

An Object-Centric Programming Framework for Ambient-Aware, Service-Oriented Sensor Networks

Manish Kushwaha, Isaac Amundson*, Xenofon Koutsoukos*,
Sandeep Neema, Janos Sztipanovits
Vanderbilt University
Nashville, TN 37235, USA

Chang Hong Lin, Wayne Wolf
Princeton University
Princeton, NJ 08544, USA

{manish.kushwaha, isaac.amundson, xenofon.koutsoukos}@vanderbilt.edu

Abstract—Classical programming languages are too low-level and not well-suited for large scale sensor network applications. Volatile connections and dynamic network topologies render the development of sensor networks very complex. In this abstract, we describe our current work on a service-oriented framework based on object-centric, ambient-aware programming. Object-centric refers to the programming model wherein objects are first class elements that represent external physical entities allowing the application developer to program at a higher level of abstraction. Ambient-aware is a paradigm that incorporates network volatility into the programming model. We outline such a framework and describe the main individual components.

I. INTRODUCTION AND MOTIVATION

Wireless sensor networks (WSNs) are inherently dynamic in nature due to node mobility, failure, and unreliable communication links. It is therefore imperative that WSN applications consider dynamic network topology in order to ensure their correct execution. Unfortunately, tackling the issue of dynamic network behavior places a substantial burden on the programmer. In addition to an application's core functionality, the programmer must implement low-level WSN operations such as robust communication protocols, resource management algorithms, and fault tolerance mechanisms. Not only does the implementation of these components require substantial time and effort on behalf of the programmer, but it also increases the risk of deploying incorrect code, due to the often unanticipated behavior of distributed network applications.

In this abstract, we propose an object-centric ambient-aware programming framework, enabling the programmer to develop WSN applications without having to deal with the complexity and unpredictability of underlying network dynamics. In object-centric programming, objects are first class programming elements representing physical phenomena being monitored by the network. The basic idea behind such a framework is to provide programmers with a higher level of network abstraction, allowing applications to be developed from the viewpoint of the object.

Object-centric programming can be used for a variety of applications such as mobile vehicle tracking, fire detection and monitoring, and distributed gesture recognition [1]. Gesture recognition on a moving object requires logical entities

for keeping track of the object collaborating with image processing components as well. A typical sensor network will consist of a series of slightly overlapping video surveillance nodes. When a human target is detected, the application spawns a tracking agent, which migrates across the network following the real-world object. Meanwhile, image-processing components are employed to identify the object's various hand movements and facial expressions. These algorithms are typically resource intensive and require collaboration among several nodes. While monitoring the object, the tracking agent must simultaneously locate various remote image processing components, and forward collected imaging data.

Object-centric programming by itself does not address the issues of network failures and dynamic network topology. The programming model must be enhanced for encompassing the dynamics of wireless networks and if possible, provide guarantees for the correct operation of software applications. Communication failures in traditional application programming 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-oriented programming has emerged as a paradigm for mobile computing [2] and provides several characteristics that are useful for robust sensor network applications. The basic idea of ambient-aware programming for sensor networks is dynamic service discovery initiated by the monitored object in order to manage the required resources in the changing network topology. Further, such objects should be reflective to allow for reconfiguration and error recovery, organize the distribution of collaborative tasks, and ensure that the application cannot be blocked, for example, due to a link failure.

To develop such a programming paradigm, our framework takes a service-oriented approach to behavioral decomposition. In a service-oriented WSN application, each activity (sensing, aggregation, service discovery) 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 that allow them to be described, published, discovered, and invoked over a network. These properties permit services to be dynamically composed into complete

*The work of these authors is partially supported by a grant from Microsoft External Research.

applications. Because many WSN applications will consist of multiple services on multiple nodes, planning, scheduling, and service discovery mechanisms are built in to the framework. These mechanisms are designed to function effectively in the presence of unreliable communication links.

The object-centric paradigm has been successfully used in the EnviroSuite programming framework [3]. EnviroSuite provides a high level of abstraction, however its modularity can be enhanced following a service-oriented approach for adding software components. In EnviroSuite, fault-tolerance can be achieved by employing periodic *heartbeat* messages to nodes surrounding an object of interest and can be improved by incorporating ambient-aware functionalities.

II. PROGRAMMING FRAMEWORK

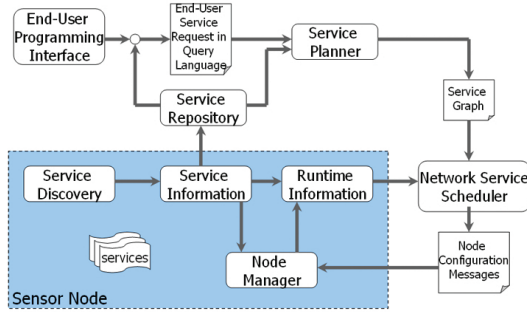


Fig. 1. Architecture

In our proposed framework (Figure 1) there are four main components, viz. end-user programming interface, service planner, network service scheduler and node manager.

Programming interface is responsible for injecting service requests into the sensor network. The end user writes an object application using the services registered in the service repository. The service repository keeps a registry of all the services discovered in the network. The repository can be similar to the UDDI registry that is used for most web-services. The output of this component is a service request in a service-request-query-language.

Service planner takes the service request as the input and generates a service graph of constituent services listed in service repository. In general, a service request is *expanded* into constituent services until no more services need to be expanded. Services are expanded based on the inputs they require to run. Consider a vehicle tracking application in which the user wishes to track any vehicle in an area of interest. Figure 2 shows the service graph for the vehicle tracking service. The service requires vehicle classification and position as inputs. A vehicle classification service provides classification and position data but requires a logical object for the vehicle. A vehicle is sensed by a vehicle sensing service, after which the logical object is created.

Network service scheduler (NSS) will schedule services at the sensor nodes that provide target sensing functionalities while satisfying certain constraints, for example on the number and locations of the nodes. NSS requires runtime information

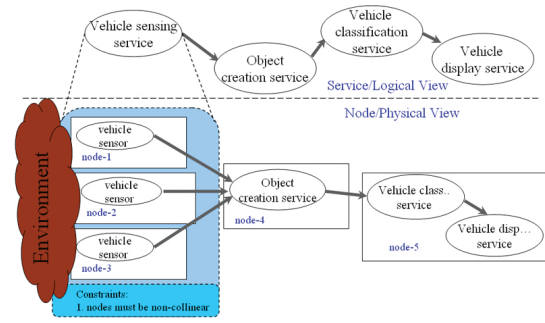


Fig. 2. An example service graph and its schedule on sensor nodes

of the services provided by nodes as well as their properties, such as location, remaining power, etc. The output of NSS consists of service configuration messages which are sent to nodes for composing the service graph.

Node manager is responsible 1) configuring the node services based on a configuration message, 2) initiating service discovery, and 3) managing the local runtime service and resource information required by the NSS. The node manager will initiate service discovery to locate the services to run on the object following the current service. An integral part of the node manager is a **service discovery protocol**. The node running service discovery sends out a publish message indicating its interest in discovering services in the network. All the nodes that receive the message respond with an information message containing information for the available services and the cost for using these services, for example, with respect to power. The receiving node processes this information and builds a local service database.

III. IMPLEMENTATION

Currently, we are developing the runtime infrastructure that is necessary for supporting the execution of object-centric, ambient-aware applications. Our target platform is Mica motes running TinyOS. The services include object related services such as objection creation, management, and migration service as well as node manager and network level components such as network service scheduler, service planner, and service repository. We use galsC [4] for sensor node programming because it provides an actor abstraction for sensor nodes that handles elegantly asynchronous communication. Using the actor abstraction provided by galsC, we can employ actors as the building blocks of services.

REFERENCES

- [1] C. H. Lin, T. Lv, W. Wolf, and B. Ozer, "A peer-to-peer architecture for distributed real-time gesture recognition," in *IEEE International Conference on Multimedia and Expo (ICME)*, 2004.
- [2] J. Dedecker, T. V. Cutsem, S. Mostinckx, T. D'Hondt, and W. D. Meuter, "Ambient-oriented programming," in *OOPSLA Companion 2005*, 2005.
- [3] L. Luo, T. Abdelzaher, T. He, and J. Stankovic, "Envirosuite: An environmentally immersive programming system for sensor networks," in *ACM Transaction on Embedded Computing System (TECS)*, 2006.
- [4] E. Cheong and J. Liu, "galsc: A language for event-driven embedded systems," in *DATE '05: Proceedings of Design, Automation and Test in Europe*, 2005.