A Component-based Approach for Service Distribution in Sensor Networks

Amirhosein Taherkordi
University of Oslo
Department of Informatics
N-0314 Oslo
amirhost@ifi.uio.no

Romain Rouvoy
ADAM Project-team
INRIA Lille – Nord Europe,
University Lille 1,
LIFL CNRS UMR 8022
F-59650 Villeneuve d'Ascq
romain.rouvoy@inria.fr

Frank Eliassen
University of Oslo
Department of Informatics
N-0314 Oslo
frank@ifi.uio.no

ABSTRACT

The increasing number of distributed applications over *Wireless Sensor Networks* (WSNs) in ubiquitous environments raises the need for high-level mechanisms to distribute sensor services and integrate them in modern IT systems. Existing work in this area mostly focuses on low-level networking issues, and fails to provide high-level and off-the-shelf programming abstractions for this purpose. In this paper, we therefore consider WSN programming models and service distribution as two interrelated factors and we present a new *component-based* abstraction for integrating WSNs within existing IT systems. Our approach emphasizes on reifying distribution strategies at the software architecture level, thus allowing remote invocation of component services, and facilitating interoperability of sensor services with the Internet through *Web service-enabled* components. The latter is efficiently provided by incorporating the REST architectural style—emphasizing on abstraction of high-level services as resources—to our component-based framework. The preliminary evaluation results show that the proposed framework has an acceptable memory overhead on a TelosB sensor platform.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*

Keywords

Wireless sensor networks, integration, component-based service distribution, RESTful Web services

1. INTRODUCTION

WSNs have been recognized as an emerging technology for monitoring and controlling a variety of phenomena, such as environmental surveillance, infrastructures, home and office, and medical environments [3, 17, 25]. Whereas the early WSN applications were primarily concerned with sensing primitive environmental data and delivering those raw data to a central node, recent applications consider sensor nodes as *service-enabled devices* with the ability of providing distributed services to other nodes of the network.

The other concern is related to using WSNs in ubiquitous computing environments, where sensor nodes populate with actuators, RFID readers, and mobile devices for monitoring ambient environments and reacting to the external stimuli gathered by different devices [1, 8, 14]. As each device in such a heterogeneous environment has its own requirements in terms of system software and communication protocol, integrating them at a higher software level brings the challenges of service orchestration and interoperability. Furthermore, sensor nodes as one of the technologies driving the future *Internet of Things* [10] should be equipped with protocols that enable them to interoperate with every IP-enabled node across the Internet.

In contrast to the approach of *plain text message streaming* widely adopted in traditional "sense and send" models [2, 20], the service-oriented approaches in WSNs rely on high-level ad-hoc communication protocols to discover and exhibit services across different computing environments. Although there have been a number of significant efforts to distribute sensor services and integrate them with conventional network platforms [11, 19, 26], the state-of-the-art mostly focuses on low-level APIs and fails to provide a concrete communication abstraction relieving the programmer from dealing with the tedious and error-prone distribution tasks in WSNs.

Thus, in this paper we aim at providing a software framework for WSNs that enables programmers to develop distributed sensor services and integrate them with existing IT systems at a high-level programming abstraction. To achieve that, we leverage the concepts of *component-based programming* to design, describe and implement distributed and interoperable WSN services. In particular, we reconsider Remora—a lightweight component model for high-level programming in WSNs [23]—in order to extend it with the capability of distributing a component's services across the network. The second part of the paper is devoted to integrating the above component-based solution to a uniform interaction model in order to facilitate interoperability of sensor services with the Internet through *Web service-*

MidSens '10, November 30, 2010 Bangalore, India.

enabled components. This interaction model is inspired by REST—an architectural style for distributed systems emphasizing scalability of component interactions, generality of interfaces, and independent deployment of components [9]. From another point of view, this paper illustrates the feasibility of utilizing a component framework to develop RESTful Web services for resource-constrained sensor nodes.

The remainder of this paper is organized as follows. Section 2 gives a survey of existing approaches. In Section 3, we briefly discuss the REMORA component model. The architecture and specification of our service distribution proposal are presented in Section 4, while the implementation and the evaluation result are discussed in Sections 5 and 6, respectively. Finally, Section 7 concludes this paper and identifies some future work.

2. RELATED WORK

Efforts in providing distribution and integration approaches for WSNs can be categorized into four groups: *i)* providing low-level APIs to enable distribution within sensor networks for exchanging raw sensed data among nodes, *ii)* high-level component-based techniques to enable remote invocation of sensor services, *iii)* exploiting Web services standards to bridge the gap between sensor nodes and the Internet, and *iv)* protocols allowing sensor nodes to connect seamlessly to other devices in pervasive computing environments.

The first group emphasizes on exposing proprietary interfaces in order to unicast or multicast plain text messages across the network. Most of the works in this field are inspired by the concepts of *message-oriented communication*. TINYOS [12] as the most popular operating system for WSNs provides a number of *Active Message Interfaces* to abstract the underlying radio communications services and *Software Components* that implement these interfaces [2]. Other sensor operating systems, such as CONTIKI [6], have also paved the same way [7]. They have also paid a significant consideration on integrating WSNs with the Internet by providing the IP protocol stack for low-power sensor nodes. However, high-level programming APIs and application-level service distribution facilities are not addressed by these proposals.

A number of lightweight component models have also been proposed to provide RPC-like service invocations in sensor networks. In addition to the fact that none of them consider Web service-based distribution, they essentially suffer from making extensive use of sensor resources or lack of generality. As an example of the former, OPENCOM [4] offers *Component Frameworks* (CFs) to model local and distributed interactions between cooperating components, but OPENCOM is a generic model and basically developed for resource-rich platforms. LOOCI [13], falling in the latter category, is a loosely-coupled component infrastructure for WSNs, featuring an *Event Bus* to bind distributed LOOCI components. Nonetheless, the Java-based implementation of LOOCI limits its usage to the SUNSPOT nodes.

The third group has been motivated by the concept of Internet of Things—a technological revolution to connect daily objects and devices to large databases and networks, and therefore to the Internet. In this model, Web services standards are used to integrate WSNs and the Internet, *e.g.*, in SOCRADES [5] Web services are tailored at the gateway device where the *Device Profile for Web Services* (DPSW) is used to enable messaging, discovery and eventing on devices with resource restrictions. However, since the current footprint of DPSW for sensor nodes is too large, this solution is only deployable on gateways. To overcome this issue, Priyantha et al. [19] propose a SOAP-based Web services, called *Tiny web services*, for WSNs. However, apart from its complexity, this work mainly focuses on low-level issues related to Web integration in TINYOS-based sensor networks.

A few works have also been devoted to the use of simple Internet protocols. In fact, these approaches are proposed due to the high resource needs and complexity of SOAP-based Web Service protocols for WSNs. TINYREST is one of the first attempts to integrate WSNs into the Internet [15]. It uses the HTTP-based REST architecture to retrieve/update the state of sensors/actuators. The TINYREST gateway maps a set of HTTP requests to TINYOS messages in order to link MICA motes to any Internet client. Beside the fact that in TINYREST only a gateway is able to connect to the Internet (not any individual sensor node), this approach fails to follow all standard HTTP methods. The work reported in [11] also presents a REST-based gateway to bridge the Web requests to powerful SUNSPOT nodes.

Approaches in the last category are based on the *Universal Plug and Play* (UPnP) architecture. UPnP is a set of computer network protocols, promoted by the UPnP Forum [24], allowing devices to connect seamlessly in the home and corporate environments. In UPnP, all communications are peer-to-peer and transferred over TCP/IP, UDP and HTTP. As most sensor node products are not equipped with the support of UPnP standards, in [16] a new WSN platform is proposed to support connectivity of sensor nodes to a UPnP-enabled device. However, UPnP discovery and control protocols are heavyweight for a typical sensor node and there is a very limited range of sensor nodes supporting UPnP-based communications. Furthermore, using UPnP in this framework imposes many new hardware and new software set-up for integration support.

The main difference between our approach and the works presented above is that we address the service distribution problem in a top-down manner. Specifically, we first formulate high-level service distribution requirements based on a programming model for WSNs, and then consider how the programming proposal can *simplify* and *generalize* distribution and integration of sensor services.

3. PROGRAMMING MODEL

We adopt a component-oriented approach to address the programming needs of service distribution in WSNs. Componentization provides a high-level programming abstraction by enforcing interface-based interactions between system modules and therefore avoiding any hidden interaction via direct function call, variable access, or inheritance relationships [21]. This abstraction instead offers the capability of black-box integration of system services. Therefore, it theoretically becomes a good candidate for developing distribution tasks in WSNs, beside the fact that component-based software development is extensively used in WSN programming. Thus, in this section we briefly discuss a WSN-specific component model we have recently proposed—REMORA [23]. Then, in the next section, we propose our service distribution model based on this component framework.

3.1 REMORA in a Nutshell

The main motivation behind proposing REMORA is to fa-

cilitate high-level and event-driven programming in WSNs through a component-based abstraction. REMORA achieves this goal by: *i)* deploying components within a lightweight framework executable on every operating system written in the C language, and *ii)* reifying the concept of *event* as a first-class architectural element simplifying the development of event-oriented scenarios. The latter is one of the key features of REMORA since a programming model for embedded systems is expected to support event-driven design. Reducing software development effort is the other objective of REMORA. A REMORA component is composed of two main artifacts: a component *description* and a component *implementation*.

Component Description. REMORA components are described in XML as an extension of the *Service Component Architecture* (SCA) model [18] in order to make WSN applications compliant with the state-of-the-art componentization standards. Based on the SCA Assembly Language, the component description indicates the specifications of the component including *services*, *references*, *interfaces*, *producers*, *consumers*, and *properties* (cf. Figure 1). A service can expose a REMORA interface, which is a separate XML document describing the functions provided by the component. A reference can request a REMORA interface, which describes the operations required by the component. Similarly, a producer identifies an event type generated by the component, while consumer specifies a component's interest on receiving a particular type of event.

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType name="COMPONENT_NAME">
  <service name="SERVICE1_NAME">
    <interface.remora name="INTERFACE1_NAME"/>
  </service> ...
  <reference name="REFERENCE1_NAME">
    <interface.remora name="INTERFACE2_NAME"/>
  </reference> ...
  <property name="PROP1_NAME" type="PROP1_TYPE">
    PROP1_DEFAULT_VALUE
  </property> ...
  <producer>
    <event.remora type="EVENT1_TYPE"
                  name="EVENT1_NAME"/>
  </producer> ...
  <consumer operation="CONSUMER_OPERATION">
    <event.remora type="EVENT2_TYPE"
                  name="EVENT2_NAME"/>
  </consumer> ...
</componentType>
```

Figure 1: The XML template for describing Remora components.

Component Implementation. The component implementation contains operations implementing: *i)* the component's service interfaces, *ii)* event handlers, and *iii)* private utilities of the component. REMORA components are implemented by using the C programming language extended with a set of new commands. This extension is essentially proposed to support the main features of REMORA, namely, component instantiation, event processing, and property manipulation.

Remora Development Process. A REMORA application consists of a set of REMORA components, containing descriptions and implementations of software modules (cf. Figure 2). The REMORA engine processes the component descriptions and generates standard C code deployable within the REMORA framework. The framework is an OS-independent C module supporting the specification of the REMORA component model. Finally, the REMORA framework is deployed on the target sensor node through the REMORA runtime, which is an OS-abstraction layer integrating the application with the system software.

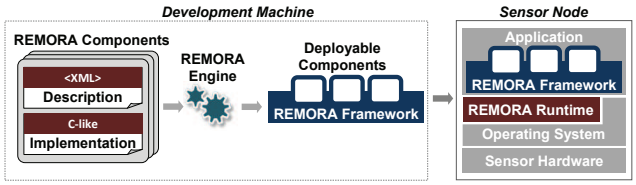

Figure 2: Development process of Remora applications.

4. COMPONENT-BASED SERVICE DISTRIBUTION

REMORA provides a complete component-based abstraction for *node-level programming* in the sense that the accessibility scope of REMORA services is limited to the runtime process of a sensor node. Services deployed on two different nodes can communicate through an indirect ad-hoc interaction model, which is message-based and handled by low-level data transmission protocols provided by sensor operating systems. It means that for network-level programming, the developer requires to switch from component-based programming to system-level messaging techniques, a cumbersome and error-prone task. It is also the case when a REMORA-based application is integrated with heterogeneous systems with a different set of software and hardware settings.

4.1 Basic Concepts

We propose a new key concept, which is considered as a basis for our distribution proposal. This concept, called REMORA *Distribution*, refers to a new paradigm in component-based software design for resource-constrained networks in which every system functionality is encapsulated in a component and distributed for the use of both homogenous and heterogeneous systems. This indicates the capability of defining platform-specific *bindings* for REMORA components in order to transparently handle the communication issues in WSNs. In this way, REMORA will automatically generate part of the code which is required for sending data over the network or invoking methods, instead of time-consuming and error-prone programming with low-level APIs.

Generally, the REMORA bindings are categorized into two classes: *remote binding* and *interoperable binding*. The former specifies a type of remote service call occurring within a sensor network between two homogenous sensor nodes in order to send the required service data from one node to the other node (like Java RMI), while the latter refers to the invocations happening beyond a sensor network between a sensor node and a node in an existing IT system, such as the Internet. To concretely describe the REMORA distribution, we first discuss the principles underpinning this framework, including:

Access Transparency. From the programmer's point of view, local and distributed services should be accessed using identical operations. Additionally, she/he should not be forced to write distribution-related code that is out of scope of application logic.

Location Transparency. The distribution mechanism should enable REMORA components to access remote services without any knowledge of their location in the network.
Synchronisms. The calling mode can be either *synchronous* or *asynchronous* corresponding to the mode supported by the network protocol, *e.g.*, RESTful calls are synchronous, while remote calls over RIME—a WSN-specific communication protocol implemented by CONTIKI [7]—are handled asynchronously.
Pluggability of Bindings. REMORA bindings should be maintained in a well-structured way in the sense that every binding-related library for a particular network protocol should be easily pluggable to or unpluggable from services provided by REMORA components without affecting other parts of the system.

Figure 3 depicts the overall architecture of the REMORA distribution model. Each component of the system exhibits a set of services, which can interact in two different ways: locally or remotely. In the local mode, service calls are carried out through a simple and lightweight node-level invocation plan, while the distributed calls occur either remotely (between two nodes with the same set of configurations), or in an interoperable manner (between a REMORA-enabled node and an IP-enabled node).

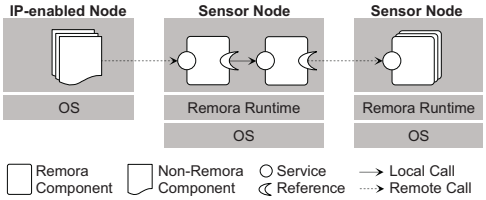

Figure 3: Architecture of the Remora distribution model.

While the development of each main type of REMORA binding would require consideration of many kinds of challenges, such as programming constructs, streaming, networking, and runtime support, the contribution of this paper is limited to providing service-level integration of sensor nodes with IP-enabled systems. We therefore leave the issues of the REMORA remote binding as our future work and in this paper we study the REMORA interoperable binding in the context of Web integration. REMORA *Web binding* is basically concerned with integrating sensor services with IP-enabled networks by translating the service requests to the equivalent RESTful web services. In this way, the distribution mechanism of REMORA follows a standard and widely-adopted protocol to link WSN services to any Web-enabled device in ubiquitous environments.

4.2 REMORA Web Services

In this section, we first describe the principles of the REST architectural style and RESTful Web services, and then present our component-based approach for integrating REMORA services with the Web.
REST Principles. The *REpresentational State Transfer* (REST), as coined by Fielding [9], is an architectural style for distributed systems emphasizing scalability of component interactions, generality of interfaces, and independent deployment of components. The "resource-oriented" principles of REST are described through the REST triangle defining the principles for *addressing*, *accessing* and *encoding* a collection of resources using the Internet standards. A distributed system, which follows REST principles, is called RESTful, *e.g.*, the Web. In a RESTful system, a component can interact with other distributed components by knowing two things: *i)* the unique identifier of the representative resource of component, and *ii)* the predefined standard operations to invoke (GET, POST, PUT and DELETE). In this model of interaction, the client-server separation of concerns can simplify component implementation, reduce the complexity of connector semantics, and increase the scalability of server components.
RESTful Web services. Web services are a set of standards and techniques for developing interoperable distributed applications that are accessed via HTTP and executed on a remote system hosting the requested services. Nowadays, Web services are generally categorized into two groups: *SOAP-based Web services* and *RESTful Web services*. The former follows the *Simple Object Access Protocol* (SOAP), which is a heavyweight protocol specification for exchanging structured information in the implementation of Web services, while the latter is a lightweight model based on REST, which does not impose SOAP or XML.

We believe that the simplicity and uniform interfaces of RESTful Web services is an efficient approach to integrate any individual sensor node to any RESTful system in pervasive environments. Additionally, there have been reported a number of valuable works focusing on the integration of REST and WSNs and providing the *primitives* required to RESTful programming in sensor systems [26]. Therefore, we adopt this approach and study how to enable RESTful Web service development in WSNs by the REMORA component model.

Since an application built from REST principles is transformed from operation-centric into a data-centric model, every entity that offers a service becomes a resource (*e.g.*, a temperature sensor) that can be identified unambiguously using a Uniform Resource Identifier (URI) [22]. Every resource then defines a uniform interface just including four main operations provided by REST (GET, POST, PUT and DELETE). Therefore, incorporating REST principles into REMORA requires applying the REST triangle of nouns, verbs, and content types to the specification of REMORA component model.
Remora Service Identifier. The service identifier is a unique noun described using the URI format. Therefore, service identifiers include a server address, a service path, and a sequence of request parameters:

```
/server-address/service-path?request-params
```

Since REMORA components in a sensor application are basically organized in a hierarchical model (like Java packages), the URI of a distributed service is corresponding to the hierarchical organization of REMORA components within the application. For instance, assume that Sensors is a component providing a service for temperature sensor (`myTemperature`) and another service for light sensor (`myLight`), thus the services of the Sensors component, located under `/app/peripheral`, can be addressed as follow:

```
/node-id/app/peripheral/Sensors/myTemperature
/node-id/app/peripheral/Sensors/myLight
```

As the REMORA engine has a complete knowledge about the structure of applications (including hierarchical organi-

zation of system components), service identifiers are automatically obtained by the REMORA engine and therefore the programmer does not need to specify any identifier for distributed services during the application development. Nevertheless, we have provided a special REST binding tag (`binding.rest`) through which the programmer can change the URI of a component, as well as its services. Figure 4 shows the description of the Sensors component, where the URI of the `myTemperature` service is changed. Therefore, the new URI of the `myTemperature` service becomes:

```
/node-id/app/myTemp
```

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType name="app.peripheral.Sensors">
  <service name="myTemperature">
    <interface.remora name="api.ITemperature"/>
    <binding.rest uri="/app/myTemp"/>
  </service>
  <service name="myLight">
    <interface.remora name="api.ILight"/>
  </service>
  ...
</componentType>
```

Figure 4: RESTful service identification in Remora.

Remora Service API. The four main operations of REST can be mapped to the operations implemented by a particular REMORA service. Figure 5 shows an excerpt of the interface `ITemperature`. Similar to the REMORA services, operations of an interface can be considered as a REST resource and have their own URI. Depending on the functionality of a service and variety of its operations, we may need to define a new URI for a collection of service operations. The other important part of the REST binding tag is that we need to provide the name of the equivalent REST operation for each operation. For example, the current temperature of environment can be retrieved from a temperature sensor node using the HTTP request:

```
GET http://device.uio.no:8080/node-id/app/myTemp
```

which returns the current temperature. Temperature configuration information can also be pushed into a sensor node by using an HTTP PUT request as follow:

```
PUT http://device.uio.no:8080/node-id/app/
                                  myTemp/threshold
```

In this case, the HTTP request is sent to the operation `setThreshold`, along with the required parameters, then the REMORA framework will call the `setThreshold` operation. Note that in the sample code of Figure 5, `threshold` is considered as a new separate REST resource with its own URI.

Content Type. Finally, for a REST binding we can specify the format of data delivered to the client. For instance, according to the REST annotation defined for `getCurrent` in the `ITemperature` interface, this operation provides the current temperature in the JSON format (cf. Figure 5).

4.3 A Concrete Use Case

An example use case that can benefit greatly from our distribution framework is *home monitoring systems*. Such applications are characterized as being filled with sensor nodes

```
<?xml version="1.0" encoding="UTF-8"?>
<interface.remora name="api.ITemperature">
  <operation name="getCurrent" return="xsd:short">
    <annotation.rest method="GET"
                     type="application/json"/>
  </operation>
  <operation name="setThreshold" return="void">
    <in name="threshold" type="xsd:short"/>
    <annotation.rest method="PUT" path="/threshold"/>
  </operation>
  ...
</interface.remora>
```

Figure 5: Mapping the REST verbs to equivalent Remora operations and identifying content types.

to observe various types of ambient context elements (temperature, smoke, occupancy, and health conditions of inhabitant), actuators to physically control home appliances (lights, TV, and air conditioning), and smart phones to provide information about the preferences of owners.

Integration of multi-scale entities is one of the main challenges in such an environment. Mobile devices and sensors have different hardware and software capabilities, which make some devices more powerful than others. Therefore, this heterogeneity requires a flexible and simple solution that supports multiple interaction mechanisms and considers the restricted capabilities of some devices. In particular, regarding sensor nodes, the immaturity of high-level communication protocols, as well as the inherent resource scarceness, bring a critical challenge to the system: how to connect sensor nodes to mobile devices and actuators through a standard high-level communication protocol. We believe that the RESTful framework presented above can significantly smooth the way to integrate a sensor network with actuators and existing infrastructure networks in the home monitoring systems.

5. IMPLEMENTATION

The implementation of the proposed model consists of two main parts according to the model's architecture. The first part is concerned with modifying the implementation of REMORA specifications in order to support the new commands related to Web services programming, such as the REST binding tag. The second part is dedicated to developing a middleware framework taking care of REST communication issues. The former is performed within the implementation of the REMORA engine. In addition to enhancing the engine to support the new commands, it should statically create the data structures required for maintaining the REST-related service data, such as resource identifiers and Service API information.

The middleware framework, supporting Web service-based communications, is a lightweight module integrated with the REMORA runtime and the REMORA framework to support runtime requirements of the proposed model. To this end, we first need to choose and exploit an existing library that is flexible and completely support the REST principles. We adopt the *REST framework* presented by Yazar et al. [26] for two reasons. Firstly, they have used the CONTIKI operating system to develop the RESTful architecture and our current REMORA runtime is also available on CONTIKI. Secondly, this framework's architecture along with CONTIKI's extensive work on the IP protocol stack provide a set of complete and easy-to-use low-level libraries for RESTful en-

hancements.

Figure 6 depicts the overall implementation architecture of Remora Web services, where Contiki's core and TCP/IP stack lie in the bottom. The gray box represents the REST framework and contains core modules required for RESTful Web services development over Contiki. The Remora runtime is integrated with the REST framework through the *REST Wrapper API*. It should be noted that Wrapper API is one of the main features of Remora, providing a well-described method for integrating a Remora application with underlying system software [23].

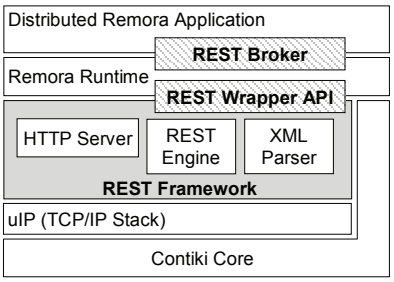

Figure 6: Overall implementation architecture of Remora Web services.

Figure 7 shows an excerpt of the REST Wrapper API, including operations for initializing REST engine, setting the representation type for a HTTP connection through the Remora component HTTPConnection, sending GET data, receiving POST data, and updating the status of a connection. The implementer of this interface is a Remora component in which the low-level APIs of the REST framework are invoked.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface.remora name="api.wrapper.IREST">
  <operation name="initialize"/>
  <operation name="setReperType">
    <in name="httpConnection"
        type="rest.comm.HTTPConnection"/>
    <in name="type" type="xsd:int"/>
  </operation>
  <operation name="setGETData">
    <in name="httpConnection"
        type="rest.comm.HTTPConnection"/>
    <in name="data" type="xsd:string"/>
  </operation>
  <operation name="getPOSTData" return="xsd:string"/>
  <operation name="setHTTPStatus">
    <in name="httpConnection"
        type="rest.comm.HTTPConnection"/>
    <in name="type" type="xsd:int"/>
  </operation>
  ...
</interface.remora>
```

Figure 7: An excerpt of the REST Wrapper API.

REST Broker contains a set of Remora components processing REST requests received from Remora runtime. Specifically, it is an intermediate module for handling the REST requests received from a Web client or sent from the sensor node to a node hosting RESTful Web services. The broker is also in charge of retaining the list of application-specific resources and the corresponding Remora Web services APIs.

6. PRELIMINARY EVALUATION

As mentioned before, we adopt Contiki as our system platform and our hardware platform is the popular TelosB mote equipped with a 16-bit TI MSP430 MCU with 48KB ROM and 10KB RAM. In the preliminary evaluation, we assess the performance of the system based on the memory overhead incurred by the REST Broker and the REST wrapping component (implementer of the REST Wrapper API) on ROM and RAM.

The memory footprints are categorized into a fixed overhead and a dynamic overhead. The fixed overhead is the minimum additional memory required for a distributed application, regardless of the number of distributed services running within the application. Table 1 shows the fixed memory requirements, which turn out to be quite reasonable with respect to both code and data memory. As seen from the table, the main memory consuming module is the REST framework. Therefore, the fixed memory cost can be further reduced by switching to a more efficient REST framework in the future.

Table 1: The fixed memory requirement of Remora Web services framework.

Module	Code Memory (bytes)	Data Memory (bytes)
Remora Runtime	494	14
REST Wrapper	94	0
REST Broker	252	8
REST Framework	4668	76
Total	**5508**	**98**

The dynamic memory overhead is calculated based on the number of distributed resources (services, components or operations) in an application and the number of REST verb mappings (GET, POST, PUT and DELETE) for each resource. The memory overhead of the former is variable according to the length of a resource's name, while the latter consumes 2 bytes of ROM to retain the starting memory address of a REST operation.

7. CONCLUSION AND FUTURE WORK

Distributing WSN services through standard and widely-accepted communication protocols is of high importance. In this paper, we presented a high-level programming abstraction in order to enable service distribution in WSN applications and relieve the programmer from the burden of dealing with low-level APIs for developing distributed sensor services. This component-based approach promises a new abstraction for integrating sensor software modules to the Internet through upgrading component-level services to Web services over a lightweight RESTful architecture. This flexible framework is also potentially able to exhibit sensor services to other types of network protocols by implementing platform-specific bindings. Our future work includes addressing the Remora remote binding, occurring when component services within a sensor network need to remotely communicate via a particular sensor network protocol.

8. ACKNOWLEDGMENTS

This work was partly funded by the Research Council of Norway through the project SWISNET, grant number 176151.

9. REFERENCES

[1] I. F. Akyildiz and I. H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351–367, 2004.

[2] P. Buonadonna, J. Hill, and D. Culler. Active Message Communication for Tiny Networked Sensors. In *INFOCOM '01: Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, Alaska, USA, 2001. IEEE.

[3] M. Ceriotti et al. Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment. In *IPSN'09: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pages 277–288, San Francisco, CA, USA, 2009. IEEE.

[4] G. Coulson et al. A generic component model for building systems software. *ACM Trans. Comput. Syst.*, 26(1):1–42, 2008.

[5] L. de Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio. SOCRADES: A Web Service Based Shop Floor Integration Infrastructure. In *The Internet of Things*, volume 4952 of *LNCS*, pages 50–67. Springer, 2008.

[6] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, Tampa, FL, USA, 2004. IEEE Computer Society.

[7] A. Dunkels, F. Österlind, and Z. He. An Adaptive Communication Architecture for Wireless Sensor Networks. In *SenSys'07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 335–349, Sydney, Australia, 2007. ACM.

[8] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the Physical World with Pervasive Networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002.

[9] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine, USA, 2000. PhD thesis.

[10] N. Gershenfeld, Raffi, R. Krikorian, and D. Cohen. The Internet of Things. *Scientific American*, pages 76–81, 2004.

[11] D. Guinard, V. Trifa, T. Pham, and O. Liechti. Towards physical mashups in the web of things. In *INSS'09: Proceedings of the 6th International Conference on Networked Sensing Systems*, pages 196–199, Pittsburgh, PA, USA, 2009. IEEE.

[12] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Networked Sensors. *SIGPLAN Not.*, 35(11):93–104, 2000.

[13] D. Hughes et al. LooCI: a Loosely-Coupled Component Infrastructure for Networked Embedded Systems. In *MoMM'09: Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*, pages 195–203, Kuala Lumpur, Malaysia, 2009. ACM.

[14] H. Liu, M. Bolic, A. Nayak, and I. Stojmenovic. Taxonomy and Challenges of the Integration of RFID and Wireless Sensor Networks. *IEEE Network*, 22(6):26–35, 2008.

[15] T. Luckenbach, P. Gober, K. Kotsopoulos, Andreas Kim, and S. Arbanowski. TinyREST: a Protocol for Integrating Sensor Networks into the Internet. In *REALWSN'05: Proceedings of the Workshop on Real-World WSNs*, Stockholm, Sweden, 2005.

[16] M. Marin-Perianu et al. Decentralized Enterprise Systems: a Multi-Platform Wireless Sensor Network Approach. *Wireless Communications, IEEE*, 14(6):57–66, 2007.

[17] A. Milenković, C. Otto, and E. Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Computer Communications (Special issue: Wireless Sensor Networks: Performance, Reliability, Security, and Beyond)*, 29:2521–2533, 2006.

[18] OSOA. The Service Component Architecture. `http://www.oasis-opencsa.org/sca`.

[19] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny Web Services: Design and Implementation of Interoperable and Evolvable Sensor Networks. In *SenSys'08: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, pages 253–266, Raleigh, NC, USA, 2008. ACM.

[20] E. Souto, G. Guimar aes, G. Vasconcelos, M. Vieira, N. Rosa, and C. Ferraz. A message-oriented middleware for sensor networks. In *MPAC'04: Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing*, pages 127–134, Toronto, Canada, 2004. ACM.

[21] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[22] R. T. Berners-Lee and M. Fielding, L. Uniform Resource Identifier (URI): Generic Syntax, 2005. `http://www.ietf.org/rfc/rfc3986.txt`.

[23] A. Taherkordi, F. Loiret, A. Abdolrazaghi, R. Rouvoy, Q. L. Trung, and F. Eliassen. Programming Sensor Networks Using Remora Component Model. In *DCOSS'10: Proceedings of the 6th International Conference on Distributed Computing in Sensor Systems*, pages 45–62, Santa Barbara, CA, USA, 2010. Springer.

[24] UPnP. Universal Plug and Play. `http://www.upnp.org`.

[25] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *EWSN'05: Proceedings of the Second European Workshop on Wireless Sensor Networks*, pages 108–120, Istanbul, Turkey, 2005.

[26] D. Yazar and A. Dunkels. Efficient Application Integration in IP-based Sensor Networks. In *BuildSys'09: Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 43–48, Berkeley, CA, USA, 2009. ACM.