國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

初稿

First draft

蘇適
Penn H. Su

指導教授：許永真 博士
Advisor: Jane Yung-Jen Hsu, Ph.D.

中華民國 102 年 1 月

January, 2013

國立臺灣大學
資訊工程學系

碩士論文

初稿

蘇適 撰

102
1

國立臺灣大學碩士學位論文
口試委員會審定書

初稿
First draft

本論文係蘇適君 (R99922157) 在國立臺灣大學資訊工程學研究所完成之碩士學位論文，於民國 102 年 1 月 21 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

____

____

____

____

____

所 主 任 ______________________________

Acknowledgments

I’m glad to thank . . .

致謝

感謝...

中文摘要

Abstract

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an overview of the thesis. First, we describe why WSN deployment and maintenance are still difficult. Then we address the problem by describing a new programming model component models which is used to separate design abstractions between high-level application design and low-level hardware constructs. Next, we describe the needs for fault tolerance system in component based middleware and why it is hard. Lastly, we describe our proposed solution to this problem.

1.1 Motivation

1.1.1 Wireless Sensor Network Deployment is Hard

Wireless sensor networks are areas filled with network of tiny, resource limited sensors communicating wirelessly. Each sensor is capable of sensing the enviornment in its proximity. Wireless sensor networks are employed in a variety of applications ranging from home automation to millitary.

Sensor networks offer the ability to monitor real-world phenomena in detail and at large scale by embedding devices into the environment. Deployment is about setting up an sensor network in a real-world environment. Deployment is a labor-intensize and cumbersome task since environmental influences or loose program logic in code might trigger bugs or sensor failures that degrade performance in any way that has not been observed during pre-deployment testing in the lab.

The real world has strong influences of the function of a sensor network that could change the quality of wireless communication links, and by putting extreme physical strains on sensor nodes. Laboratory testbed or simulator can only model to a very limited extent of those influences.

There have been several reports on sensor network installations where they encountered problems during their deployment[3][12][2][18][11][14][19][20].

Testbed in laboratory environment can still not model the full extents of the influences a real world enviroment could do. Deployment still a big problem in wireless sensor network applications.

1.1.2 Maintaining WSN deployment

The possibilities of sensor failures is a fundamental characteristic of distributed applications. So it is critical for distributed applications to have the ability to detect and recover from failures automatically, since not only the sensor deployments are usually huge in scale, the deployment are to run unattended for a long period of time.

There have been abundance of research in fault tolerance for wireless sensor network deployments proposing low-level programming abstractions or framework to implement redundancy or replication.[21][10]

1.1.3 Component based middleware for distributed applications

However, the increase in number, size and complexity in WSN applications makes high-level programming an essential needs for development in WSN platforms.

This is supported by several reasons. Firstly, the diversity of hardware and software for WSN platform is as diverse as the programming models for such platforms[15]. Secondly, existing programming models usually sacrifice resource usage with efficiency, which is not suitable for tiny sensors in sensor networks. Thirdly, existing programming models still forces developers to learn low-level languages, which imposes an extra burden to developers, and it goes to show when the reusability for those programming models are low in existing applications.

Software componentization, or lightweight component models, has been recognized to tackle the concerns above. It brings several advantages over past approaches with separate of concern, module reusability, de-coupling, late binding.

The primary advantages of this approach is reconfigability, adaptability in applications like never before, since the high-level components and low-level constructs are loosely decoupled and interpretted by the middleware, high-level application logic can be added functionalities and framework around it without changing the application logic at all, and applications can adapt to different hardware configurations without changing any internal logic.

In Intel-NTU Center Special Interests Group for Context Analysis and Management (SIGCAM), our team have been collaborating on a project, called WuKong, to develop an intelligent middleware for developing, and deploying machine-to-mahicne (M2M) applications. The main contribution of this project is to support inlligent mapping from a high-level flow based program (FBP) to self-identified, context-specific sensors in a target environment[13].

1.2 Problem definition

1.2.1 Component Based Fault Tolerance System

The development and deployment for a fault tolerant application is still immature in most component based middleware. Even though components are modular in providing reconfigability to applications, they are still not failure resistent and cannot recover from failures. The problem is worsen when the number of components increase in applications, the developers would still bear the burden of manually programming the applications to ensure fault tolerance.

1.3 Proposed Solution

I propose to investigate how applications described in high-level flow based program language could translate to low level constructs to create a fault tolerant, applications in WuKong. In detail, we investigate how system components collaborate to achieve a common goal while satisfying application requirements to achieve self-fault detection, self-fault diagnosis, and self-fault recovery.

With colleagues from the Intel-NTU Special Interests Group for Context Analysis and Management (SIGCAM) at National Taiwan University, I have developed a new intelligent fault tolerance system, called Fur (temp), as part of WuKong project, an Intelligent Middleware for developing and deploying applications on distributed platforms. Our proposed system consists of agents collaborating to simplify fault tolerance development and to shorten deployment cycle for heterogeneous M2M applications.

The frivolous nature of requirements in applications, and actual physical sensor environements, along with the hard to predict user priorities, each contributes a unique challenge in its flavor to developing an adapatable fault tolerance solution.

Below are the list of areas that this work will address solutions in.

Intelligent Mapping

Flow-Based Programming (FBP) Paradigm has been used in WuKong to enable loosely coupling between high-level application logic and low-level hardware constructs. WuKongs also achieves late-binding to bridge the worlds together at the last stage of deployment using a technique known as intelligent mapping. Intelligent Mapping is a process in which high-level application logics are broken down into components and then mapped to appropriate nodes. Sensor Profile Framework (SPF) are used in mapping to handle heterogeneous sensor platforms, thus applications can be successfully converted into lower hardware constructs to generate low-level intermediate code to deploy to the sensor network.

Sensor Profile Framework

Sensor Profile framework provides an high-level abstraction to sensor capabilities to enable building more complex application logic.[13]

User Policy Framework

Allowing user-friendly specification of application executive objectives, and context-dependent management of system peformance.

Group Communication Systems

A group communication system deals protocols for synchronizing group states among group members in an consistent manner.[4]. When the applications are deployed to the sensor network, groups will be formed to implement redundancy. For a group of low-power sensor to collaborate on a common goal, along with high-level application requirements and Sensor Profile Framework, a new group communication protocol is needed to support fault tolerence.

1.4 Thesis Organization

Our work overlaps many diverse but interconnected domains, each topic being itself a subject of advanced research and abundant literature. The second chapter gives a brief background overview of these domains. We start by describing wireless sensor networks. Then we go on to discuss fault tolerant design for distributed systems, it's objectives and recent developments. Finally, we will be talking about component model based middleware. The third chapter gives some overview of related work. The following two chapters describe our work in fault tolerance for WuKong system. We first give an overview of some essential components in WuKong. Then we give a comprehensive description of our fault tolerance system architecture. We conclude this chapter by highlighting how the integration of those subsystems, through careful scrutiny, could bring to the development of a fault tolerant WSN application. We then present two experiments to evaluate the performance, correctness of each mechanism in the following chapter. This thesis concludes with a summarization of our proposed system and future work.

Chapter 2

Background

2.1 Wireless Sensor Networks

Sensor nodes are equipped with low-power, low-cost, and failure-prone sensors or actuators. Sensor networks are networks of sensor nodes that connect to the physical space that are intrumented to produce data that could be meaningful for further research. They collaborate to collect, process and disseminate environmental information[1].

Sensor network could be homogeneous, meaning all nodes are identical with same sensors, actuators and hardware setup. Sensor networks could also be heterogeneous where nodes have different sensors, actuators and hardware setup. Heterogeneous networks require higher level management and organization resources. Wireless sensor networks are nodes that communicate through air by sending electronic signals. Wireless communications aren't stable, as it is highly influenced by environmental factors.

2.1.1 Redundancy

Sensor networks are usually deployed in large scale and unattended in long period of time. Sensor networks communicate with low-power wireless radios to aid scientists in collecting spatial data that could lead to more understanding of the environment. However, several challenges such as node failures, message loss, and sensor calibration leaves the effectiveness of sensor networks in question. With the assumption of spare homogeneous resources, redundancy is used in sensor networks to increase fault tolerance against node failures. The system is designed with backup nodes that could automatically recover and replace should one node fail.

2.2 Component Based Middleware

Middleware enables communication and management of data that simplfies complex distributed applications.

As most applications for wireless sensor network involves management of data and communication between network of nodes, middleware is integral in providing a unified experience for implementing more complex architecture such as service-oriented architecture.

However, the separation of design abstractions between low-level hardware and high-level application logic has not been successful in sensor based systems.

It is also not successful in terms of making them adapatable and evolvable for new services in new environments.

2.3 WuKong: The intelligent middleware for M2M applications

2.3.1 Goal

Deployment and development for M2M applications are in its infancy today. As many applications are still single purpose in homogeneous networks with specific network protocols. The hardware has a fixed range of sensors, and the applications cannot be easily ported to other platforms.

The existing middleware support that decouple high-level application design abstractions and low-level hardware has not been successful.

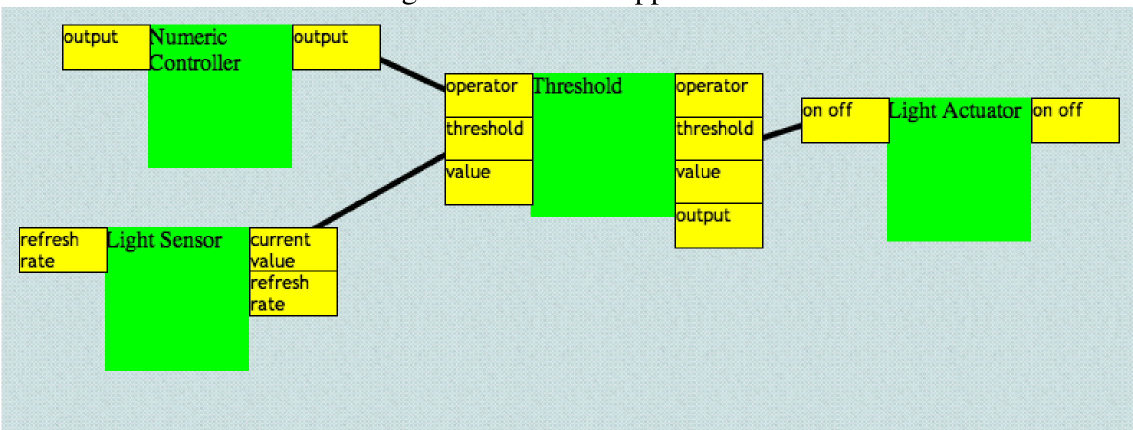
In Intel-NTU Center Special Interests Group for Context Analysis and Management (SIGCAM), we have been collaborating on a project, called WuKong, aiming to develop an intelligent middleware for developing, and deploying machine-to-mahicne (M2M) applications with ease. The main contribution of this project is to support inlligent mapping from a high-level flow based program (FBP) to self-identified, context-specific sensors in a target environment[13].

2.3.2 Flow Based Programming

M2M applications are by definition distributed where the application requirements involve a network of nodes collaborating for some common goals. M2M applications are typically defined by its flow of information between components, as opposed to more traditional applications that focus more on local information processing.

Flow Base Programming is best suited for describing M2M applications as it allows the developers for the applications to focus more on the abstraction meaning of the com-

Figure 2.1: A FBP application



ponents instead letting the unimportant details such as the hardware to stick right in the face. The result application will contain all necessary information for the framework to construct low-level details to implement the flow.

Applications are designed and constructed on FBP canvas by dragging a set of abstract components from the library as illustrated in Figure 2.1 Each component is illustrated by a green block, each block has a set of properties, each with different access modes, such as readonly, writeonly, readwrite. Properties on the left of the greenblocks are properties that could be written, and properties on the right are readable. Components are connected by links, which is drawn by linking two properties in different components.

Some components represent physical hardware such as a sensor, or an actuator while some other components could represent virtual processes such as mathmatical computations, comparisons, etc. However, the final physical implementations of the components are only made during application deployment by the Master but not during FBP construction.

Components expose their interface through properties. A link is only made with properties with matchinkg data type. The FBP applciation in Figure 2.1 illustrates a simple scenario where the light actuator will turn on the light if light level drops below some value. The Numeric Controller component will be assigned to a user input device used by users to set its desire light threshold, which its output is sent to Threshold component. The light value is sensed from Light Sensor component and sent to Threshold. If the light value sensed is below the threshold value, Threshold will output a boolean to set the on off property of Light Actuator to turn the device, which will be deteremined during deployment, that it is represented by on or off.

2.3.3 Sensor Profile Framework

While FBP defines the logical view of an application, WuKong profile framework allows tracking, identification of physical resources within the Sensor Network. There are a range of sensors which provide similar functionality with different level of quality, it could model the sensor capability to enable handling heterogeneous sensors and provide a common abstraction for the logical view.

There are two main concepts in Sensor Profile Framework, WuClasses and WuObjects. WuClasss model components by exposing a number of properties describing, and allow access to, a specific resource represented by the class. Drawing from the example in Figure 2.1, the on off property of Light Actuator component is boolean writeonly. WuClass also implements an update() function to describe a component's behavior. For example, Threshold has four properties: operator, threshold, value, output. The output value is determined from the previous 3 properties that it returns true when the value is lower or higher than the threshold which depends on the value of the operator, and it returns false otherwise.

WuObjects are the main unit of processing that are hosted on the nodes. Each WuObject is an instance of WuClasses. It allows the framework to achieve 4 responsibilities:

1. Allow the Master to discover the current status of a node with the list of WuClasses and WuObjects it has.

2. Create new WuObject instances on a node to start receiving data and doing local data processing.

3. Trigger executions in WuObjects, either periodically or as a result of changing inputs.

4. Propagate changes of properties between linked properties in different components, which may be hosted locally or remotely.

Property Propagation

The profile framework is in charge of communication between WuObjects as well, which are not necessarily on the same nodes. Profile Framework monitors the changes in properties and propagate the changes to the connected WuObjects. For example, if a Temperature WuObject is connected to a Threshold WuObject, the changes in Temperature current value property will trigger propagation from the Profile Framework to propagate the new value in current value to the Threshold WuObject connected property, and since Threshold WuObject could be on a different node, the framework will take care of this by initiating a wireless connection between the nodes to send the data over. Once a new value has been set, Threshold WuObject will also trigger its update() function to recompute its output properties which in turn would cause another chain of propagation to the linked WuObjects.

2.3.4 Compilation and Mapping

Figure 2.2: WuKong application build flow

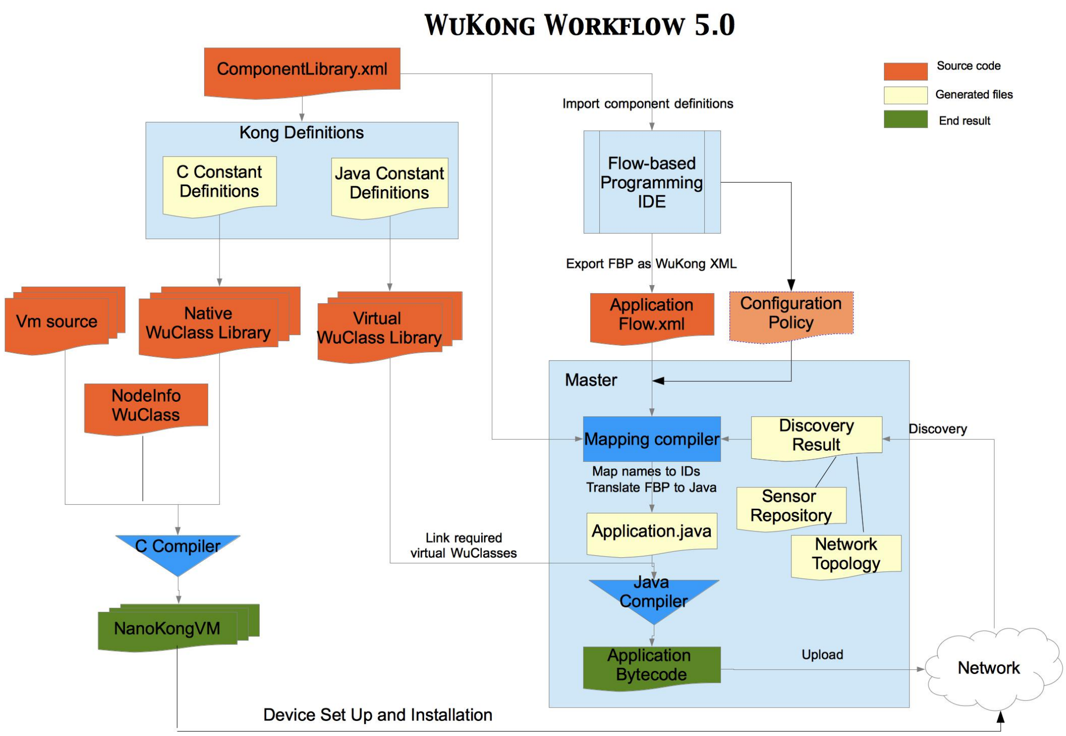


Figure 2.2 shows the overview of WuKong's build flow. The left part represents the build process for NanoKong VM which will be installed on the sensor nodes as part of the WuKong framework. The top part represents the build process for generating component libraries and Virtual WuClass library which will be used in other parts of the process. The right part illustrates the build process for FBP applications from being drawn in the IDE to Java bytecode that will be transmitted to the nodes.

The FBP program from the IDE will be exported as XML to the Master, the Master will then take this XML and passed to Mapper to generate a Java program that will be executed on the nodes. Finally, the compiled code is then wirelessly uploaded to the nodes in the network.

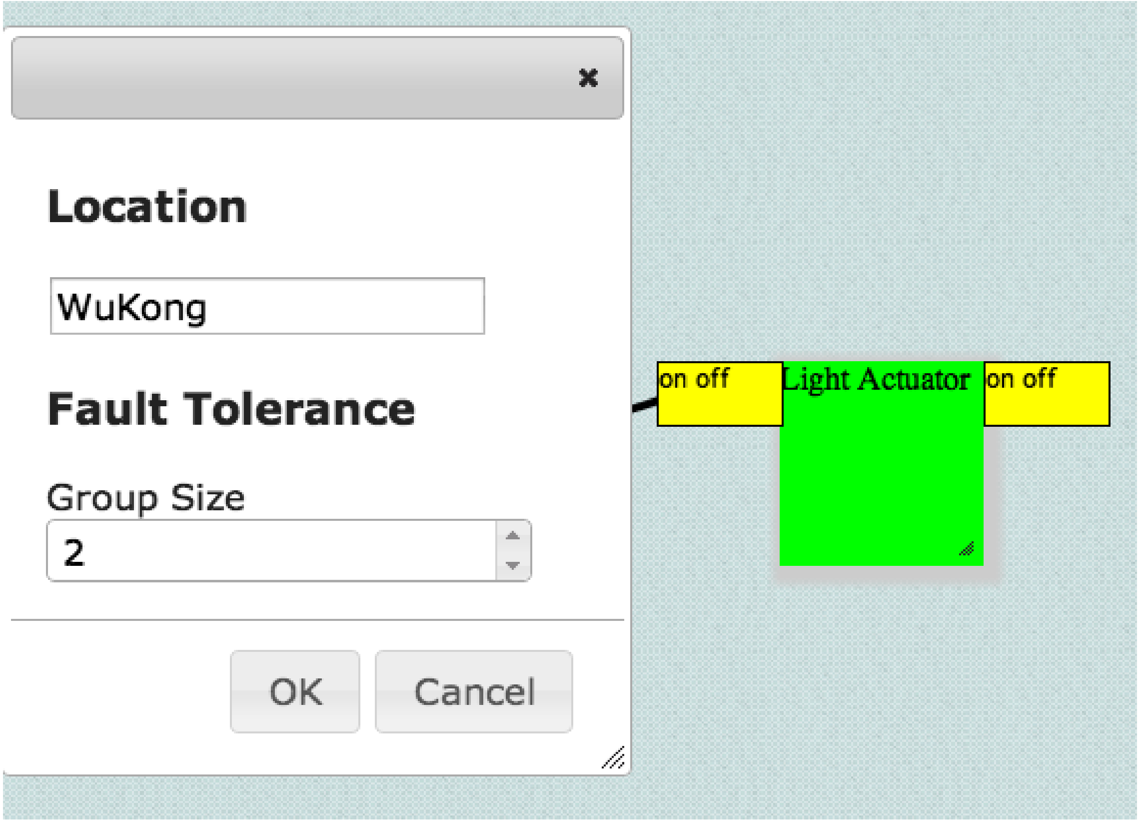
The Java code consists of many parts from different phases of the mapping process. First, the Java code contains information about links between components that were taken from the FBP XML passed in earlier from the IDE. A link contains the source component id, destination component id. The library code for components corresponding to the component ids are stored in the node if it is written in native language, or uploaded as part of the Java bytecode if it is written in Java language. Second, it contains information about the mapping from application component ids to actual node identifications and positions. The purpose of a mapping which separates from the actual link makes it easier to substitude the actual host of the WuObject later during reconfiguration from the Master. This mapping is created by the Master during discovery phase that probe the network for node's capabilities in terms of available WuClasses, then mapper will decide the final candidates that will be hosting for a component. If no native version of a component is found on the nodes, mapper will substitute with a Java version of it.

2.3.5 System Progression Framework

There are a few popular wireless communication protocols in M2M applications: ZigBee, ZWave. It is expected that in the future more diverse wireless nodes equipped with radios that support protocols such as low-power bluebooth and WiFi that all have one or more powerful gateway to connect to the outside world. In WuKong system, one of the gateways will take on the role of higher management decision maker called *Master* to making the decisions for deployment and producing the configuration of wireless sensor networks.

2.3.6 User Policy Framework

Figure 2.3: A user component policy dialog



Many M2M applications are heavily influenced by user preferences and current environmental context, as users and objects are mobile and application requirements and policy could change in time. Users are able to specify user policy for every component in the FBP and the application as a whole as illustrated in Figure 2.3

Fault Tolerance policy

M2M applications are inherently distributed, and hence it is inherently prone to failures since all nodes are running autonomously unattended for a long period of time where the external enviornmental influences could break and shut down the devices easily. Fault Tolerance policy enables users to specify relevent policy for tolerating failures in the granular level. This thesis will discuss more on fault tolerance policy in the following chapter.

Chapter 3

Related Work

3.1 Component Based Middleware

Distributed Systems such as Wireless Sensor Netweorks (WSNs) composed of multiple resource constrained devices equipped with low-power radios, low-cost sensors collaborating to perform tasks for applications in multiple discipline such as habitat monitoring. However, typically WSN applications are huge in scale, subject to unreliable network, high risk of node failures, and expected to run unattended for a long period of time, several work on lightweight component models have been proposed[5][7][6].

3.1.1 Lightweight Component Models

Work such as LooCI by Hughes et. al.[8] and Remora by Taherkordi et. al.[17] have demonstrated the feasibility of reconfigurable infrastructure with event-driven programming model, event management system, dynamic-binding to reduce development overhead, simplify abstractions, and easier, dynamic reconfiguration for distributed applications.

3.2 Fault Tolerance in Distributed Systems

Many systems has been proposed to provide fault tolerance in distributed systems, as the possibilities partial failures is a fundamental characterisics of distributed systems. However, distributed systems such as Wireless Sensor Networks typically involves a large number of resource-constrained devices equipped with various hardware components such as microprocessors, sensors, memory, wireless communication. An application rely on the collaboration among the sensor nodes in the network to perform tasks.

In order to prevent a single point of failure in applications that would bring application to a halt in the event of failure, one of the primary goals for most of the work for fault tolerance is to eliminiate single point of failure in an application, so replications, redundancies are recognized to be a good model for tackling the problems mentioned earlier in distributed systems. Replication and redundancy are both techniques that allow the system to duplicate multiple copies of a specific system component such that in the event of failure of one of the components, one of the other duplicated components can take over to perform the same services or functionality that the previous one provides.

3.2.1 Group Communication System

Some work emphasizes on providing a set of group communication protocols to ensure the information among a single replication or redundancy group could be synchronized[16]. By modeling replications and redundancies into a single system component unit in fault tolerant distributed applications, Sussman et. al. work on Virtual Synchrony could allow an efficient, distributed way to manage replications autonomously.

3.2.2 Multi-Agent System Resource Allocation

Luna et. al.[9] proposed a new approach to building reliable Multi-Agent Systems involves assigning agents to machines to manage resources, and by measuring the criticality of the components in real time to replicate only for critical agents under system resource constraints for fault tolerance.

3.2.3 Service-Oriented Architecture

Neumann et. al.[10] proposed a new redundancy infrastructure to bring Service-oriented Architecture (SOA) to Wireless Sensor Networks (WSN).

SOA is an established approach to ease development of complex distributed applications by encapsulating system compoennts into services, this allows a more flexible way to construct and develop interactions in application.

However, these approaches don't work well with applications where reconfigurability, interoperability, and heterogeneity are requirements. Existing approaches do not consider reconfigurability in the application level that might invalidate existing infrastructure built upon one instance of applications in another new instance when redeployed. Most existing approaches also are not efficient in providing reducing the development overhead to simplify fault tolerance in existing applications.

Chapter 4

Fur: An Intelligent Fault Tolerance System

This chapter describes how subsystems in Fur collaborate to provide fault tolerance for applications. Then we will move on to describing how WuKong Master takes this FBP and policy to convert into low-level intermediate representation ready to be deployed. Lastly, we describe how the sensor network detect, recognize, and recover from failures.

4.1 WuKong Applications

?? illustrates a typical WuKong application with components connecting to form a small network. What this figure shows is that applications are made of components, and components are connected through properties. Any pair of properties form links that binds components together. Connected components interact with each other based on the direction of the connection. The properties on the left of a component are inputs to this component, to the right side are the outputs of an component. Thus one can infer that a connection cannot connect to both inputs or outputs of any pair of components as both of them have the same data flow.

4.2 Redundancy

A typical fault tolerant distributed system is based on the concept of redundancy and distributed systems usually have advantages to have spare resources.

Spare resources addresses the first fundanmental characteristics of fault tolerance that there is no single point of failure.

Redundancy in WuKong would be to duplicate the nodes that host critical components with the intention to switch to one of the remaining instances in case of a failure.

However, heterogeneous sensor network imposes a challenge to design a fault tolerant system when compatible spare resources will have to be considered rather than narrowing for identical spare resources for redundancy.

4.3 Groups

I propose to use group abstraction to implement redundancy in WuKong application.

Any component in the application can thought of a collection of nodes that might be hosting this component. When the mapper in WuKong tries to find a mapping for a component, instead of looking for just one node, it will try to find a number of spare nodes to be part of the redundancy.

When a component needs to increase reliability, a group size is specified for that component in the FBP as shown in **??**.

Every component can set a number greater than one to indicate the intended size of the group of nodes which will be mapped to host this component. The component will be called a *component group*. The size of the group is an positive integer that cannot be less than 1.

4.3.1 Mapping for component groups

As illustrated in **??**, a mapping consists of a set of application components as shown as circles on the top of the figure, and a set of nodes with a list of capability that certain application components could map to.

An application component can only map to compatible nodes. Compatible nodes are nodes with features that match the desire features from the component profile. For example, a temperature sensor component requires temperature sensor feature. A node with a temperature sensor installed will have this feature and it will be a compatible node for this component.

A software application component can be mapped to any node that is capable of computation, but a hardware application component can only map to the nodes that has the features corresponded to the component profile.

A valid mapping is a mapping would have all components mapped to at least one compatible node or to the number of compatible nodes specified in group size, that's the desire output for the mapper.

Final algorithm

For the mapper to find a valid mapping, we have to modify the algorithm to consider multiple compatible nodes in questions at the same time. The 1 shown is used.

Algorithm 1 Generate Mapping with groups

0: test

4.4 Goal

4.5 Group abstraction

4.6 Policy

4.6.1 Fault tolerance category

4.7 Mapping

4.7.1 Matching

4.7.2 Translation

4.7.3 Compilation

4.7.4 Reprogramming

4.7.5 Reconfiguration

Member ranking

If application could specify a full ranking among a group, Whenever a leader died, a member has to replace its position. Leader identification, successor of current leader.

agent should specify whether this group has any ranking rules for a members. Whether there is a full ranking for current members and future members. This could lead to different actions reacting to similar events. ...

4.8 Agent architecture

This section will first go into details of how applications in a form of an abstract graph are being managed in a distributed system after being compiled and transformed into lower bytecode representation, then I will further discuss how the agent architecture on every node collaborate to form groups that will be the basic unit of redundancy in the application to detect sensor failures, diagnosis the failure, and finally recover from failure.

?? illustrates the proposed agent architecture in the sensor network. Every component in the application is converted into groups, which consists one or more nodes, and one of them is a leader. Every group has only one leader. Heartbeats are sent out from the members and leaders to monitor each other.

4.8.1 Autonomous Systems

Sensor networks composed of a large number of diverse subsystems. Subsystems intertwined with complex relationships that prohibit human intervention. Subsystems such as deployment, operation, reconfiguration, maintenance must be automated.

The inability, passiveness to errors makes the past systems unable to deal with perturbations, or unpredictable changes in the environment. Such systems know a limited amount of patterns and trigger predefined actions when they encounter these patterns. In order to make system adapt to new environment in a way similar to biological systems, they need to react to events as a whole in real-time.

4.8.2 Distributed agents

As our system consists of complex elements and subsystems mingled together, an appropriate way to handle complex behaviors in decentralized systems is to based it on a society of agents. [**?**]

4.8.3 Towards failure detection

As we mentioned earlier why sensor systems has to evolve to adapt to crutial, ever changing environment, one of the first things a system could achieve that goal is to detect failures autonomously.

Group membership

In our model, sensor networks consists of a diverse of sensor platforms, and some subset of sensor nodes are equipped with similar sensors situated near each other.

Since sensor networks are inheritly concurrent, it is very hard to reason about the states of the nodes in a distributed fashion. In order to provide fault tolerance in the system, a number of nodes need to be in sync and form a group to monitor and replace faulty node if necessary.

?? illustrates a running applications with three components: Temperature, Light and Threshold. Every component is implemented by a group of sensors hosting the same object that represents the component.

Every node consists of two agents, namely membership and controller as illustrated in the **??**. When application is deployed, every node is populated with a link table full of links and a list of objects representing the components on the application that are being assigned to.

Membership agent is there to maintain and update the membership list by also managing a watchlist and a reportlist to receive/send heartbeat from/to.

Heartbeat and node failure A failure in distributed system could come from different sources. Some nodes might fail because of software bugs; some nodes might fail because of poor wireless link quality caused by interference. One of the biggest challenges in distributed systems is to be able to detect the types of failures when it occurs correctly. Pulled between efficiency and reliability, decisions to make individual sensor nodes at the same time able to detect failures with knowledge of others but also have to reduce the amount of resources is essential for designing a effective distributed system.

There is another type of failures caused by a completely different reasons. Byzentine failures are failures inspired by the Byzentine General's Problem where components of a system fail in arbitrary ways (besides stopping or crashing) by processing requests incorrectly, corrupting local states, or producing incorrect or inconsistent outputs.

As we assume society of cooperative agents, every agent in the system will strive to be helpful, and share a common goal. This is strengthen by the fact that the agent goals' are bootstrapped from a common source which is the WuKong Master. However, nodes could still fail. In order to be consistent, we model failure by whether a node could send messages or not. This is called a fail-stop failure model.

A fail-stop failure model is a model in which sensor nodes are suspected of failure when they stop sending messages which could caused by multitude of reasons such as network partition, or software bugs.

Heartbeats are messages sent by individual nodes periodically to indicate to the monitoring nodes its health [].

?? illustrates some nodes sending heartbeats that detects a failure when a number of expected consecutive heartbeats have not been received.

A node is considered dead when it has not been sending heartbeats for more than 2 rounds of timeout.

Group setup The WuKong Master has already assigned an ordered list of nodes for every component, every node would be able to know who are in their group for any particular component.

Wielded with this knowledge, membership agent will setup all heartbeat links between the nodes if not already.

Since the setup is asynchronous without lockstep, all nodes are free to do whatever they want while others are being programmed. We added a random backoff after the nodes are programmed to prevent the problem of nodes reporting failure when they startup, because when the node finished setup, the nodes it is watching might not be finished reprogramming, thus it will delay its heartbeat and exceed the timeout causing the node to send false alarm.

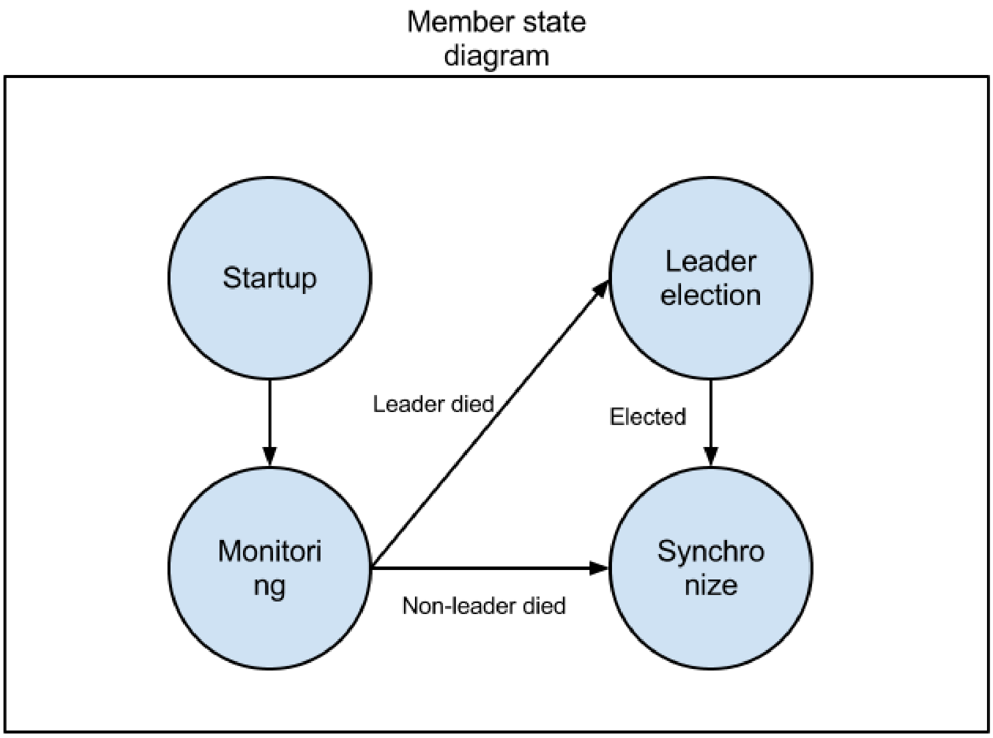
As illustrated by **??**, every node has a leader which is elected when the group is formed.

Connecting them up

When a node failed, or has not sent heartbeats for too long, a node failure will be picked up by the monitoring nodes. However, a system is not fault tolerant if it cannot decide who could be replacing the failing component. So there must be some way to organize groups such that when a failure is detected, a replacement could be decided.

It is possible that there will be multiple failures occurred in a timely fashion. We need to be able to detect all possible failures within a group, so we need a heartbeat network.

Figure 4.1: Node states



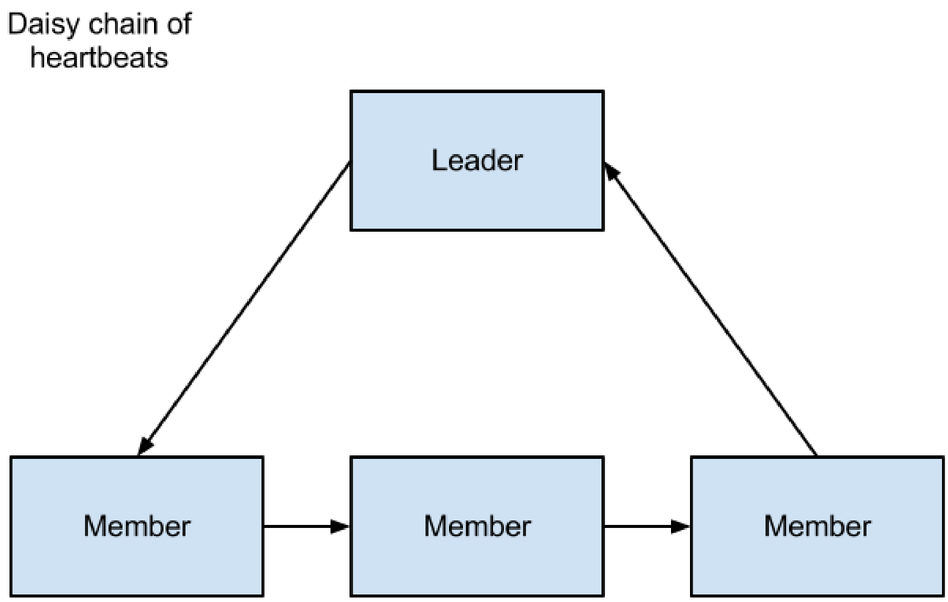
Of course the structure of heartbeat communication pattern highly depends on the underlying network assumption and infrastructure of the application. To produce the most efficient network with the least connections, heartbeat network has to satisfy two properties.

1. Every node has to monitor at least one node other than itself

2. Every node can only has one node monitoring itself

A heartbeat network in the form of a daisy chain is one of the networks that satisfy both properties as shown in 4.8.3. Every daisy chain heartbeat network monitors all nodes in the group, given by the properties that every node has to monitor at least one node and every node can only have one node monitor itself. So if there is n nodes, there can be at most n nodes being monitored, every monitoring node can only monitor one node, every node can only monitor node that others have not, thus every node is monitored by only one unique node. Thus this daisy chain guarantees every failure can be detected.

Figure 4.2: Daisy Chain of heartbeats



Message complexity Every heartbeat takes one message to send from one to another. As of current, we assume every heartbeat is sent using unicast, and there is no ACK for heartbeat messages. The message complexity for standard one-hop star network takes about O(2n-2) messages since the leader sends n-1 to every member and every member would also has to send a message back to leader, that comes to double of the single traversal from leader to other members.

The message complexity for the daisy loop takes about O(n) messages for a group, because every member only sends one heartbeat to one other member at a time including the leader.

4.8.4 Failure recovery

When a node failed, it is guaranteed that at least one node will detect this. However, the chain is broken, and it would be impossible for the membership agent in the node to decide what the subsequent actions could be. They are not designed to do such things. There need to have another agent to decide based on local information the actions to take to resolve this issue.

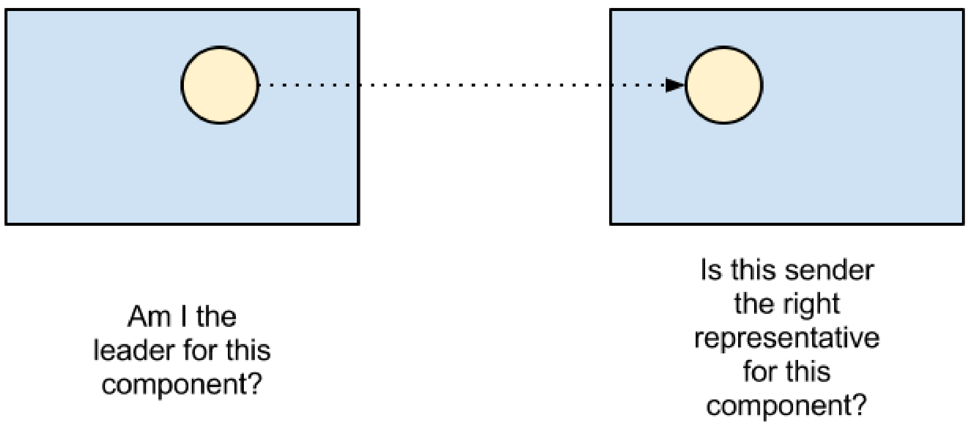
Link table entry protocol

When the application is deployed, every operating node contains at least one object that represents one component in the application. To connect the objects just like what it did in the application as shown in FBP needs a table to store the information of the links. Link table is broken down into two layers, where the top layer gives the information between two components, and the bottom layer gives a list of nodes that are part of the same group that could provide the service for operating the application. By breaking into two layers gives us immense flexibility to handle many different kinds of graph that the FBP could produce.

As illustrated in 4.8.4, the premise of a link between components rely on both end's agreement. If either one of them disagree, then the link is not valid.

When a leader of a component failed, the elected new leader will have to update everyone's link table so they all are in consensus of who is currently the representative for this component. The result of the protocol affects the lower layer of the link table, so all nodes in the network affiliated with a particular component could be assure that they are not sending messages to nodes that doesn't exist or listening for messages from nodes that are not in charge of sending the sensor values.

Figure 4.3: Link table switching



Controller agent

We have described a part of the system that does membership, fault detection. However with only those parts the system, they do not form a fault tolerant system. Without a controller making a right decision at the right time, to synchronize group states, groups will not have consensus and will not be able to proceed as a whole **??**. Controller's responsibilities range from responding to failure events, synchronizing states to group members and connected nodes to ensure the application could still operate.

Reactions and actions tables are set up during deployment. Controller will be mainly handling events and handle according to the rules defined in the tables.

Member ranking Most other work on failure recovery relies on leader election to elect a new leader when the old leader failed which is followed by a group multicast to ensure group state is synchronized to all members.

However, leader election is not necessary if a full member ranking could be determined. Member ranking serves as a lookup table to successor to any node if it happens to fail, also used to recover heartbeat network.

Responding to failure **??** describes the dynamics between agents when a failure has been detected.

When a member detected a failure from other member, the membership agent on the member node will notify local controller agent of this event.

By default, controller agent will notify the leader of the group if it is not, the leader's controller agent will initiate synchronization protocol to synchronize members' membership list.

If the leader failed, the controller agent of the monitoring node will initiate the leader election protocol and become a new leader itself.

Leader election When there is no clear ranking among members, a leader election could provide a mechanism for determine the successor of a failed node.

Recover from failure *Membership synchronization*

Some actions will be syncing membership list across all members for a particular group. This protocol is triggered either by the reaction table above or by explicit request from other agents, such as the Membership agent. Controller agent will access the current membership table and instruct Synchronization agent to start the membership list synchronization.

Link table synchronization

Once the fault has been confirmed and has shunned the faulty node from the network, GMS coordinator will be responsible for propagating this event to other groups in the system that has subscribe to this event.

Let's say that a group has a simple one hop star topology, so all nodes will transmit its data to the group leader, when the group leader has been reported to be partitioned away from the network or failed, then what will happen is that each group member will elect a new leader by some heuristics, since it has the link table for the previous group leader, it will know which node it is suppose to link to outside the group upstream, and change its local link table accordingly, the new leader will determine the new linking table of its members by consulting the group policy, send a multicast to the group to route all their data to the new leader. The upstream node should either be inside a group managed by the GMS, or a Master assigned gateway, it should also receive this update according to the group policy which GMS will also inform them of.

Essentially, GMS is global in the network and will have full knowledge of the group policy for all groups in the application, and it can follow the group policy to propagation fault events to the appropriate groups.

False positive fault detection

It appears to be possible to have false positive fault detection when a node is not dead but actually got partitioned away from the network for a short period of time. If it is the leader that got partitioned away for too long, several members will be detecting this failure and they might all initiate a leader election. Since they all know of this situation, every node detecting a failure will wait for a random amount of time before sending the message. If a leader election message has been received, it will terminate its current action and continue to the second phase of the leader election process.

However, it is possible that leader is not actually dead, and it is also monitoring the members. The leader might conclude that the members are all gone and will also generate a failure event (since there is no one to synchronize to). This is a split brain problem because the remaining members will elect a new leader and proceed in synchronizing the link table in neighbor nodes, but the old leader is still operating and sending data between the neighbor nodes, this will create a conflict both in the group and cause a confusion among the outsiders.

Assuming both partitions can talk to the neighbor nodes with objects connected to their objects, there is no way for the partitions to detect the problem within themselves but only the outsiders.

The outsiders, whose objects are connected to the group, will be the fault detector and will notify both leaders of their existence along with their scoring. The leader with less scoring will give up their leadership, and try to merge with the other partition if possible.

If it is still not possible after a timeout, it will try to notify the Master of this situation.

Daisy chaining

Since it is also possible that heartbeat is in daisy chain that the any node only monitors one node at a time, and no two nodes monitor the same node. When that happens it is still possible that leader could partition away from the system and appear again later in time. Since the new leader is the one monitoring the old leader, when the old leader resume and start sending heartbeat, the new leader could be sending a reconfiguration message to Master, or if it is not severe enough to do a full reconfiguration, it could resign by sending the old leader a resign message to inform the leader to reconfigure its connected objects about this change of leadership, and it will resume to become a normal member again.

The message complexity for the operation of resignation should be O(2+2H(m)) where H is a function that returns the number of nodes hosting connected objects m.

Misc. need review

Reaction table Similar to a routing table, Controller agent will also have a table to look up which actions it could take given a certain type of failure event.

Source	Detector	Event	Action id
"all members"	"any member"	no heartbeat	0
"all members"	"any leader"	no heartbeat	0
"leader"	"any member"	no heartbeat	4

Table 4.1: An example of reaction table

Action table This table list all actions that a controller could do. This table follow similar style described in the work of ADAE with rules that provides graceful degradation for most actions.

Id	Profile type	Next	Secondary	Doer	Receiver	Fun
0	standard	3	1	“leader”	“all members”	“me
1	secondary			“leader”	“operating members”	“me
2	standard			“any member”	“leader”	“ser
3	standard			“leader”	“all members and connected nodes”	“lin
4	standard			“operating members”	“operating members”	“lea

Table 4.2: An example of action table

When a standard action failed, the secondary action will be triggered, for example, if action 0 failed, action 1 will take over since it is a secondary of action 0. When action 0 finish execution, action 3 will follow it immediately since it is a next of action 0.

Event queue Controller will be bombarded by lots of events coming from multiple sources in a short period of time, to ensure all events are stored in chronological or importance order, a event queue is used. Any event that other agents send to notify controller will be stored in this queue.

Event name	Priority
“No heartbeat”	0
“Event queue synchronization”	1

Table 4.3: Event queue

Event queue synchronization

It is very likely that if nodes failed, the events carried on the nodes will never see the light of day in other nodes, and will never be handled. To prevent missing events, controller will have to synchronize event queue among members.

Preemptive event queue

Some events have higher priority, and those events should come in in sparse intervals. Event queue synchronization is an important event that any controller should preempted before other unhandled events.

Chapter 5

Experimental Design and Result

5.1 Experimental Setup

5.1.1 Sensor Nodes

5.1.2 Fault Tolerant Policy for FBP

Chapter 6

Conclusion

dkljfa;lkfj; sd;lkf j

6.1 Summary of Contribution

Chapter 7

Future Work

Group connection types

Mapping

Policy When application are getting complex full with features and configurations, it is important to have a high level declarative configuration policy language to specify the control for features and control of their respective behaviors smoothly. I propose a high level policy for fault tolerance that could be translated to low level application requirements.

Bibliography

[1] V. A. Archana Bharathidasan. Sensor Networks: An Overview.

[2] A. Arora, P. Dutta, S. Bapat, and V. Kulathumani.... A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, Jan. 2004.

[3] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. In *Proceedings of the 6th ACM conference on Embedded network sensor systems SenSys 08*, volume D, pages 43–56. ACM Press, 2008.

[4] K. P. Birman. Dynamic Membership. In *Guide to Reliable Distributed Systems*, chapter 10, pages 339–367. Springer London, 2012.

[5] P. Costa, G. Coulson, and R. Gold. The RUNES middleware for networked embedded systems and its application in a disaster management scenario. *..., 2007. PerCom'07. ...*, pages 69–78, 2007.

[6] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan. A generic component model for building systems software. *ACM Transactions on Computer Systems*, 26(1):1–42, Feb. 2008.

[7] D. Gay, P. Levis, R. Von Behren, and M. Welsh.... The nesC language: A holistic approach to networked embedded systems. *Proceedings of the ...*, Jan. 2003.

[8] D. Hughes, K. Thoelen, J. Maerien, N. Matthys, W. Horre, J. Del Cid, C. Huygens, S. Michiels, and W. Joosen. LooCI: The Loosely-coupled Component Infrastructure. In *2012 IEEE 11th International Symposium on Network Computing and Applications*, pages 236–243. IEEE, Aug. 2012.

[9] A. D. Luna, S. Aknine, and J. Briot. Dynamic resource allocation heuristics for providing fault tolerance in multi-agent systems. *... of the 2008 ACM symposium on ...*, 2008.

[10] J. Neumann, N. Hoeller, C. Reinke, and V. Linnemann. Redundancy Infrastructure for Service-Oriented Wireless Sensor Networks. In *2010 Ninth IEEE International Symposium on Network Computing and Applications*, pages 269–274. IEEE, July 2010.

[11] P. Padhy, K. Martinez, A. Riddoch, H. L. R. Ong, and J. K. Hart. Glacial Environment Monitoring using Sensor Networks. *RealWSN*, pages 10–14, 2005.

[12] J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson. Analysis of wireless sensor networks for habitat monitoring. In C. S. Raghavendra, K. M. Sivalingam, and T. Znati, editors, *Wireless Sensor Networks*, chapter 18, pages 399–423. Kluwer Academic Publishers, 2004.

[13] N. Reijers, K.-j. Lin, Y.-c. Wang, C.-s. Shih, and J. Y. Hsu. Design of an Intelligent Middleware for Flexible Sensor Configuration in M2M Systems. *Sensornets*, 2013.

[14] I. Stoianov, L. Nachman, S. Madden, T. Tokmouline, and M. Csail. PIPENET: A Wireless Sensor Network for Pipeline Monitoring, 2007.

[15] R. Sugihara and R. Gupta. Programming models for sensor networks: A survey. *ACM Transactions on Sensor Networks (TOSN)*, V:1–27, 2008.

[16] J. Sussman, I. Keidar, and K. Marzullo. Optimistic Virtual Synchrony. *Reliable Distributed Systems, ...*, (914), 2000.

[17] A. Taherkordi and F. Loiret. Programming sensor networks using REMORA component model. *... Computing in Sensor ...*, pages 1–14, 2010.

[18] J. Tateson, C. Roadknight, A. Gonzalez, T. Khan, S. Fitz, I. Henning, N. Boyd, C. Vincent, and I. Marshall. Real World Issues in Deploying a Wireless Sensor Network for Oceanography. *REALWSN 2005*, 2005.

[19] G. Tolle, J. Polastre, R. Szewczyk, and D. Culler.... A macroscope in the redwoods. *Proceedings of the ...*, Jan. 2005.

[20] G. Werner-Allen, K. Lorincz, and J. Johnson.... Fidelity and yield in a volcano monitoring sensor network. *Proceedings of the ...*, Jan. 2006.

[21] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks.