Technical Report MSR-TR-2005-28, February, 2005

Towards Service-Oriented Networked Embedded Computing

Jie Liu and **Feng Zhao**
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
{liuj, zhao}@microsoft.com

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 complains from people that there are several cars driving extremely fast in certain areas 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 a generic wireless sensor network recently installed in the garage and augment it with special purpose sensors as needed.

Systems like these are not readily supported by the current networked embedded system technologies developed by the sensor network research community. Today, sensor network applications are designed using primarily one of the two philosophies: the dada-collection view and the application-specific view. In the data-collection view (see the various sensor database projects [Cougar; tinyDB; GDI]), a sensor network is regarded as data collection fibers where sensor data are streamed to servers. The data can be either aggregated through routing or processed centrally. Although advances have been made on running user queries within the network, the architecture is ill-suited for monitoring and tracking sparse events in a resource-constrained environment. The application-specific view [Shooter] acknowledges the fact that many sensor network system behaviors depend heavily on the physical stimuli, and tries to make the best use of the power and bandwidth resources through exploiting the application-specific dynamics. For example, a microphone-based vehicle tracking system would be designed quite differently from a camera-based system, although many of the system components are similar. Everything is optimized at the design time. The system is rigid and hard to change afterwards.

These two philosophies can be viewed as two extremes for handling application logics. In the data collection view, there is no application logic in the network; everything is processed off line. It is quite generic but not always resource optimal. In the application specific view, the application logic is hard wired into the network. The designers have fine grained resource control but the system is rigid and hard to reuse. We motivate here a service-oriented architecture for networked embedded computing, SONGS[1], where the system is open, retaskable, and resource-aware, a “happy median” between the data-collection and application-specific views.

As illustrated in the CEDAR building scenario, we envision that a large-scale networked embedded system is likely to be deployed by

[1] SONGS stands for Service Oriented Networked proGramming of Sensors.

multiple venders incrementally. A system may consist of both mobile and stationary nodes with varying capabilities. It must be integrated seamlessly into the Internet and must provide intuitive interfaces for remote user interactions. There will be multiple, concurrent users exercising different functionalities of the system for different purposes. The system is self-monitored and resource-aware. It has a certain level of autonomy to decide on the best use of available resources to fulfill multiple users' uncoordinated requests.

SONGS: A Service-Oriented Architecture

Service orientation as a programming paradigm emerged from distributed enterprise computing technologies such as CORBA [CORBA] and Jini. It has been adopted at a larger scale through the notion of "web services" [WS]. Services are loosely coupled, self-describing, and typically stateless software units that are interoperable through message-based communication models.

Services are identified by their interfaces. Users typically program against these interfaces without caring about where the particular services execute (a property known as *location transparency*). An underlying infrastructure, often called a *framework* or a *service bus*, may find the best implementation through service discovery and dynamic binding. Thus, a service-based architecture contains substantial infrastructure supports in addition to the services themselves. Typical supports include service description languages that allow platform independent description of the service interfaces, service registration mechanisms that allow services advertise themselves once deployed, service discovery mechanisms that allow user applications to find and choose appropriate services at run time, and authorization, authentication and accounting mechanisms to establish trusts among service providers and service users. To achieve dynamic composability and sharing among multiple applications, services are generally *stateless* (a.k.a. *functional*). That is, services do not store application state. It is up to the service integrator to pass the state with service requests.

Services in Networked Embedded Computing

In the context of networked embedded computing, we adopt the concept of services primarily for their interoperability, scalability, and retaskability. Services are open and self-descriptive. This allows systems from different venders to work together. Services are not necessarily tied to particular nodes. For example, a car detection service can be implemented using acoustic, magnetic, loop inductor, break beam, or image sensing. In a real system, there is typically a great deal of redundant information available to the users. Hiding this information detail behind the service interfaces makes interactions with physical phenomena scalable to the number of sensor nodes. In a service-oriented architecture, services are reusable and often generic. Applications are composed after the services are deployed, thus the whole system is retaskable. The scarce resources can be shared among multiple concurrent tasks.

Using this model, the network of sensors in the CEDAR building may expose a set of services, for example, human detection, vehicle detection, speed calculation, length calculation, certain chemical element detections, picture taking, etc. Then, the three different users can send their own queries independently. These queries are processed by composing a subset of these services on demand. When all queries are running, parts of the query processing services can be shared, as shown in Figure 1.

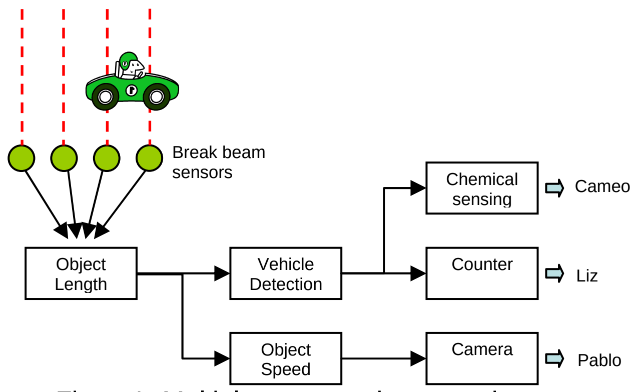


Figure1. Multiple user queries may share common services.

Services in networked embedded systems, in addition to sharing general properties of web services, have their own distinct characteristics. Many of these characteristics are derived from the resource constraints in the systems and their direct interactions with the noisy physical world.

Physicality of services: Many networked embedded services tie closely to the layout of the sensor nodes, their sensing modality, and their physical surrounding, thus they cannot be totally location transparent. For example, a car tracking service is only valid when a car has been detected in that neighborhood, and the service may migrate when the car moves. It is also hard to replicate these services without sacrificing timing properties, since sensor data are only available at particular locations and particular time. Shipping raw data over the network may not be the best use of resources. This gives load balancing a whole new meaning when optimizing on multiple user queries.

Stateful services: The dependency on physical input also makes it meaningless to have completely stateless services. Why would an anomaly detection service be useful if all the required detection states are fed by the users? To further develop this notion, we distinguish query-dependent state from query-independent state. A query-dependent state is the information that is specific to a user's task. For example, suppose the query is to count the number of passing-by cars from 8AM to 11AM. Then, the counting result up to the current time is a query specific state. The existence of the vehicles, on the other hand, is query independent. Services with only query independent state may be shared among queries, while those with query-dependent state may be hard to be shared.

Quality of service: The interaction with the noisy and uncertain physical world also brings new meanings for the term "quality of service." In sensing physical phenomena, the quality is usually defined by the information quality in terms of, e.g., signal-to-noise ratio, belief state, or entropy. In general, one might have alternative services with different qualities. Depending on the quality of answers expected by the user, a service framework may choose the services of sufficient quality while considering response time and resource usage.

Nature of events: Data in a networked embedded system may be periodic, as in streaming data, or aperiodic and infrequent, as in detection events. The triggers for the events may be external physical phenomena, user queries, or system requests. Unlike in the more resourceful wired environment, web services for processing unpredictable, sporadic events in a resource-constrained networked embedded system cannot afford to run all the time, and must be extremely stringent about where and when to invoke a service. For an infrequently run service, there is also the need to appropriately save intermediate states between the processing episodes.

Service-Oriented Query Processing

Due to resource constraints in networked embedded systems, there is an advantage of processing the information within the network rather than sending raw bits to the edge of the network [CSIP; tinyDB]. Using a service-oriented architecture, in-network information processing becomes natural. Services can easily encapsulate operators, such as sensing and signal processing algorithms, over streams of sensor data or sporadic events.

Figure 2 illustrates a way that networked embedded services can be used by end users. There are infrastructural supports at both schedule time and run time.

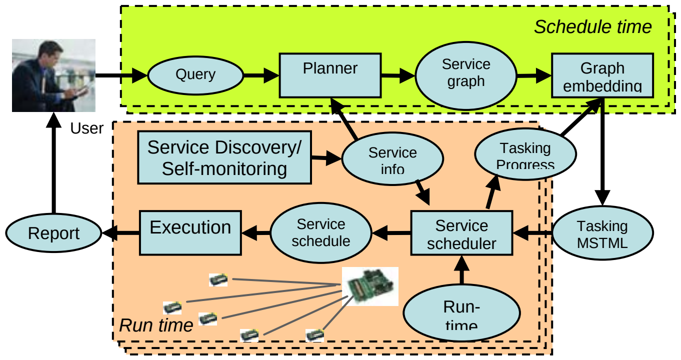


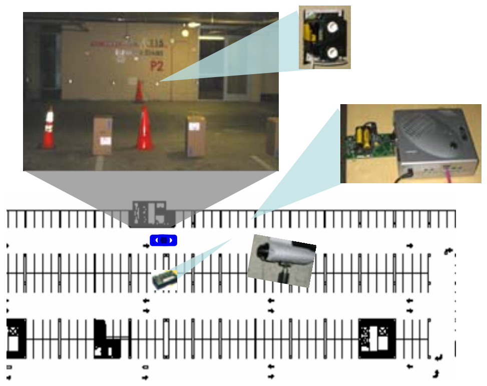
Figure 2. A way networked embedded services are used by users.

As part of the runtime system, a networked embedded system self-monitors its operating conditions, such as network connectivity, resource availability, as well as preconditions for registered services. This information is made available to the schedule time query planner. When a user issues a query, it is first decomposed by the planner into a set of services forming a *service graph* or workflow. The logical graph is then embedded onto the physical sensor nodes, resulting in an assignment of logical operations to distributed processors. For an optimal embedding of the graph, the assignment must take into account node locations, sensing modalities, network topology, service availability, and resource constraints. Since the system configuration may change during the course of executing a long-running query, the result may only serve as an initial embedding subject to run-time adaptation.

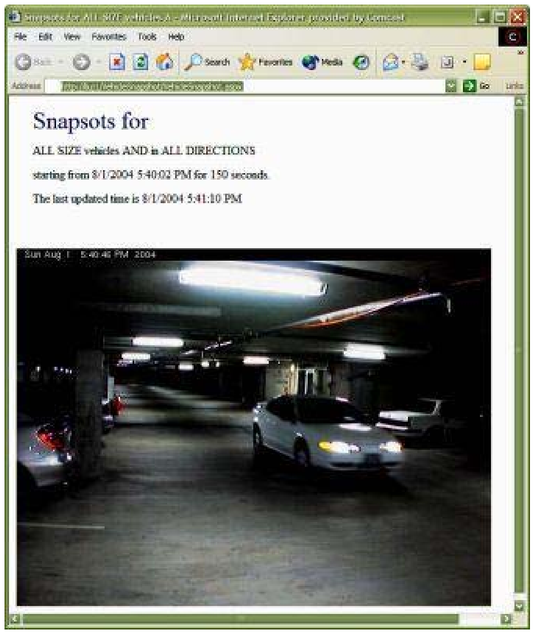
The task assignment is then injected into the network in the form of a tasking description language – the micro-server tasking markup language, or MSTML. This description is accepted by a service scheduler running on each node. In order to control memory consumption, not all services are preloaded. They 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. After the services are instantiated, the service execution engine executes and monitors the tasks across multiple nodes. When resources change in the network, some services may be migrated to other nodes. If the execution engine cannot determine alternate task composition locally, the schedule time planner may be invoked again. When tasks terminate, the execution engine is also responsible to clean up parts of the task that is not shared by other tasks.

A Proof-of-Concept Testbed

We have built a testbed to study the SONGS architecture. An initial prototype includes two microserver nodes, six sensor motes and a web camera deployed in a parking garage. Figure 2(a) shows the layout of an experiment. Five Crossbow MicaZ motes are places in a row, each attached to a break beam sensor. The break beam sensor emits an infrared beam which bounces back from a reflector on the opposite side of the lane. When a car drives by this section of the lane, it blocks the infrared beam one by one. By correlating the sequence of break beam signals, one can detect vehicles and estimate their lengths and speeds. A magnetometer is placed further down the road. When receiving a trigger, it can collect and transmit magnetometer readings to a microserver. Both the five break beam motes and the magnetometer motes communicate wirelessly with a microserver. A camera, connected to another microserver, is also placed near the magnetometer. When receiving a trigger, it can take a picture and send it to the microserver.



(a)



Snapsots for

ALL SIZE vehicles AND in ALL DIRECTIONS

starting from 8/1/2004 5:40:02 PM for 150 seconds.

The last updated time is 8/1/2004 5:41:10 PM

(b)

Figure 3. A garage deployment of a sensor network testbed, and a GUI showing a successful query of the network.

The system is used to answer three user queries such as the ones shown in Figure 1, with the magnetometer replacing the chemical sensor (see Figure 2(b) for an example).

Research Challenges

The service-oriented architecture and the proof-of-concept testbed motivate us to look at some important research issues.

Service Interfaces

A service-oriented platform for networked embedded computing is essentially a layer of abstraction, which, to application users, supports application programming and user interaction, while, to system developers, encapsulates essential system functionalities such as locations, timing, sensing, signal processing, data storage and retrieval, and routing. How good that platform is must be judged by how well it can support both two kinds of users.

Traditional software service descriptions, e.g. the web-service description language (WSDL), only capture the invocation mechanisms of the services – their address/port, name, and input-output argument types. In networked embedded systems, services need richer interfaces to abstract the non-functional aspects and to facilitate resource-aware execution. In addition to general service invocation information, this richer set of interfaces may include resource requirements, QoS metric (which can be a function of resource usage), configuration parameters, input/output data semantics, etc. We need to investigate what the minimum set of these descriptions for compile-time scheduling and runtime adaptation is. At the same time, there need to be systematic ways in which this information can be assigned to the services, either at compile time or by learning the environment after the services are deployed.

Service Framework

Services can be instantiated either by users' query or in response to physical triggers. They can reside on fixed nodes, migrate among nodes, or collaboratively run on multiple nodes. These make service registration and discovery quite tricky, especially when there is no central registration point for an *ad hoc* or unreliable system deployment. Multi-tier architectures may help shield away local service dynamics from high-level query planners. In these architectures, high-level nodes become proxies for low-level nodes, and service descriptions are aggregated spatially and temporally to improve scalability. We also expect distributed data structures such as distributed hash table [DHT] to play an important role in service registration and discovery.

One of the key components in a resource-aware execution environment is the capability of optimizing resource uses by reducing duplicate service instantiations. This can be done syntactically by comparing the service composition graphs, or based on data semantics. For example, if there is already a car detection service running, a new task that needs car detection event can directly subscribes to the existing detection service without reinitiating another car detection process. This problem gets more interesting when taking quality of information, cost for producing data, and the nature of events into consideration. Take sampling as an example, if one task requires sampling at 20Hz and another requires sampling the same sensor at 50Hz, a 100Hz sampling can satisfy both requirements, but 40% of the samples are useless. On the other hand, given the continuity of the underlying signal, the 20Hz stream can be constructed from the 50Hz stream with limited error. Many examples like this exist, motivating us to take advantage of the physicality of real world data.

A resource-aware service execution framework needs to monitor itself and to deal with uncertainties in the system. This is particularly challenging given unreliable wireless links, low power devices, and sometimes harsh physical environment. Fault tolerance and load balancing under these uncertainties becomes extremely important for keeping the system functioning.

Application Programming Model

There are many ways to provide a programming interface in SONGS. A straightforward approach is to design imperative or visual languages that directly address services and their composition. However, since networked embedded computing programs are usually tied into specific application domains, it is more user-friendly and powerful to provide high-level domain-specific languages.

Service-oriented architectures are desirable for application programmer and end users to program and interact with the entire network as a whole rather than programming each individual node. They are not intended to address tightly-coupled node level interactions such as network protocols. An ideal programming interface for these architectures then should allow programmers to directly specify system behaviors in terms of their domain knowledge, such as physical phenomena of interests. For example, a collaboration group is an abstraction of nodes that share common application states [State]. Given that the program does not address individual nodes, it is possible for compilers and run-time systems to preserve the specified behavior against the uncertainties in system topology, signal noise, and node capabilities. Since physical phenomena will be first class citizens in these programming models, the notion of space, time, sampling, and information quality must be rigorously introduced into the languages. Extensions to spreadsheet programming such as Excel, functional/reactive programming such as Regiment [Regiment], or logic programming such as Datalog [Datalog] are particularly promising.

Conclusion

We have developed a service-oriented architecture SONGS for open, retaskable, and scalable networked embedded computing. The service model simplifies user programming and interaction with such complex systems, and enables resource sharing among uncoordinated concurrent user queries. We have also presented a prototype implementation of the service model for a parking garage monitoring application.

Acknowledgement

We wish to thank Elaine Cheong, Prabal Dutta, and Kamin Whitehouse for their help in the development of the prototype system and testbed, and Jeremy Elson and Alec Woo for discussions.

References

[tinyDB] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks." *Proc. 5th Symp. Operating Systems Design and Implementation*, December 2002.

[Cougar] P. Bonnet, J. E. Gehrke, and P. Seshadri. "Querying the Physical World." *IEEE Personal Communications*, Vol. 7, No. 5, October 2000, pages 10-15.

[GDI] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, J. Anderson. "Wireless Sensor Networks for Habitat Monitoring." *First ACM Workshop on Wireless Sensor Networks and Applications*. Atlanta, 2002.

[Shooter] M. Maroti, G. Simon, A. Ledeczi, J. Sztipanovits. "Shooter Localization in Urban Terrain." *IEEE Computer*, August 2004.

[CORBA] Object Management Group. The common object request broker: Architecture and specification. Technical report, Object Management Group, June 1999.

[WS] Web Services Description Working Group. "Web Services Description Language (WSDL) Version 2.0: Primer." W3C working draft, December 2004.

[CSIP] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich, "Collaborative Signal and Information Processing: An Information Directed Approach." *Proceedings of the IEEE*, 91(8):1199-1209, 2003.

[Chord] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A scalable peer-to-peer lookup service for Internet applications." *Proc. ACM SIGCOMM '01 Conference*, San Diego, Aug. 2001.

[State] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao, "State-Centric Programming for Sensor and Actuator Network Systems." *IEEE Pervasive Computing*, 2(4):50-62, 2003.

[Regiment] R. Newton and M. Welsh. "Region Streams: Functional Macroprogramming for Sensor Networks." *Proceedings of the First International Workshop on Data Management for Sensor Networks (DMSN)*, Toronto, Canada, August 2004.

[Datalog] R. Ramakrishman and J. D. Ullman. "A Survey of Deductive Database Systems." *J. Logic Programming*, 23(2):125-150, 1995.