sensors may be deployed in the same space, possibly by different organizations. The sensor positions and connectivity may have already been fine tuned to maximize sensing accuracy and communication reliability. The network may have a trivial star topology. Many sensors are plugged in, thus energy may not always be the top concern. However, the applications that use these sensors are not pre-defined. Many users may bring their own preferences, goals, and incentives to the system. They may carry their own devices to enrich the existing network. They may be willing to pay to get good quality of service. Above all, they may not have the expertise to or interests in programming and controlling these networks directly.

Declarative queries have recently gained attention to enable on-demand interaction with sensor systems [2][11]. In the Service-Oriented Network of Generic Sensors (SONGS) project, we study the architecture and techniques for declarative application composition through the notion of semantic services. *Semantics services* are abstractions for asynchronous software components that process data with clear semantic interpretations. These data, called *semantic streams*, are typically derived from sensor data over time, encapsulating physical contextual information. Semantics services process input streams and generate new streams, possibly with different semantic interpretations. They can be

composed into useful applications on demand. Users only need to specify the end results rather than how these results are computed. Declarative queries with automatic service composition can significantly simplify how users interact with OSREs.

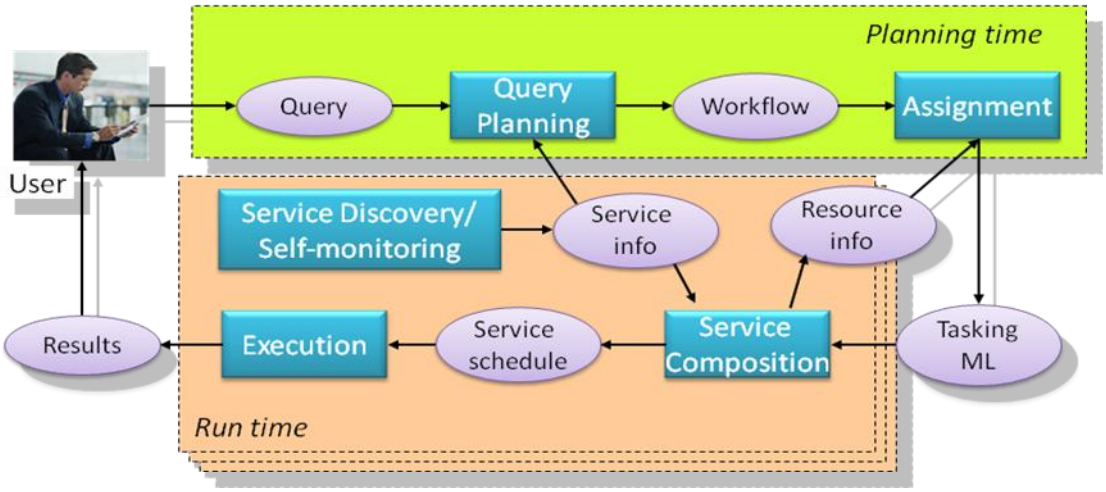

Figure 1. The SONGS architecture.

As shown in Figure 1, the environment provides a set of semantics services that can be discovered by mobile users. A user issues declarative queries that express the goal for the environment to report back or react to. A *query planner* generates a workflow of semantics services that is then assigned to a set of physical nodes in the environment for execution. The assignment process takes into account the service workflow, its resource requirements, and available resources in the network, such as memory, processing speed, network bandwidth, and sensing capabilities. The assignment output is a task graph that assigns each service to a node in the network. A node takes a subset of the workflow (called a tasklet), instantiates corresponding services, and executes them. The execution results are then returned to the user or cached for future queries.

An Application Example

We use an experimental setup in a parking garage to motivate and explain the SONGS architecture, which allows users to run simultaneous queries over real-time sensor data.

The setup is located near the entrance of the second floor (P2) in a parking garage with one-way traffic, shown in Figure 2. The focus of the network is a 4x5 meter area directly in front of an elevator. All vehicles entering this floor of the parking deck pass through this area, as do most pedestrians using the elevator. There are three types of sensors in the system: a network camera, a magnetometer and infrared breakbeam sensors. A breakbeam sensor bounces an infrared beam against a distant reflector. When an object comes between the sensor and the reflector, it detects that the beam has been broken; when the object moves away it detects that the beam has been re-detected. This is the same type of sensor that might be found at a store entrance to detect customers entering and leaving.

Both breakbeam sensors and the magnetometer are connected to micaZ motes[1]. Each of these motes is equipped with a 2.4GHz IEEE 802.15.4 radio. Five infrared breakbeam sensors are placed in a row across the area, 1 meter apart and about 0.5 meter from the ground. The beams are broken in succession by any passing human or vehicle. Each blocking or unblocking event generates an interrupt to the mote. Slightly down the road from the breakbeam sensors, we installed a magnetometer-equipped mote that can detect the changing magnetic field of a moving object. The motes and the camera are connected to the Internet via microservers, which provide libraries of semantic services. Examples of these services are breakbeam objector detector, breakbeam speed detector, car detector, pedestrian detector, metal detector, and object counting.

[1] See http://www.xbow.com

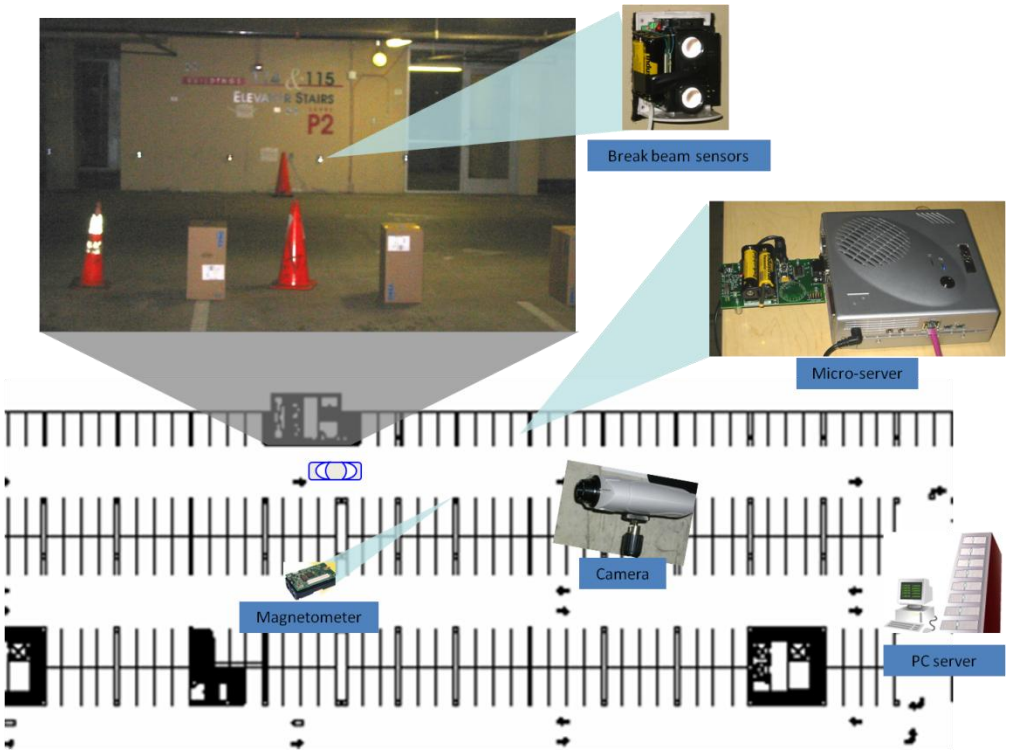

Figure 2. A parking garage sensor network. The breakbeam sensors were laid out in a row on the wall in the focus area. The digital camera was focused on the same area. The magnetometer was placed several meters downstream near the microserver.

Suppose there are three users using the system for three distinct tasks:

- **Task T**. Traffic engineer Todd wants photographs of all vehicles moving faster than 15mph.
- **Task E**. Employee Emma wants to know whether she can find parking spaces in P1 if she arrives at 10AM.
- **Task C**. Corp. security officer Cory wants to collect the magnetic field signature whenever there is a moving object (human or vehicle) passing through the section.

These tasks can be achieved in many ways by the system. For example, there are three ways to detect a vehicle: by using the breakbeam sensor array, the magnetometer, or the camera. But they are not all equivalent: the breakbeam data can be used to estimate the speed of the vehicle, while the magnetometer data cannot be. The camera can be used to recognize the license plate number, while the others cannot be. To detect a vehicle the magnetometer only needs to be sampled at 16Hz, but to make the magnetic field signature useful for Cory, it needs to be sampled at 256Hz.

In the next sections, we show how services are organized in this kind of open sensor-rich environments, how novice users specify their goals, and how the system automatically composes and executes these queries in resource efficient ways.

Semantics Streams and Services

In order to facilitate automatic service composition and assignment, the services need to provide rich meta-information about their input and output data streams as well as their resource requirements.

A semantic stream is a data stream representing a flow of events, each of which encapsulates an observation, such as a vehicle, a person or motion detection. Events have properties. For example, a vehicle event has properties, such as the time or the location it was detected, its speed, direction, and identity. Formally, an event is a tuple of an event type, a timestamp and a set of properties. Together, the tuple defines the semantics of the event.

```
<event>::=<eventType><timestamp>{<property>*}
      <property>::=<propertyName><value>
```

A semantic stream is a sequence of such events with the same type and non-decreasing timestamps:

```
<stream>::=<event>|<event>;<stream>
```

For example, a `VehicleStream` is a sequence of time-stamped events with property `Region` indicating where the vehicle is detected.

A semantic service is a transformation of semantics streams. It takes zero or more input streams and produces one or more output streams.

```
<service>::=<identifier>{<needs>}{<creates>}
            <needs>::={<stream>*}
           <creates>::={<stream>*}
```

The <needs> clauses are the pre-conditions of the services, and the <creates> clauses are the post-conditions. Under this formulation, a sensor is a service with no preconditions. It produces events based

on its sampling rate or event detection logic. For example, in our parking garage, the sensors can provide the following streams:

```
Sensor(BreakbeamMotionDetector,
      Creates(BreakbeamMotionStream(P2_Entrance)))
Sensor(Camera,
      Creates(VideoStream(P2_Entrance)))
Sensor(Magnetometer,
      Creates(MagneticFieldStream(P2_Entrance)))
```

where, `P2_Entrance` is a location constant in the system.

There can be multiple ways to generate the same semantics stream. For example, to detect vehicles at the P2 entrance, one service can use the motion detector:

```
Service(MotionVehicleDetector,
      Needs(BreakbeamMotionStream(region)),
      Creates(VehicleStream(region)))
```

where, `region` is a parameterization of the streams. Its presence on both <needs> and <creates> clauses indicates that the output streams inherit the `region` property from the input streams. An alternative vehicle detection service can use the cameras:

```
Service(CameraVehicleDetector,
      Needs(VideoStream(region)),
      Creates(vehicleStream(region)))
```

Yet another service can use the `MagneticFieldStream`.

In this framework, a user query can be specified as an output stream with particular properties as the goal. For example, assuming people usually prefer parking at P1 before going to P2, Emma's Task E can be expressed as:

```
Stream(VehicleStream,
      property(timestamp, >, 8:00AM),
      property(timestamp, <, 10:00AM),
      property(region, =, P2_Entrance)).
```

Declarative Service Composition

Once the sensors and services of a particular OSRE are defined, our system responds to queries by automatically selecting sensors and composing services using a variant of the backward chaining algorithm.

We treat each known semantic stream as a *fact*, and each semantic service as an *inference rule* that can derive output semantic streams from input streams. Using backward chaining, all facts the system has are stored in a Knowledge Base (KB), which initially contains all physical sensors. Each unproven predicate of the query is matched with consequences of other rules or facts in the KB. If it matches with a rule, the antecedents of the rule must be proved by matching with another rule or a fact. The backward chaining terminates when all antecedents have been matched with facts. Otherwise it fails after an exhaustive search of all rules. The sensors and services used to prove the query constitute the inference graph that will provide the desired semantic values specified in the query as the output. The inference graph is what the system executes.

Multi-query optimization

When multiple queries exist in the system at the same time, there are opportunities to optimize across them by reusing partial results. In the parking garage example, suppose that the task T is running while the task E arrives. Noticing that there are several options to achieve vehicle detections to answer E, and that T is already using the breakbeam sensors for estimating vehicle speeds, we can build a joint workflow as shown in Figure 3.

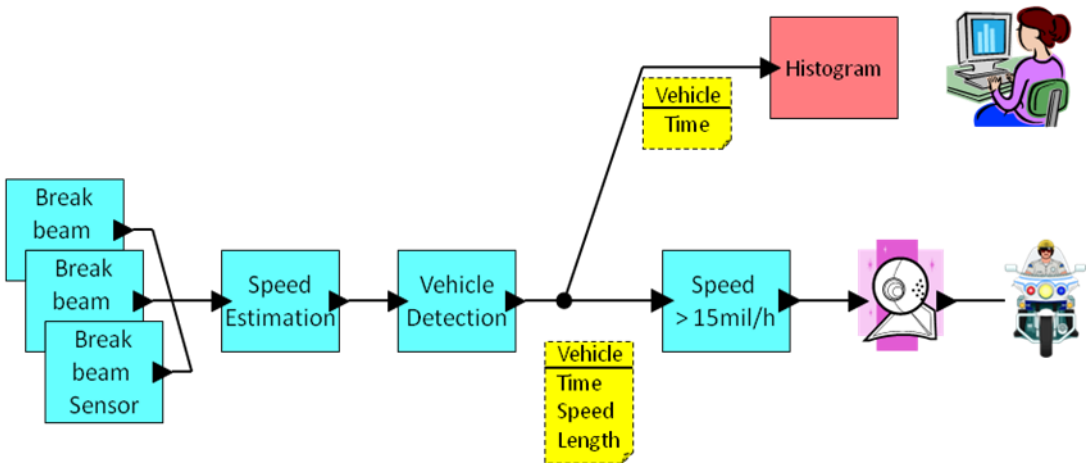

Figure 3.Sharing vehicle detection events across two queries.

The SONGS query composition engine improves upon typical backward chaining inferences to achieve minimum intermediate facts. It achieves this by instantiating a virtual representation of each service in the KB the first time it is used in the proof. Subsequent predicates are proved by matching against all existing virtual instantiations before creating new inference units. For instance, in the example task T above, the composition engine matches the first predicate to the `VehicleSpeedService`, as did standard backward chaining, but this time it creates a virtual instance of `VehicleStream` with properties Time, Speed, and Length and assigns a unique ID to the event stream. When task E arrives, the inference engine does a breadth-first search to find a fact within the KB that can answer the query, before creating new rules.

Backward chaining is supported in most logic programming languages such as Prolog. The unique challenge of applying the technique to sensor data is to handle physical properties, such as sampling rates, event detection accuracies, and locations. Many of these constraints are defined on the set of real numberds, which is not part of typical logic programming paradigms. In SONGS, we used CLP-R (constraint logic programming – Reals) [11], which can handle constraints with real numbers.

Signal Type System

While CLP-R is powerful and flexible to handle real-numbered constraints, they are computationally expensive. In practice, a large class of physical properties has to do with the sequencing of events. We designed a signal-type system, which is much faster to resolve [7]. A signal-type system is like a class type system in object-oriented designs, but applyed to event streams (aka signals). In particular, for events of the same EventType, we can build a lattice that describes their timing properties, as shown in Figure 4.

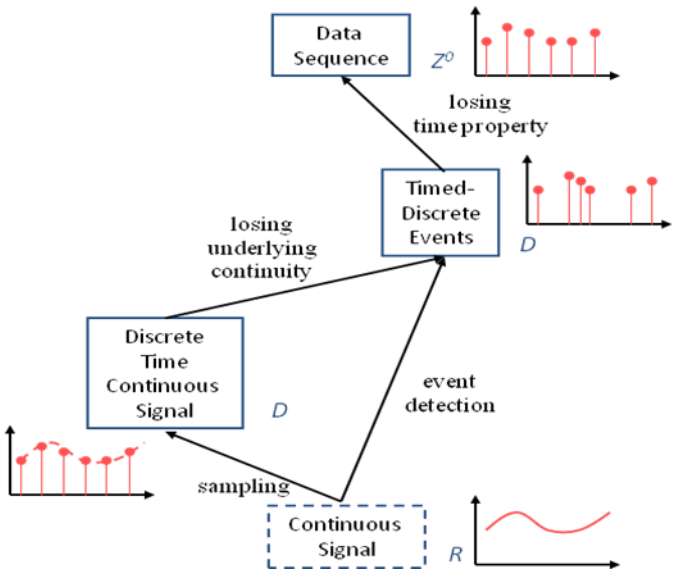

Figure 4. A signal type system of continuous and discrete signals.

We classify signals based on their timestamp domains. Take the magnetic field stream as an example. Physically, the strength of a magnetic field at a location is a continuous signal. By sampling that signal with a particular interval, we can obtain a discrete time representation of the continuous signal (DTCS), or a sampled signal. The DTCS class of signals can further be clustered by their sampling periods. Furthermore, by losing the underlying continuity, a DTCS signal can be viewed as a sequence of timed discrete events. For example, we can view a set of magnetic field samples as measurements with timestamps. Furthermore, if we ignore the timestamps completely, then we have a sequence of

untimed measurements. This most abstract view tolerates asynchrony in the system, such as clock skewing among nodes.

A semantic service specifies the most generic signal type it can accept and the most specific signal type it produces. A signal-type system can reason about the input-output signal compatibility based on the type lattice and can automatically insert type converters when necessary. For example, Figure 5 shows the automatic signal type conversion between the 256Hz sampling and the 16Hz sampling of the magnetometer when tasks C and E are planned at the same time.

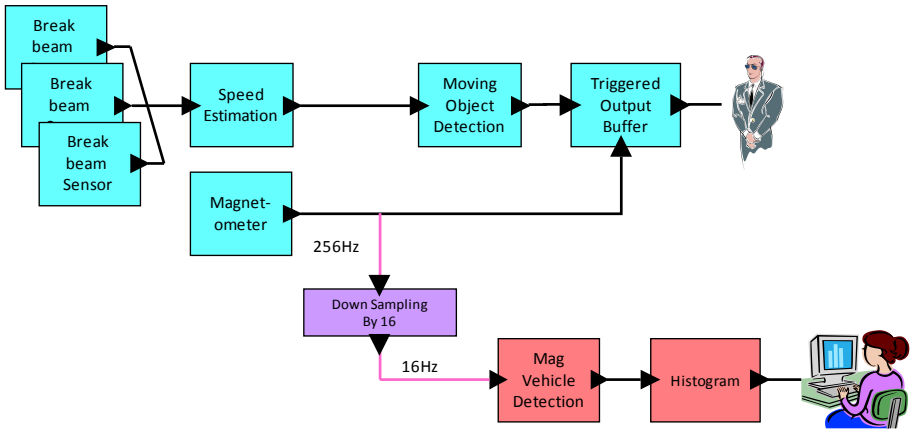

Figure 5. MagneticVehicleDetection service need a 16Hz input, a downsampling service can be added directly to convert a 256Hz magnetic field sampling used by task T.

Service Assignments

Once a service workflow is generated by the query planner, services need to be assembled across multiple devices for execution. Since multiple services may co-exist on a single node and multiple nodes may provide semantically equivalent services, it is important to compose the set of services to achieve optimal end-to-end performance, in terms of shortest latency or minimum energy consumption, for example.

In SONGS, each device describes its resource capacity when advertising its services, such as power availability (battery or line powered), processing modes (various voltage/frequency scaling modes),

network connectivity, and bandwidths. The service assignment component in SONGS assigns each service to a device subject to capacity and performance constraints.

The service *assignment problem*: Given a workflow of services, as a directed graph $W = (S, E)$, where S are services and $E \subseteq S \times S$ are dependencies among services; given a network of devices $N = (D(m), L)$, where D are devices, parameterized by their modes *m*, and L are network links among the devices; and given resource requirement functions $P : S \times D(m) \to R^+$ (computing cost) and $C : E \times L \to R^+$ (communication cost), find an optimal assignment $A : S \to D(m)$, such that

$$J = \sum_{s \in S, d(m) = A(s)} p(s, d_m) + \sum_{e=(s_1, s_2), l=(A(s_1), A(s_2))} c(e, l)$$

is minimized.

The service assignment problem is NP-hard in general, but it can be formulated into an integer linear programming (ILP) problem when the devices operate at discrete modes. The following constraints must be considered in the formulation of the ILP problem:

- Each task is allocated to a single device at a single mode;
- There is a cost of executing each service on a given device under a given mode;
- There is a cost of communicating a piece of data between two devices over a particular link;
- Once a device is put into sleep, there is a time and corresponding energy cost to wake-up the device;
- Invocations of services have dependency relations defined by the service workflow;
- Communication can only take place after corresponding data are generated by the services;
- There may be end-to-end quality of service constraints such as the maximum total latency specified by the user query;
- There may be predetermined assignments, if a service is only provided by some devices.

A full treatment of the problem can be found in [3].

Although the general service assignment problem is NP-hard, a degenerated case, where the service workflow has a tree structure, is simpler. Service trees cover a large class of sensor fusion applications,

where the information is always condensed through the workflow. In such cases, the optimal assignment can be found through dynamic programming. In [1], a modified greedy assignment strategy is derived that merges services along the tree until we achieve ¼ reductions in data rate. It can be shown that the modified greedy algorithm can achieve a sub-optimal total cost that is a constant factor (x8) worse than the optimal assignment, independent of the tree size. This bound is usually much tighter in reality. For example, Figure 6 compares the total costs normalized over the optimal cost (min = 1 in the figure) among three service tree assignment algorithms: the modified greedy algorithm designed in [1], the unmodified greedy algorithm, and an in-network relaxation algorithm where services locally adjust their assigned locations. The service tree is the Task C as in Figure 3. Each algorithm is tested on 20 networks with 64 nodes in perturbed grid topologies or random topologies. We can see that the modified greedy algorithm performs very well, closely matching the optimal costs.

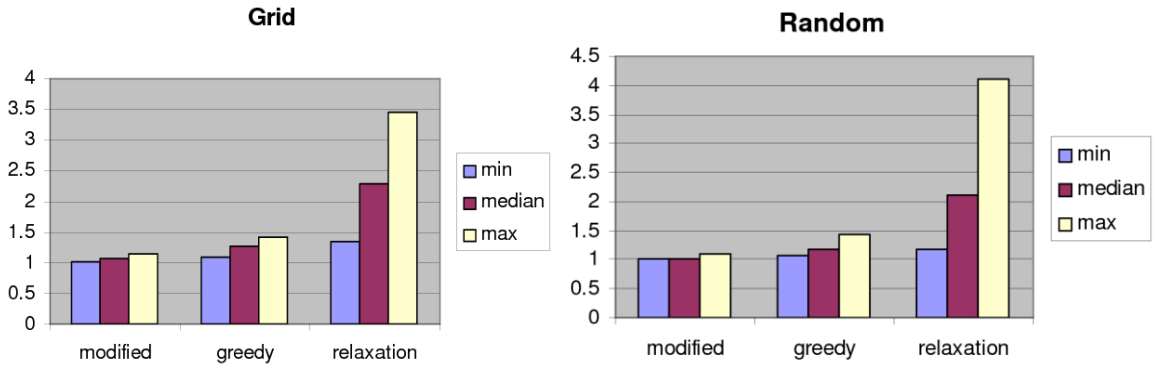

Figure 6. Service tree assignment result comparisons over 20 networks of 64 nodes with grid topologies or random topologies.

A Service Composition Platform

We have experimented with service-oriented architectures in several hardware and software platforms, from 16-bit MCU-based embedded systems to PC-class microservers. MSR Sense is a software platform for microservers in a tiered architecture for sensor networks. Unlike in Tenet [5], we assume microservers manage and reflect heterogeneous sensor tier capabilities. Thus, depending on which

sensors are within its range and contact, a microserver stores a set of services reflecting the sensors under its control. The MicroServer Execution Environment (μSEE) is a service-oriented runtime environment in MSR Sense, implemented on the .NET framework. It is designed to facilitate loosely couple service instantiation and execution, enabling runtime service sharing among uncoordinated queries. Service compositions are sent to microservers as tasklets, specified in MicroServer Tasking Markup Language (MSTML). The microservers dynamically create services from their libraries to form executable tasklets.

In μSEE, services are implemented as threads. There is no global state in a service composition. The communications between services in a single device are mediated by publish-subscribe data channels implemented using .NET event delegates. Communication between devices goes through the IP network. All outputs from services go into an event mediator and are visible to all other services subscribing to the events. Every subscription service receives a copy of that event. Events can be further cached in database to answer future queries. After an event is delivered to every subscriber, it is garbage collected from the memory. Since the mediators separate the publishers and subscribers, a service does not need to specify where its inputs come from or where its outputs go to. Thus, intermediate computation results can be easily shared across tasks.

The MSR Sense platform is released in *Microsoft Research Sensor Network Academic Resource Kit 2007*. It has been used in various sensor network applications, including fast prototyping a sensor network deployed in data centers to monitor cooling performance [3].

Conclusion

This paper summarizes main research results from the SONGS project that enables users to issue declarative queries to interact with OSREs. We show that with rich service interfaces, such as data semantics, signal types, and resource requirements, declarative queries can be automatically compiled

into composition of semantics services that are then assigned to the sensor network nodes for resource-efficient execution. To make this architecture pervasive, further challenges remain in building standard ontology or ontology transformations that can incorporate services built by multiple organizations.

Acknowledgement

The authors thank Zoe Abrams, Elaine Cheong, David Chu, Prabal Dutta, Slobodan Matic, Ryan Newton, and Kamin Whitehouse, who made significant contributions to the SONGS project during their internship with Microsoft Research at various times between 2004 and 2007.

References

[1] Zoe Abrams and Jie Liu, “Greedy is Good: On Service Tree Placement for In-Network Stream Processing.” The 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006), Lisboa, Portugal, July 4-7, 2006.

[2] David Chu, Lucian Popa, Arsalan Tavakoli, Joseph Hellerstein, Philip Levis, Scott Shenker, Ion Stoica, “The Design and Implementation of A Declarative Sensor Network System,” The 5th ACM Conference on Embedded networked Sensor Systems (SenSys 2007), Sydney, Australia, 6-9 Nov, 2007

[3] David Chu, Feng Zhao, Jie Liu, and Michel Gorczko, “Que: A Sensor Network Rapid Prototyping Tool With Application Experiences From A Data Center Deployment,” The 5th European Conference on Wireless Sensor Networks (EWSN 2008), Bologna, Italy, 30 Jan-1 Feb, 2008.

[4] Michel Goraczko, Jie Liu, Dimitrios Lymberopoulos, Slobodan Matic, Bodhi Priyantha, Feng Zhao, “Energy Optimal Software Partitioning in Heterogeneous Multiprocessor Embedded Systems,” The 45th Design Automation Conference (DAC 2008), Anaheim, CA, June 2008.

[5] Omprakash Gnawali, Ben Greenstein, Ki-Young Jang, August Joki, Jeongyeup Paek, Marcos Vieira, Deborah Estrin, Ramesh Govindan, Eddie Kohler, “The TENET Architecture for Tiered Sensor Networks,” in Proceedings of the ACM Conference on Embedded Networked Sensor Systems (Sensys), Boulder, Colorado, November 2006.

[6] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, B. Krogh, “An Energy-Efficient Surveillance System for Sensor Networks,” The 2nd Intl. Conference on Mobile Systems, Applications, and Services (MobiSys 2004), Boston, MA, June 2004.

[7] Jie Liu, Elaine Cheong, and Feng Zhao, "Semantics-Based Optimization Across Uncoordinated Tasks in Networked Embedded Systems," The 5th ACM Conference on Embedded Software (EMSOFT 2005), Jersey City, New Jersey, September 18-22, 2005.

[8] 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.

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

[10] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh, “Fidelity and Yield in a Volcano Monitoring Sensor Network.” In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006), Seattle, November 2006.

[11] Kamin Whitehouse, Feng Zhao, and Jie Liu, "Semantic Streams: A Framework for Composable Semantic Interpretation of Sensor Data ". The Third European Workshop on Wireless Sensor Networks (EWSN), Springer-Verlag Lecture Notes in Computer Science. Zurich, Switzerland. February 13-15, 2006.