Service-oriented Architectures for Networked Embedded Sensor Systems

Xenofon Koutsoukos, Manish Kushwaha, Isaac Amundson, Sandeep Neema, and Janos Sztipanovits

Institute for Software Integrated Systems
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235, USA
Xenofon.Koutsoukos@vanderbilt.edu

Extended Abstract

1 Introduction

Wireless sensor networks (WSNs) are inherently dynamic in nature due to node mobility, limited resources, and unreliable communication links. Addressing the problem of dynamic network behavior places a substantial burden on the programmer. In addition to an application’s core functionality, it is necessary to implement low-level operations such as robust communication protocols, resource management algorithms, and fault tolerance mechanisms. In addition, unlike traditional network programming, WSN programming requires a coupling between application-level and system-level logic that creates significant challenges and increases the risk of deploying incorrect code.

We have developed OASiS, an Object-centric, Ambient-aware, Service-oriented Sensornet programming framework and middleware which enables the development of WSN applications without having to deal with the complexity and unpredictability of low-level system and network issues. The framework provides a well-defined model of computation based on globally asynchronous locally synchronous dataflow [1], and is complemented by a user-friendly modeling environment. OASiS decomposes specified application behavior and generates the appropriate node-level code for deployment onto the sensor network.

OASiS takes a service-oriented approach to behavioral decomposition and application execution. Each activity (sensing, aggregation, service discovery, etc.) is implemented as a separate service. The advantages of using a service-oriented architecture for WSN applications are similar to those of Web services. Services are modular, autonomous, and have well-defined interfaces which allow them to be described, published, discovered, and invoked over a network. These properties permit services to be dynamically composed into complete applications. Because many WSN applications will consist of multiple services on multiple nodes, planning, scheduling, and service discovery mechanisms are built into the framework middleware.
In object-centric programming, an object is a logical element which represents some physical phenomenon being monitored by the network. The application is driven by the object, and its behavior is governed by the object’s current state. The programmer is able to specify this behavior from the viewpoint of the object, and need not worry about how the behavior is implemented at the node level.

In traditional application programming, communication failures are handled on an individual basis as exceptional events. In WSN programming, communication failures are not exceptions, but normal operational behavior, and must therefore be treated as such. Ambient-aware programming [2] has emerged as a paradigm for mobile computing and provides several characteristics which are useful for robust sensor network applications. Each node in an ambient-aware sensor network always has up-to-date knowledge of its immediate neighborhood.

A significant benefit of the service oriented architecture is that it enables the efficient integration of heterogeneous sensor networks to powerful and robust sensor webs. A sensor web is a heterogeneous collection of sensor networks that may include small inexpensive computing devices as well as resource-intensive sensing platforms such as satellite imaging systems, meteorological stations, air quality stations, and security cameras. Service-oriented architectures have already proved successful on the World Wide Web, however Web service technologies have been developed assuming standard Internet protocols and are not realizable in resource-constrained sensor networks. Our framework allows the efficient integration of OASiS with Web services and offers the ability to seamlessly assemble a sensor web from various sensor networks which can greatly benefit applications ranging from emergency response to homeland security.

We have implemented a suite of middleware services which support OASiS. The services include components for managing internal sensor node operations, communication, service discovery, and object maintenance. The OASiS programming framework can be used to develop any type of dataflow application such as vehicle tracking, fire detection, and distributed gesture recognition. As proof of concept, we have developed a simplified indoor tracking experiment, which monitors a heat source as it travels through the sensor network. The application employs many resource-constrained sensor nodes, but it can also access Web services in resource-intensive nodes. Our work demonstrates the feasibility and utility of a service-oriented WSN programming model.

2 OASiS Programming Framework

At present, users wishing to deploy WSN applications must be adept at developing the sensor network middleware, the domain-specific functionality, and perhaps even an interactive front-end. Application development will benefit from a programming paradigm that provides separation of concerns (SoC). OASiS is a programming framework that facilitates SoCs for application development through a multilayer development process. Figure 1 illustrates the relationship between each development stage in OASiS.
In OASiS, core sensor network functionalities are bundled as *middleware services* including service discovery, service graph composition, failure detection, node management, and others. The *domain services* development layer provides domain-specific service libraries written by the domain experts, which then can be used by the application developer. OASiS then wires the domain services onto the middleware to produce node-level executable code for deployment on the network. *Application development* in OASiS does not require any expertise in sensor network programming. Instead, complete applications are developed using model-integrated computing techniques [3].

In addition to providing multilayer development, the programming framework addresses many design challenges for networked embedded sensor systems:

- Our ambient-aware middleware supports dynamic service discovery and configuration to address changes in network topology due to failures and unreliable communication links.
- Heterogeneity is addressed by employing well-defined services on *heterogeneous* platforms that can be composed together in a seamless manner. For example, resource-intensive Web services can easily be plugged into OASiS applications and are treated as ordinary WSN domain services.
- OASiS supports *real-world integration* in application design by providing the means to specify spatio-temporal service constraints. The ability to attach such constraints to services before they are invoked is an important aspect of WSN application programming.
- The service-oriented approach enables in-network *data aggregation* using services with inputs from multiple sensor services.
- OASiS supports specification of both application-specific and network QoS *requirements* and can handle QoS violations using application re-configuration to satisfy QoS requirements.

3 OASiS Programming Model

The OASiS programming model is organized into three key paradigms: service-oriented, object-centric, and ambient-aware programming.
Service-Oriented Architecture

In OASiS, a service is the basic unit of application functionality. Services have well-defined interfaces consisting of input and output ports, as well as the data types supported by each. Wiring a group of domain services together results in a service graph describing application flow. In OASiS, services can be wired together to compose a service graph representing any type of data flow application. In addition to services and their connections, a service graph can include constraints which restrict where and when a service can be invoked.

Because it is desirable for services to communicate asynchronously with each other, OASiS employs the globally asynchronous, locally synchronous (GALS) model of computation [4]. GALS maintains asynchronous communication between services, while intra-service communication, such as method calls, exhibit synchronous behavior.

Dynamic network behavior in WSNs can cause problems during application execution such as service unavailability and violation of constraints. In OASiS, each node maintains two service repositories, a local service repository (LSR) that catalogs the application services available locally, and a discovered services repository (DSR) that catalogs the remote application services that have been discovered in the past. The combined use of these two repositories allows the system to be more responsive in the event of service unavailability. Service discovery is passive in that service providers wait for a specific request before advertising a service. This passive approach was found to be the most energy efficient for mobile ad hoc networks with limited power resources [5].

Object-Centric Programming

Complexity in OASiS is minimized by providing the application developer with a means for specifying global network behavior; that is, behavior from the viewpoint of the physical phenomenon of interest. A physical phenomenon of interest is represented by a finite state machine (FSM), also referred to as the logical object. We selected the FSM representation because this model of computation is an intuitive method for describing the distinct states a physical object might exhibit. Each FSM mode corresponds to a different physical state, and contains a service graph specifying the appropriate actions to take for the specific situation. A service graph corresponding to an object FSM mode encapsulates the application behavior if the physical object is in a particular physical state.

Before a logical object is instantiated, a physical phenomenon must be detected. This is achieved by comparing sensor data with an object context. Because multiple nodes may detect the same physical phenomenon at roughly the same time, a mechanism is required to ensure only one logical object is instantiated. To provide this guarantee, OASiS employs an object-owner election algorithm, similar to that of [6].

After the object creation protocol completes, exactly one node, referred to as the object node, is elected owner of the logical object corresponding to the
physical phenomenon. The object also has a migration condition, which if evaluates true, invokes the object migration protocol. In tracking applications, for example, an increase in the variance of location estimate can serve as a migration condition, and the owner selection policy will choose the node closest to the physical phenomenon. The migration process consists of running the owner election algorithm to select the migration destination based on the selection policy and transferring the object state to the new object node.

**Ambient-Aware Programming**

Nodes in an ambient-aware sensor network must always have up-to-date knowledge of their neighborhood, including the services provided, and the properties of the nodes providing them. OASiS allows constraints to be placed on services relating them to a specific physical location or to the relative locations of other services. For example, correct execution of an application may only be possible if sensing services are provided by nodes surrounding the physical phenomenon. Such real-world integration requires a dynamic service configuration mechanism. Dynamic service configuration instantiates the service graph by satisfying constraints, binding the services together, checking for any constraint violations, and reconfiguring the application when a violation is found.

We have identified several different types of constraints that can be placed in the service graph. Depending on the scope of the constraint, it can be atomic (applying to a single service) or compositional (applying to a group of services). Constraints can be further categorized as either property-based or resource-allocation-based. Property constraints specify a relation between the properties of the scoped services, while resource-allocation constraints define a relationship between nodes that provide the scoped services. A significant constraint for real-world integration is *enclose*. The definition of *enclose* varies for different sensor domains. For example, one domain can define an *enclosed* region to be the overlap of member sensing ranges. The enclosed region in case of camera sensors with is the intersection of field-of-view of all cameras.

We model service graph configuration as a constraint satisfaction problem (CSP). A feasible solution to the CSP is the host sensor node assignment for each service in the service graph. The main idea of our algorithm is to prune the service graph design space as much as possible for all different types of constraints, followed by backtracking until a feasible solution is found. The specific pruning method depends on the constraint under consideration. Additional details on types, representation and satisfaction of service constraints can be found in [7].

**4 OASiS Middleware**

We have developed a suite of middleware services that support the programming model features presented in Section 3. OASiS is intended for resource-constrained devices which are unable to use current Web service standards such as SOAP, WSDL, and UDDI due to their bulky XML-based messaging format. Instead,
OASiS uses a compact byte-sequence message structure for accessing services and passing data between them.

In order to take advantage of high-bandwidth Web services, the sensor network must have access to at least one World Wide Web Gateway. The Gateway resides on a base station and provides access to Web services by translating node-based byte sequence messages to the XML-based messages used in most Web service standards. When a service discovery message arrives, the Gateway must locate this service on the Internet. This is accomplished by using the UDDI protocol and service access is achieved by means of XML-based SOAP messages. It is up to the Gateway to compose and parse these various XML messages and marshal the data appropriately when translating between the sensor network and the World Wide Web.

We have implemented our middleware on the Mica2 mote hardware platform running TinyOS [8]. The code was developed using galsC [1], a GALS-enabled extension of nesC [9], the programming language for the motes. Further details about the implementation and a case study can be found in [7].

Acknowledgements This work is partially supported by Microsoft External Research and by NSF Grant CCR-0225610.

References