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

口試委員會審定書 **i**

Acknowledgments **iii**

致謝 **v**

中文摘要 **vii**

Abstract **ix**

1 Introduction **1**

1.1 Motivation 1

1.1.1 Wireless Sensor Network Deployment is Hard 1

1.1.2 Maintaining WSN deployment 2

1.1.3 Component based middleware for distributed applications 3

1.2 Problem Definition 4

1.2.1 Service Availability for Component Systems 4

1.3 Approach 4

1.3.1 Related Work 6

1.4 Thesis Organization 7

2 Background **9**

2.1 Wireless Sensor Networks 9

2.1.1 Redundancy 9

2.2 Component Based Middleware 10

2.3 WuKong: The intelligent middleware for M2M applications 10

2.3.1 Goal 10

2.3.2 Flow Based Programming 11

2.3.3 Sensor Profile Framework 13

2.3.4 Compilation and Mapping 14

2.3.5 System Progression Framework 16

2.3.6 User Policy Framework 16

3 Reconfigurable Atomic Service for Component Objects 19

3.1 Reconfigurable Service System 19
3.2 Profile Framework 20
3.3 Strips 20
3.4 Decentralized Failure Detection 22
3.5 Failure Recovery 23
3.5.1 Reconfiguration 23

4 Deployment of Reconfigurable Atomic Service for Component Objects 25

4.1 Reconfigurable Atomic Service for Component Object 25
4.2 Policy Framework 25
4.2.1 Fault Tolerance Policy 25
4.3 Set Covering Problem 25
4.4 Greedy Approximation Method 25
4.5 Hybrid Method 25
4.6 Discussion 25

5 Conclusion 27

5.1 Summary of Contribution 27
5.2 Discussion 27
5.2.1 Deployment 27
5.3 Future Work 27
5.3.1 Questions 27
5.3.2 False positive fault detection 28
5.3.3 Group connection types 29
5.3.4 Mapping 29
5.3.5 Policy 29

Bibliography 31

List of Figures

2.1 A FBP application 11
2.2 WuKong application build flow 15
2.3 A user component policy dialog 17

3.1 An example network with several strips 21

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, and how an intelligent middleware which employs a new programming model which separates design abstractions between high-level application design and low-level hardware constructs could be addressing the challenge. Next, we describe the needs for fault tolerance and why it is hard on such middleware. 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][7][2][11][6][9][12][13].

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.[14][5]

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[10]. 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[8].

1.2 Problem Definition

1.2.1 Service Availability for Component Systems

Overview: State the fault tolerance problem for WuKong

Loosely-coupled Component Based Systems for M2M Applications

Overview: Motivates the problem in M2M systems, the unique situations that leads to RASCO and SetCovering algorithms, namely service availability, and deployment problem

1.3 Approach

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.[8]

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.

1.3.1 Related Work

RAMBO: A Reconfigurable Atomic Memory Service for Dynamic Networks

ESDS: Eventually-Serializable Data Services

Service-Oriented Architecture

Neumann et. al.[5] 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.

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

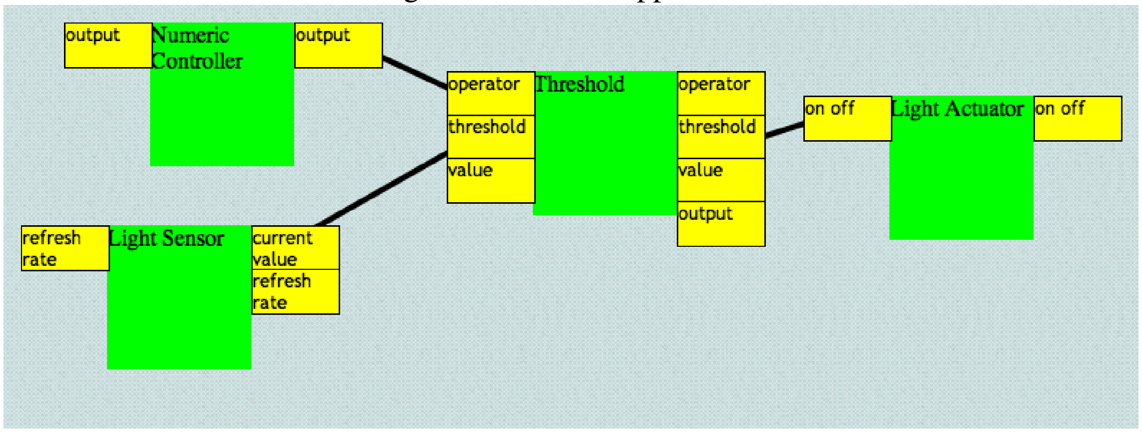
Deployment and development for M2M applications are in its infancy today. As many applications are still single purpose in homogeneous networks with specific network protocols. The hardware has a fixed range of sensors, and the applications cannot be easily ported to other platforms.

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[8].

2.3.2 Flow Based Programming

Figure 2.1: A FBP application



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 components 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

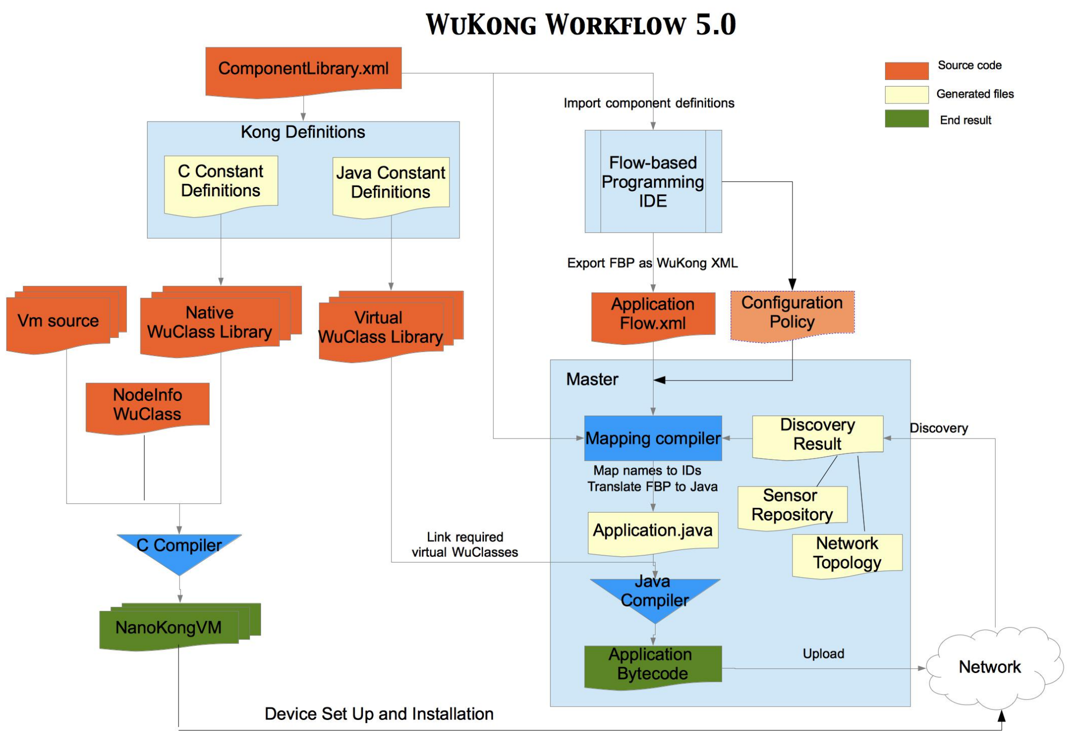
The profile framework is in charge of communication between WuObjects as well, which are not necessarily on the same nodes. Profile Framework monitors the changes in properties and propagate the changes to the connected WuObjects. For example, if a Temperature WuObject is connected to a Threshold WuObject, 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 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

Figure 2.2: WuKong application build flow



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.

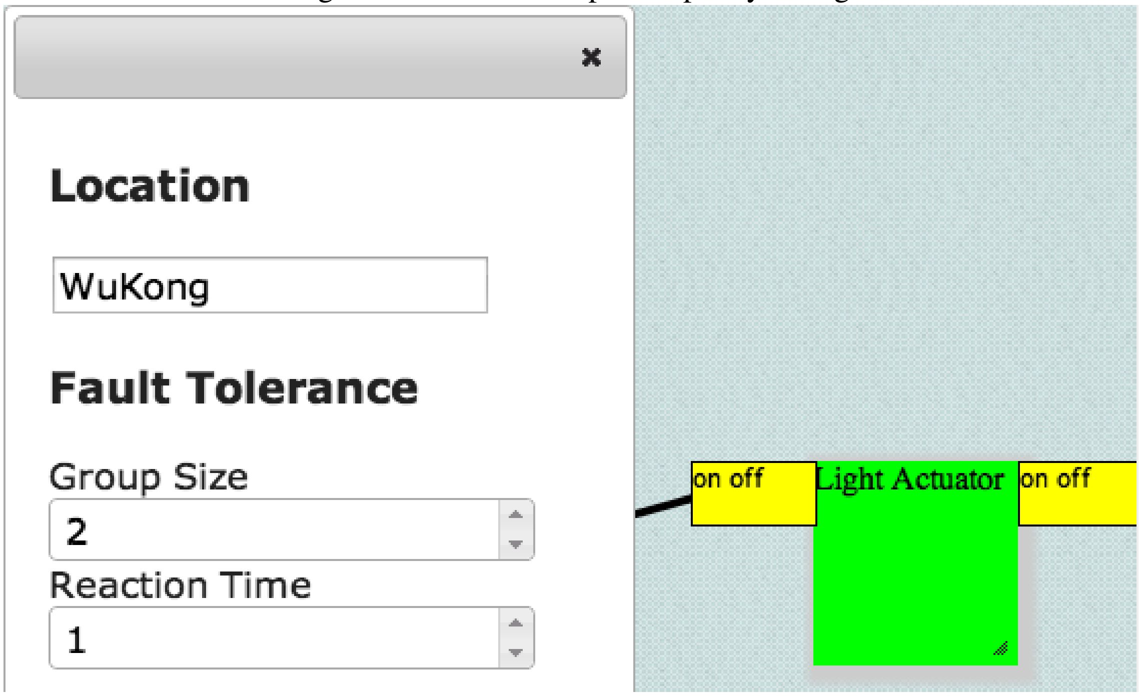
2.3.5 System Progression Framework

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

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

Figure 2.3: A user component policy dialog


Location

WuKong

Fault Tolerance

Group Size

2

Reaction Time

1

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

Reconfigurable Atomic Service for Component Objects

This chapter presents the Reconfigurable Atomic Service for Component Objects, RASCO in short. First the reconfigurable service problem is described in section 3.1 and the profile framework is described in section 3.2. Section 3.3 describes a new replicas configuration models called strips to track and handle replicas dynamically. After that section 3.4, 3.5 and 3.5.1 presents the distributed algorithm used to solve the problem. The models and algorithms are tested extensively on various benchmarks, described in section **??**, and the results are discussed and compared with existing fault tolerant system models in section **??**

3.1 Reconfigurable Service System

Data replication is a fundamental technique to distributed systems as it improves availability, eliminiates single points of failure. A system that is hardwired to get data from node X will fail when X fails. We want to design a system where node Y, which can provide the same service, can take over when X fails.

Reconfigurable Atomic Service for Component Objects (RASCO) is designed to solve this problem by introducing a new replication model. RASCO allows in situ reconfiguration for maintaining consistency between services and replicas.

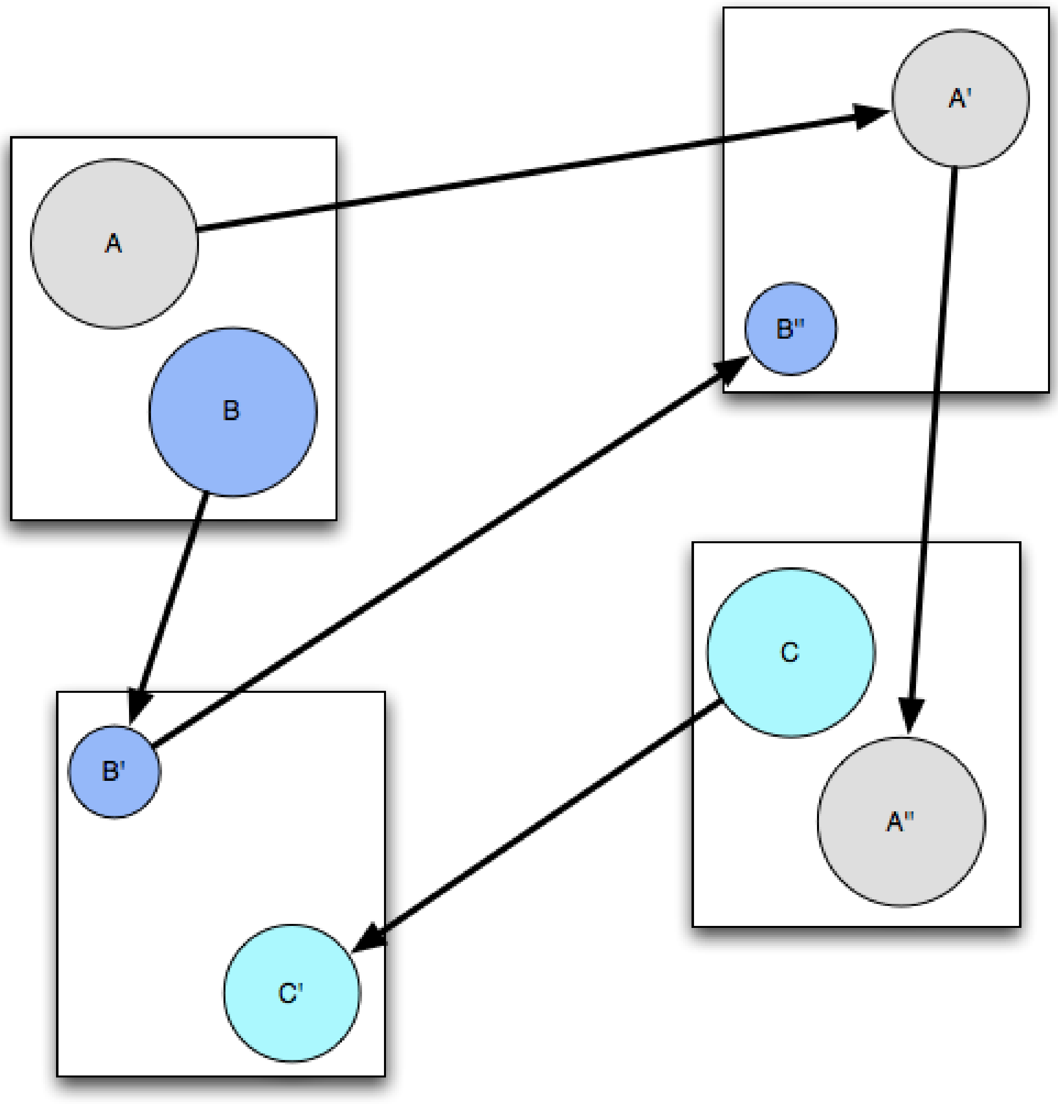
3.2 Profile Framework

In this work, we build upon WuKong, a loosely-coupled component based architecture for M2M systems. WuKong uses profile framework to enable the handling of physical resources on heterogeneous sensor nodes, and for higher abstractions of software component capabilities. As future M2M systems could consist of many heterogeneous sensor nodes and actuator nodes, two main concepts in profile framework, namely WuClass and WuObject, was introduced to allow WuKong to track, and manage physical resources in the network. [8] However WuKong has no support for fault tolerance. We proposed a solution to track, manage and maintain consistency among replicas based on concepts from profile framework.

3.3 Strips

This work proposed an algorithm that uses strips, each of which consists of service holders prioritized by some metrics to keep track of changes and maintain service availability in the presence of failures. Strips consists of members which are chained together in series to the next that when one member failed, the next one will take over, except the last one. For example, for a strip constructed like this $\rightarrow 1-2-3-4-5$, when 3 failed 4 will take over the place of 3 and shift all the objects after it forward, and the new chain will look like this: $\rightarrow 1-2-4-5$. Now if 1 failed, 2 will take over and members after 2 (including 2) will shift one position forward that would result in $\rightarrow 2-4-5$. Typically the head of the strips is the component in the application, while the rest are replicas.

Figure 3.1: An example network with several strips



In a heterogeneous network, each host could carry more than one WuObjects and could provide more than one service. Since each strip represents a specific component in the application, there will typically be many of these strips present in the network where each of them could crisscross with one and others. It would mean that a node could be a service provider and a replica at the same time.

A node stores membership information of the strips where it is a member of and also the strips of the nodes it is monitoring. Each stripe is stored as a list with membership address information in the same order as the order of the strips so for example, the current head of the strip is the first element of the list. Nodes use the information stored to track and notify the nodes for any changes in the strips.

Figure 3.1 illustrates a network with many strips as strips crisscross and layout in the network. Each block represents a host, and each circle is a component. The name of the component represents the type of the component, and replicas have the same name as the component but with an apostrophe next to it. As shown in the figure, each strip could crisscoss and could have replicas residing with another component within the same host.

3.4 Decentralized Failure Detection

The system assumes a fully connected heterogeneous network with sensors and actuators where each node in the network could host multiple components. This work uses heartbeat technique, a popular technique widely used to detect failures in high-availability distributed systems. Each node would send a heartbeat message to its detector periodically in a fixed interval until it's unable to send messages anymore. Each node is therefore suspected dead when it stops sending messages after a period of time.

There are many related work on heartbeat protocols to ensure high-availability whether it uses star topology, or a ring topology; the main purpose for a heartbeat protocol is to detect failure within a network as fast as possible. Our work assumed a ring topology heartbeat protocol such that a node A would send heartbeat to node B and so on, but the last node would send heartbeat back to node A.

3.5 Failure Recovery

Several strips would be cut off in the event of failure, and if the failed node carries the component of some strips, the system which hardwire the location of the failed component would not able to continue to function. Every node needs to have knowledge of the members of the strips of the nodes it is monitoring. For example, if node A is monitoring node B, A would know the members of all strips in node B in addition to its local strips. In this case, the detected node will prepare a update message to inform all members of the strips with which the failed node is associated with. Assuming that every node that monitors other node will have knowledge of the strips that it contains and the members that the strips pertain. The node would send out a marker multicast first to confirm the nodes which are still functioning, and once all acknowledges have been received, it will proceed to send the update message to update their local knowledge of the strips to reach a consensus. The ordering of the messages wouldn't matter since the end state of any failure sequence for any strip would be the same. For example, given a strip of three members $\rightarrow 1 - 2 - 3$, if the updated failure sequence is given in any permutation by $[1, 2]$ or $[2, 1]$, the end results would be the same $\rightarrow 3$ since the remaining members from those two failure sequence is the same and the relative order of the members would stay the same. Therefore there is no need for extra communication overhead to maintain ordering to gaurantee level of consistency between members since they will all come to the same conclusion given each receiver receive the same messages.

3.5.1 Reconfiguration

Even though consensus of the new configuration for each affected strip has been reached, some nodes with component that acts as a client to another component for data or the reverse would also have to have consensus on the updated primary holder of each affected component. And the node that is monitoring the node monitored by the dead node would also need to update on its knowledge of its monitored node in order to recover the next possible failure.

RASCO will initiate a reconfiguration service to reconfigure the network to adapt to the new strip configurations. Reconfiguration service is implemented by a distributed algorithm which is inititated by the detected nodes. First the initiator node would have to identify the components that are reading/writing data with the components carried by the dead host. This would be done by requesting the link information between components at the higher level provided by the application. Then each member of the strips of the connected components would be updated with the information about the change in the host of the failed components.

Reconfiguration service has message complexity of $O(m)$ where m is the number of strips where its components are linked to the failed components.

Chapter 4

Deployment of Reconfigurable Atomic Service for Component Objects

4.1 Reconfigurable Atomic Service for Component Object

4.2 Policy Framework

4.2.1 Fault Tolerance Policy

4.3 Set Covering Problem

4.4 Greedy Approximation Method

4.5 Hybrid Method

4.6 Discussion

Chapter 5

Conclusion

5.1 Summary of Contribution

5.2 Discussion

5.2.1 Deployment

5.3 Future Work

5.3.1 Questions

Q: How does the system recover from two consequtive failures in the network?

A: We assume a much simpler failure model where there can be at most one failure happen at any point in time. It is a reasonable assumption given that our system recover from failures fairly quickly (2 secs) ....

Q: How does an application is being compiled into low level bytecode? And when and how does WuKong reconfigure the nodes?

Q: What happen when an application is deployed? What is specifially at work?

5.3.2 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.

5.3.3 Group connection types

5.3.4 Mapping

Q: Inclusion problem, how to solve it? How severe it is?

A: The problem is due to the low redundancy level of a compoennt which is mapped to a node with other compoennts which have higher redundancy levels. This problem does not always have a solution, it depends highly upon the resources in the network, it is unavoidable. However, we are considering possible solutions to address this problem in the future by making sure the mapping results does not have this pattern and could generate possible solutions to solve this problem by either increase the redundancy level of the vulnerable component or purposefully avoid mapping to dangerous nodes that would put the component to risk.

But the bigger question is, if compoent FT policy is enough to ensure application durability, at the current state, if any component does not have minimum redundancy level of 2, that application is in the risk of single point of fialure, there would be needed a way to inspect whether the policy is fault tolerance at the application level, or introducing several application level FT policy and that is something that we could be looking into in our next research work.

5.3.5 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] 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.

[6] 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.

[7] 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.

[8] 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.

[9] I. Stoianov, L. Nachman, S. Madden, T. Tokmouline, and M. Csail. PIPENET: A Wireless Sensor Network for Pipeline Monitoring. *6th International Symposium on Information Processing in Sensor Networks, 2007. IPSN 2007.*, pages 264–273, 2007.

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

[11] 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.

[12] G. Tolle, J. Polastre, R. Szewczyk, and D. Culler.... A macroscope in the redwoods. *Proceedings of the 3rd international conference on Embedded networked sensor systems SenSys '05*, Jan. 2005.

[13] G. Werner-Allen, K. Lorincz, and J. Johnson.... Fidelity and yield in a volcano monitoring sensor network. *Proceedings of the 7th symposium on Operating systems design and implementation OSDI '06*, pages 381–396, Jan. 2006.

[14] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks. *Proceedings of the 2nd international conference on Mobile systems, applications, and services MobiSys '04*, pages 99–110, 2004.