DARMA: Adaptable Service and Resource Management for Wireless Sensor Networks

Pedro Javier Del Cid¹, Danny Hughes², Jó Ueyama³, Sam Michiels¹, Wouter Joosen¹

¹ Department of Computer Science, Katholieke Universiteit Leuven, 3000, Leuven, Belgium.
² Xi’an Jiaotong-Liverpool University, Suzhou Industrial Park, Suzhou, Jiangsu, 215123, China.
³ Escola de Artes, Ciências e Humanidades, Rua Arlindo Béttio 1000, 03828-000, Sao Paulo, SP, Brasil.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ABSTRACT

This paper argues that service oriented architectures provide a good mechanism for managing outside interaction with software resources on wireless sensor networks. Specifically, this paper introduces a lightweight service platform designed to meet the specific characteristics of wireless sensor networks. The proposed architecture provides lightweight, yet flexible service-level agreements, together with adaptive management of sensor network resources. Critically for resource constrained sensor networks, our framework ensures that services are shared in an optimal manner between multiple client applications, while providing autonomic mechanisms to reason about fault tolerance and optimization. Furthermore, our approach actively promotes point of action processing which provides significant benefits in both embedded and enterprise deployments. We illustrate the appropriateness of the proposed architecture through a prototype implementation and evaluation using the LooCI component model and the SunSPOT platform.

Categories and Subject Descriptors
C.2.4 [Computer-Communication Networks]: Distributed Systems, Distributed Applications

General Terms
Management, Design.

Keywords
Service Oriented Architectures, Wireless Sensor Networks, Resource management, Service provision, middleware

1. INTRODUCTION

Wireless Sensor Networks (WSNs) are highly resource constrained, dynamic and heterogeneous. In such an unreliable environment, Service Level Agreements (SLAs) may be used as a mechanism to specify the quality attributes that an application requires from the WSN. SLAs can also be used by the WSN to inform the sharing of scarce resources between concurrently executing applications. In concert with autonomic management entities, SLAs may also be used to inform adaptive resource management and optimize how underlying WSN resources are used to provide services.

This paper introduces the Distrinet Adaptive Resource Management Architecture (DARMA), a light-weight service platform that provides simple, yet flexible, SLAs tailored to meet the requirements of sensor network environments. Additionally, Service Level Agreements (SLAs) are an appropriate abstraction through which to delegate management functionality to the point of action. The point of action in WSNs refers to the physical location where events occur. Delegating decision making to the point of action has clear benefits for enterprise applications, such as: (i.) decreasing process execution and transactional costs, (ii.) providing quicker response times and (iii.) improving service quality and the overall flexibility of the system [8]. Point of action processing also has particular benefits in the field of WSN, as it reduces multi-hop network traffic, thus freeing bandwidth and increasing node battery life.

SLAs are interpreted and enforced by an autonomic service manager, which gathers context data from the WSN and uses this to adapt the way in which underlying software resources may be used to provide a given service. We will show that such an autonomic approach can increase fault tolerance while optimizing the sharing of components between applications. Furthermore, by separating application logic from resource management, through the use of the service manager, our system allows for a more appropriate control over the non-functional requirements of WSN. We have realized and evaluated an initial implementation of DARMA for the Loosely Coupled Component Infrastructure (LooCI) [7] on the Sun SPOT platform [10].

The remainder of this paper is structured as follows: Section 2 provides background on LooCI and Service Oriented Architectures (SOA) for WSN. Section 3 presents the design of DARMA. Section 4 presents DARMA in operation. Section 5 evaluates the prototype implementation of DARMA. Finally, section 6 concludes and discusses directions for future work.

2. BACKGROUND

This section begins with an overview of the Loosely-coupled Component Infrastructure, using which DARMA is built. We then provide a brief overview of the state-of-the-art in
Service Oriented Architectures for WSN and other networked embedded systems.

2.1 Component Architecture

Run-time reconfigurable component models address many critical problems encountered in WSN environments, which have limited resources, are highly dynamic and, as they are expected to operate unattended for long periods of time, must evolve to meet changing application requirements:

- Managing Limited Resources: through formally defined interfaces, component based architectures have been shown to encourage re-use of software elements.
- Managing Dynamism: sensor networks are highly dynamic. Component-based reconfiguration offers adaptation mechanisms to manage this dynamism.
- Managing Evolution: as application requirements change over time, component based approaches allow for system evolution through the deployment of new components.

The Loosely-coupled Component Infrastructure (LooCI) [7] is designed to support embedded Java ME platforms such as the Sun SPOT [10] or Java ME smart-phones. LooCI is comprised of an easy-to-use component model and a simple yet extensible networking framework. Each LooCI node is connected via a common event-bus communication substrate. Like other embedded component platforms such as RUNES [2] or OpenCOM [3], LooCI components support run-time reconfiguration, concrete interface definitions, introspection and support for the re-wiring of bindings.

Unlike OpenCOM or RUNES, LooCI components are indirectly bound over the event bus. Thus all LooCI components define their interfaces as the set of LooCI events that they publish. The receptacles of a LooCI component are similarly defined as the events to which they subscribe. Each LooCI event has a globally unique identifier which classifies the event type in terms of a global descriptive hierarchy.

We believe that a loosely-bound component model such as LooCI is an excellent fit with adaptive resource management. As the LooCI binding model is inherently indirect, operating over an event bus, it is possible for a service manager to modify bindings based upon execution context in a manner that is transparent to the upper layers. As binding modification is transparent, we are also able to more easily separate the concerns of resource management and application functionality. This allows end-users to specify the services they require and WSN administrators to tailor system behavior without the need to implement new components. The LooCI event bus also provides a common point of interception to support the gathering of contextual data.

2.2 Service Orientation for WSN

Pohl et al. [9] apply loosely coupled service bindings to the problem of industrial automation and control on a powerful sensor and actuator platform. Pohl argues that loosely coupled services are a promising model for binding components on sensor networks, as they promote interoperability, service discovery and allow for the re-wiring of bindings at run time. However, the traditional Web Service approaches used in this paper are too heavy-weight to be feasible in embedded WSN environments.

The approach used in [1] offers a more compact mechanism for data representation by using the WAP binary XML content format. This reduces the size of XML documents for transmission and simplifies parsing. Other lightweight approaches include the Tesserae language [6] and specialized service description languages such as Servilla [5].

These approaches enumerated above vary in the level of application independence they provide: [9] and [6] are designed for specific application domains, while [1] and [5] are application independent. The latter three approaches also provide services specifications that include Quality of Service (QoS) attributes. Additionally, [6] and [5] also provide optimized service selection that takes into account contextual factors such as the current energy available on potential service providers. Only [1] includes end of service definitions and provides the capability to set threshold alarms for specific services. Tesserae [6] also provides an autonomic mechanism to select the optimal provider.

The approach embodied in DARMA extends the current state-of-the-art by considering the complete life-cycle management of services, using autonomic service managers to optimize the way in which WSN resources are used to fulfill multiple, concurrent service requests. DARMA also provides mechanisms to reason about fault tolerance and component parameterization. Additionally, our approach actively promotes point of action processing; which, as discussed previously, provides significant benefits in both embedded and enterprise deployments.

3. ADAPTABLE RESOURCE MANAGEMENT ARCHITECTURE

The core DARMA architecture has four key conceptual elements; (i.) service managers, (ii.) service selection, (iii.) service resources (SRC) and (iv.) context monitors. Each system element is modeled as a LooCI component. These elements are described below, and illustrated in Figure 1 below:
3.1 Service Managers

Each service manager maintains references to collections of remote components of a common type (service resources). Each service manager takes as input a number of SLAs and, based upon these specifications, the service manager will select the most appropriate service (using the service selection component) to fulfill the specified requirements of the SLA. The appropriate selection of service resource components is informed by context monitors running on each sensor node which provide dynamic contextual data. Where a service resource is used to meet the requirements of multiple SLAs, the service manager will negotiate an optimized parameterization for that service.

3.2 Service Selection Components

Service selection components implement the service selection control loop. This component maintains a directory of all services offered within the managed group, which is used to match service requests to available services. If multiple potential service providers exist, then contextual data on all potential providers will be gathered. The service selection component then uses this data to select the best available service provider. Service providers can be ranked based on any factor for which a context monitor is available, including: available battery life, current CPU load or any other QoS-related factor. The choice selection factor can be easily modified in the selection policy, e.g. you can select some providers based on available energy or current work load.

Following initiation of a service, the service selection component periodically polls available service resources to ensure that the selected service remains (i.) available and (ii.) the optimal choice to fulfill the SLA. If a change is required, due to malfunction or service degradation, the service manager will re-wire the binding the component with the highest current ranking. This is all done transparently to the requesting application, thus lowering the burden on application developers.

3.3 Service Resource Components

Service Resource Components (SRCs), see Figure 2, are an extension of the base LooCI component type [7] and may run on any sensor node. This component is the core mechanism that we use to implement services and is used by applications on the WSN. Example SRCs include a temperature monitoring service, or location awareness service. SRCs extend basic LooCI components by allowing for multiple applications to use the same service concurrently. Each SRC component has independently defined parameterization interfaces and alarms. As SRCs are based on standard LooCI components, developers may easily add services to a WSN at run-time.

3.3.1 Parameterization Interfaces

The parameterization interface provides inspection and control over a collection of component parameters. Each parameter controls some aspect of component behaviour, specifying the following data: (i.) parameter name (ii.) minimum value (iii.) maximum value and (iv.) units. Consider the following example of the parameterization interface of a temperature monitoring component, which may be parameterized to sample at a rate of between 1 sample per second and 1 sample per minute. This component would expose a “SAMPLE_TIME” interface that has a minimum of 1 (second), a maximum of 60 (seconds) and units of 1 (second).

When multiple applications request different parameterizations, the SRC will first check that each parameterization is allowed by the relevant parameterization interface. Where multiple valid parameterizations are requested, the SRC will configure the relevant parameterization interface to the minimum value that can meet all parameterizations. The SRC then dynamically deploys a filter on each interface, which from the perspective of remote components makes it appear that each bound component has received the requested parameterization (while in reality, the component has a single optimized parameterization). If any parameterization request cannot be met, the SRC will generate a reconfiguration exception.

3.3.2 Alarms

Each SRC also maintains a list of alarm conditions, realized in the previously mentioned interface filters. Alarms are specified at run-time and they are specified in service requests. When an alarm condition is met, SLAs allow for the dissemination of actuating events over the LooCI event bus. For example: if the temperature in a container of frozen food rises above zero degrees Celsius, an SLA may specify that a message should be sent to the monitoring application. This is an example of point of action processing, where our flexible SLA scheme allows decisions to be taken locally to resolve problems without the overhead of contacting back-end management systems.

Figure 1: Deployment view and binding diagram.

Figure 2: Service to resource component.
3.3.3 Context Monitors

Context monitors are standard LooCI components and may be deployed on any sensor node. Context monitors provide contextual data to service managers and SRCs. Example data includes, but is not limited to: battery life, available dynamic memory, available flash memory and current CPU load. We provide a set of common context monitors, but also expect that developers will develop their own monitors to support the requirements of their target application domain.

4. DARMA OPERATION

Interaction with DARMA may be thought of as being composed of four phases: (i.) service request, (ii.) service initiation, (iii.) SLA enforcement and (iv.) service termination.

4.1 Service Request

Service requests are specified using a light-weight, high-level and human-readable specification language. Each service request is identified by a unique ID. A service request comprises of the following data:

- Spatial Accuracy: This is defined by a target node address and the hop-radius around that address that may adequately serve the request. For example, an address of (A) and hop radius of (1) requires that the request be served by (A) or any of its immediate neighbors.
- Service Parameterization: describes the desired parameterization of the SRC (as described in section 3.3.1).
- Alarms: specify threshold conditions, an event type and the LooCI address to which alarm notifications should be sent, as described in section 3.3.2.
- Service Termination: services may be required to terminate after a given time period (measured in seconds) or upon receipt of specified event.

An example service request is shown below:

```
[Request Information]
[Requesting Application Address]
[Managing Node Address]
[Service requested; unique event ID]
[Targeting]
[Component Interface]
[Target Address]
[Hop Radius]
[Parameterization]
[Optimal Parameterization]
[Minimum Parameterization]
[Alarms]
[Threshold values]
[Operation: EQUALS, <, >]
[Value: BOOLEAN, NUMBER, STRING]
[Alarm recipient Ip address]
[Tx message for notification]
[Termination]
[Transmit on Completion]
[Event ID] / [Time Out]
```

As the service request specification is lightweight and imposes minimal additional overhead while allowing for flexible targeting, parameterization and termination. As we will demonstrate in section 5, flexible targeting also allows for fault tolerance, while flexible parameterization allows components to be safely used by multiple compositions.

4.2 Service Initialization

The service selection component handles service discovery. The service selection component maintains a registry of available services within the managed group. When a service request is received, the service selection component inspects all nodes in the targeting specification, comparing context data to the optimization specification. The service selection component will then attempt to parameterize the optimal service as specified in the service request. Where this is successful, the service has been initialized, where parameterization fails, the target node will respond with an error event and the selection component will attempt to initiate the service on the next most optimal SRC.

4.3 SLA Enforcement

Following service initialization, the service selection component will periodically poll all nodes within the target specification, gathering context data. Where an SRC serving a request is found to be no longer optimal, it will be swapped for the most optimal SRC available. Service providers can be ranked based on any factor for which a context monitor is available, as explained in section 3.2. In this way, the desired service optimizations are respected throughout the lifetime of each service. As this process is autonomic and separate from application logic, it lowers the burden on developers and increases control over non functional requirements.

4.4 Event Caching and Termination

Each service request also contains a service termination specification, which defines when the Service Manager should terminate a service. Termination may be based upon a time-out, or based upon the reception of a unique event type by the service manager. The termination specification also defines whether events should be transmitted to the client in real-time, as they are produced, or whether they should be cached locally and transmitted only when a service completes. While simple, this functionality provides effective primitives to decrease power costs due to transmission as described in [4].

5. EVALUATION

In this section, we provide a preliminary evaluation of our initial implementation of DARMA built using LooCI micro-components [7] on the Sun SPOT sensor platform (180MHz ARM CPU, 512KB RAM, 4MB flash memory) running the SQUAWK JVM (BLUE version). Section 5.1 analyses the footprint of DARMA. Section 5.2 describes a simple case-study scenario. Section 5.3 then explores how our flexible SLA specification language can be used to provide fault tolerance and efficient sharing of components. Finally, Section 5.4 analyses overhead for developers and discusses the role of SLAs in separating the concerns of application development and service management.
5.1 DARMA Footprint

We have analyzed the DARMA footprint in terms of the additional static and dynamic memory consumed by a DARMA SRC compared to a standard LooCI micro-component. We also analyzed the static and dynamic memory consumed by our prototype service manager. These results are shown in Table 1 and Table 2 respectively.

As can be seen from Table 1, a null DARMA component (i.e. a component with no functional code, interfaces or receptacles) imposes minimal additional static (778) and dynamic (1901) of memory overhead compared to a standard LooCI component. Each additional parameterization interface exposed consumes an additional 178 bytes of static memory and 200 bytes of dynamic memory.

Table 1 – SRC / LooCI Component Footprint Comparison

<table>
<thead>
<tr>
<th>LooCI Component</th>
<th>DARMA SRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Memory</td>
<td>587 bytes</td>
</tr>
<tr>
<td>RAM</td>
<td>1800 bytes</td>
</tr>
</tbody>
</table>

Table 2 – DARMA Service Manager Footprint

<table>
<thead>
<tr>
<th>DARMA Service Manager</th>
<th>9.6KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Memory</td>
<td>19.6KB</td>
</tr>
</tbody>
</table>

Finally, DARMA also implies significant power and bandwidth savings when compared to the use of a purely component driven approach, as a single, parameterizable SRC may be deployed along with a per-service SLA (order of 80 bytes) rather one component per service (order of 600 bytes).

5.2 Case-Study Scenario

Consider the following case-study scenario: STORAGE_CO owns and operates a temperature controlled warehouse storage facility. STORAGE_CO provides temperature monitoring using a DARMA-enabled WSN, which acts as a shared resource for the customers of STORAGE_CO, each of which negotiates a price for their specific SLA.

CHOCOLATE_CO requires only coarse-resolution temperature data (in terms of time and space) about the temperature their packages are being stored at. Additionally they do not require having this information sent to the backend systems in real time, they just need the tracing information available by the end of the service. CHEMICAL_CO, however, stores hazardous materials and thus requires high-resolution monitoring of the temperature of each of their packages and requires the information be transmitted in quasi real time to the backend systems.

CHEMOLATE_CO would thus specify an SLA that requires temperature data from within 2 hops of each package at a low frequency, e.g. 60 seconds. This allows STORAGE_CO great flexibility in terms of assigning this service to a node, allowing them to better optimize service assignment, which, along with low data rates will reduce service cost. CHEMICAL_CO would however specify an SLA that required temperature data from their specific package and a higher frequency, e.g. 10 seconds. This guarantees the temporal and spatial accuracy of data received by CHEMICAL_CO, though it implies higher overhead and thus cost.

In the case of CHEMICAL_CO, you can see that their SLA could be specified as [event ID] a generic temperature service in degrees Celsius, the [Target] location as its package address + 2 hops away and the [parameters] as Optimal sampling frequency of 60 seconds and minimal of 60 seconds, [alarms] where not required and [termination] specified as transmit on completion TRUE and specified an end of service event Id. This allows the service manager to select a service provider SRC from any node within 2 hops of radio range from the package and a minimum sampling frequency of 60 seconds. An important thing to notice is that as temperature is sampled, the data is cached locally on the node and is only transmitted after the end of service event has been received. This is an example of point of action processing that effectively decreases the amount of transmission and network load.

In the case that the chosen provider would fail or the service degrades to a point where the sampling frequency was above 60 seconds (due to low battery), the service manager would select another provider to provision the required service. The selection process is done as explained in section 3.2. The SLA is a simple, lightweight and semantically correct way to request services in a way that is transparent to the requesting application. Furthermore, the service manager will ensure that a qualified service is available for the required duration.

5.3 Component Sharing and Fault Tolerance

It may also be that the temperature monitoring SRC deployed to meet the SLA of CHEMICAL_CO is within the hop radius specified by the SLA of CHEMICAL_CO. In this case, the component would be parameterized as described in section 3.3.1, allowing the SLA of CHEMOLATE_CO to be met without the overhead of deploying an additional component.

In the case of CHEMOLATE_CO, this may also provide additional fault tolerance. As the resource manager handling the CHEMOLATE_CO SLA may select between multiple SRCs within the target hop-radius to provide the service. In reaction to service degradation or service unavailability; binding repair is also a relatively light-weight operation. We measured the time that the resource manager required to repair a binding using a test-bed of three nodes, configured as described in the introduction to this section. All nodes
were within radio range of each other (thus minimizing network overhead, which would vary depending upon the WSN). For each experiment, a random binding was broken and the time taken to rebind to an alternative component in the target hop-radius was recorded. On average, this operation took only 48 milliseconds, making seamless and transparent repair feasible for many application domains.

5.4 Separation of Concerns and Developer Overhead

Our service specification language is very simple and human readable (see section 4.1). Service specifications require a maximum of 15 lines of ‘code’ and the creation and distribution of SLAs may be further facilitated using high-level, potentially domain-specific tools.

The service manager provides an effective mechanism to separate application logic from resource management. This separation provides the following benefits: (i) ease the burden for application developers, where now they do not need to concern themselves with how a service is to be provided and what resources provide the optimal options. Furthermore they do not need to manage the service lifetime and worry about service degradation and service failures. (ii) WSN resource management can be independently controlled by network managers, where applications do not control individual resources or their allocations. The service manager can use selection policies that optimize service reuse, uniform workload distribution or prolong network lifetime, amongst others. (iii) Promote runtime platform extensibility; services can be developed and deployed at runtime in the WSN. The new services are automatically registered with the service managers which can allocate them to any requesting application transparently to upper layers.

6. CONCLUSIONS AND FUTURE WORK

This paper presented a lightweight, component-based service platform for WSN. This architecture manages WSN software resources based upon a simple SLA specification and, based upon contextual information gathered from the network, will optimize component behavior and dynamically optimize the enactment of an SLA.

Evaluation shows that our initial prototype of DARMA is capable of optimizing how components are shared between multiple services and the separation of application logic and resource management can provide significant benefits in WSN deployments. Furthermore the DARMA SLA language can be used to provide fault tolerance and extend the lifetime of sensor nodes. In the short term, our future work will focus upon further evaluation of DARMA using a real-world logistics, case-studies. In the medium term, we will also investigate how DARMA may be integrated with traditional Web Service technologies.

7. ACKNOWLEDGEMENT

Research for this paper was partially funded by IMEC, IWT, the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U. Leuven. This research is conducted in the context of the IWT-SBO-STADIUM project No. 80037 [11].

8. REFERENCES