I-ADMIN: A FRAMEWORK FOR DERIVING ADAPTIVE SERVICE CONFIGURATION IN WIRELESS SMART SENSOR NETWORKS

BY

PARYA MOINZADEH

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

ABSTRACT

Facilitating application development for distributed systems has been the focus of much research. Composing an application from existing components can simplify software development and has been adopted in a number of domains such as wireless sensor networks, mobile computing, ubiquitous systems, cloud computing, etc. Efficient application development in wireless smart sensor networks (WSSNs) generally faces more restrictions and is the focus of this thesis. Inherent limitations of wireless sensor networks such as memory size, bandwidth, computational capacity, and energy have driven WSSN application development towards low-level programming approaches which provide efficiency but hinder sharing and reuse. Varying environmental conditions, faults, and changing application requirements are also common in long-term deployments of WSSNs. Environmental conditions and faults are important considerations in this domain since they can affect the availability of resources such as energy. For example, a stretch of cloudy weather can affect the energy availability of sensor nodes that are equipped with solar panels. On the other hand, requirements of WSSN applications vary considerably and can include energy consumption, time synchronization error, packet loss, etc. The increased dynamicity and complexity of WSSN applications require open systems that interact with their environment while addressing application constraints and hardware limitations.

Our goal is to facilitate WSSN application development by allowing component sharing and reuse and dynamicity. Due to the importance of energy management on the lifespan of WSSN applications, our primary focus is on optimizing energy consumption while satisfying constraints that are derived from application requirements.

We model applications as a composition of services. Services are self-contained software components with self-describing interfaces that represent their inputs and outputs as well as their non-functional properties. We illustrate the need for service sharing and dynamic service composition and their challenges through examples of real-world applications, namely structural health monitoring (SHM) and environmental and agricultural monitoring. In fact, our experience in the design, development, and implementation of these applications resulted in our effort to build a framework that facilitates software development for WSSN applications. We have developed middleware services that are deployed in two main testbeds. On the first testbed, the Jindo Bridge in Korea, 113 nodes are deployed for long-term monitoring of structural health. The second testbed aims at environmental observation (soil moisture and nitrate) in a 40 acre field in Champaign, Illinois that has 4 types of vegetation.

The proposed solutions can be divided into three parts. First, we design a framework called I-AdMiN, which provides component deployment to enable dynamic service composition and adaptive reconfiguration, while respecting the resource constraints and efficiency requirements of wireless sensor networks. Second, we address the effect of deployment characteristics and environmental conditions by dynamically deriving energy characteristics of services that comprise the WSSN application. This is done in a component called *Monitor* by using aggregate information on system energy consumption. Dynamic and on-line profiling of services is important for two main reasons: i) many service characteristics such as energy consumption cannot be accurately determined until the full-scale deployment of the service, and ii) dependency relationships between different services and between the hardware platform and services can affect the overall behavior of the system and must be taken into account in the course of service selection. Many such dependencies cannot be determined apriori and depend on the environment and run time characteristics. Finally, we design and implement a system called *S4* to enable automatic selection of components and parameters to satisfy application requirements. S4 derives a constraint satisfaction problem from application constraints and service specifications and solves it to derive a selection of available services that form the application. Whenever available, S4 leverages dynamic information from the Monitor on service energy characteristics to optimize the energy consumption of the sensor network.

In memory of my grandmother, Fatemeh Kashani.
To my dear family.

ACKNOWLEDGMENTS

I am deeply grateful to my advisor, Professor Gul Agha, for his patient guidance and invaluable support throughout my graduate studies. His knowledge, vision, and moral support have been invaluable on both academic and personal levels. I would like to express my sincere gratitude to Professor Tarek Abdelzaher for his contributions as a thesis committee member and research collaborator. I would also like to thank my other committee members, Professor Nitin Vaidya and Dr. Ranveer Chandra for their valuable feedback on this work. In particular, Dr. Chandra's mentoring during my 7-month internship at Microsoft Research had a significant impact on my academic development. I am also grateful to Dr. Yinglian Xie for all I learned from her during my internships at Microsoft Research. I would like to thank Professor Bill Spencer, with whom I have collaborated on structural health monitoring.

My deepest appreciation goes to my colleagues and collaborators who have made this dissertation possible. I would especially like to thank Dr. Kirill Mechitov, with whom I worked at the Open Systems Laboratory at the University of Illinois. His vast knowledge of distributed systems and his support have greatly helped me along the way. His ideas particularly influenced Chapters 3, 4, and 6. Chapter 4 was done in collaboration with Kirill Mechitov and will appear in subsequent papers. I would also like to express my gratitude to Reza Shiftehfar, with whom I have worked on many occasions, including the energy management middleware service presented in Chapter 3. I would like to thank my group members and collaborators at the Open Systems Laboratory and the Smart Structures Technology Laboratory: Rajesh Kumar Karmani, Vijay Anand Korthikanti, Peter Dinges, Ashish Vulimiri, Minas Charalambides, Shinae Jang, Hongki Jo, Jian Li, and Robin Kim.

I am deeply grateful for the love and support of my parents, Maryam and Kazem Moinzadeh, and my sisters, Ramona and Pardis. This thesis would not have been possible without their help and encouragement. A special thanks to my mother and father in law, Homa and Hossein Fatemieh and my sister in law, Tannaz, for their continuous help and support. Last but not least, I would like to thank my husband Omid, for his eternal support and understanding of my goals and aspirations. His love and support have been and will remain a true source of encouragement for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Wireless sensor networks have evolved from simple systems of sensors into networks of smart devices with increased efficiency, functionality, and reliability. This allows Wireless Smart Sensor Networks (WSSNs) to penetrate new application areas. WSSNs are used today in complex applications such as structural health monitoring and earthquake monitoring. Deployments of such WSSN applications face varying environmental conditions that affect application requirements as well as the availability of resources such as energy. The increased dynamicity and complexity of WSSN applications require open systems that interact with their environment while addressing application constraints and hardware limitations.

Developing software for WSSNs is challenging for two main reasons:

1. Small changes in application behavior or hardware often require significant software modifications. For example, a structural health monitoring (SHM) application may need long-term monitoring of structural condition, as well as shorter-term investigative structural monitoring campaigns to diagnose specific problems. Each of these campaigns requires different services for sensing, data collection, etc. Hardware specifications can also affect software requirements. For example, switching from a battery-powered energy source to solar charging power can change the requirements of the underlying energy management service.

2. Not only does an application consist of many concurrent tasks, but several applications may be scheduled and executed concurrently.

This motivates the need for modular programming languages and tools for software development as individual modules may be used by different tasks or applications. Building applications by composing components can also increase flexibility. However, motivated by the need for efficiency, wireless and sensor network software developers have traditionally used one of two approaches: (1) development of software tailored to a specific application, and (2) use of architectures and models that allow software reuse but aim at general-purpose applications. The former prevents reuse and sharing of software components but satisfies application requirements while the latter approach is usually used to develop general-purpose sensor network applications or those without critical constraints.

Software development for long-term deployments of WSSN applications faces the additional challenge of dynamic environmental conditions and varying application needs [1]. Addressing environmental changes is especially important in domains where the availability of resources is affected by the surrounding conditions. For example, in applications where the energy supply is provided by solar panels, weather factors (e.g. sunlight and temperature) as well as local node characteristics such as the direction of the solar panel can significantly impact energy availability of individual nodes. As a result, application behavior should be tailored to the available resources to ensure long-term network operation. Application requirements can also vary at times. For example, a structural health monitoring application may have a periodic monitoring campaign for studying general structural conditions, and an intense short-term monitoring for diagnosing specific problems.

The goal of this research is to bridge the gap between developing reusable software for WSSNs and satisfying application requirements. We take this one step further and provide a framework that allows dynamic selection and configuration of services in response to environmental conditions that affect their energy characteristics. The energy consumption of services that compose an application can significantly affect the lifespan of the sensor network. Service energy consumption depends on deployment characteristics and environmental conditions and cannot be accurately determined until the full-scale deployment of the sensor network. Our framework embodies components that estimate energy consumption of services and detect and isolate energy anomalies based on aggregate information from the sensor network.

We call our framework *Illinois Adaptive Middleware for wireless smart sensor Networks*, or *I-AdMiN*. The design of I-AdMiN is motivated by our experience in the design and development of middleware services such as reliable multi-hop communication and energy management that address the specific needs of data-intensive applications. The design of these middleware services is a good illustration of services that are tied to hardware and environmental characteristics and can address application requirements by allowing adaptation and dynamic parameterization. The implemented services have successfully been deployed in two testbeds: Jindo Bridge in Korea, and a 40 acre agricultural field in Champaign, Illinois.

We consider applications that can be fully represented as a composition of services where a service is a self-contained software components with self-describing interfaces that represent not only the service's inputs and outputs but also *non-functional* properties. Our framework allows for automatic service selection from a pool of available services such that the selected services address application needs. This ensures *sharing* and *reuse*. We address the *dynamicity* in environmental conditions and application requirements by designing a light-weight monitoring system that captures energy characteristics of services. Service selection and configuration can be dynamically adapted in response to energy consumption of services or their detected anomalies.

1.1 Thesis Statement

This research focuses on the problem of software development for large-scale WSSN applications. We have an application-centric view in that the ultimate goal is the satisfaction of requirements of dynamic WSSN applications. Among the main axes of our design goals are *sharing* and *reuse*, *separation of concerns*, and more importantly, *adaptivity*. Our proposed solutions can be divided into three parts. First, we design an adaptive middleware framework that allows for agile and policy-driven adaptation . Second, we design and implement a light-weight monitoring system to capture dynamic service properties, as well as constraint violations. Finally, we design and implement a framework called *S4* for automatic selection of services that address specific application needs. We summarize the thesis statement as follows:

WSSN applications require adaptive and dynamic selection, configuration, and sharing of components. Such adaptation can be achieved by on-line monitoring of sensor network behavior and solving constraints to update component configuration.

1.2 Contributions

To summarize, this thesis has the following main contributions.

- We have investigated and implemented two real-world WSSN applications, namely *structural health monitoring* and *environmental and agri-*

cultural monitoring. Specifically, we have identified the requirements for successful long-term deployment of these applications. The study of these applications motivates our work to design a framework that facilitates software development for WSSNs while allowing dynamic configuration based on environmental characteristics (Chapter 3).

- We have designed a framework called I-AdMiN that uses on-line monitoring and service selection to enable dynamic service composition and adaptive reconfiguration. In this model, we follow principles of *service oriented architecture* and use the *Actor model of computation* to represent service instances and their interactions. *Actors* are concurrent active objects interacting via asynchronous message passing. Our proposed model breaks the static linkage and pre-determined customization of the service parameters in favor of *dynamic service composition*, where such actions take place at runtime (Chapter 4).

- We have designed and implemented a light-weight, on-line monitoring system to derive energy characteristics of services which may be composed to implement a WSSN application. We follow a data-driven approach for the monitor where coarse-grained data is used to derive service-specific information. The proposed monitoring system is comprised of two parts. The first part dynamically estimates energy consumption of individual services using weighted least squares regression. The second part identifies and isolates energy anomalies. We focus on three classes of energy anomalies: hardware issues, service issues and interaction issues. We have designed a Bayesian belief revision system where each hypothesis states the probability of having high energy consumption when a set of services are running. As new data becomes available, the probability of each hypotheses is revised using Bayesian methods. In both of these parts we use data from ISHMP Jindo Bridge deployment [2] (Chapter 5).

- We have developed a systematic method to support the use of component-based customizable and adaptive services in application development and have applied it to WSSNs. Specifically, we design an approach using specifications of application requirements and service descriptions. We consider the service selection as a constraint satisfaction problem (CSP). Each service provides a set of specifications, defining the requirements it satisfies. These specifications will be used in the CSP as variable domains, and the constraints are generated from application requirements. A set of services that match application requirements is chosen, and a configuration file with service parameters and dependencies is automatically generated (Chapter 6).

1.3 Organization

We illustrate the important considerations in our framework design through examples of real-world applications, namely structural health monitoring (SHM) and environmental and agricultural monitoring.

In Chapter 2 we present a review of related work. Specifically, we overview previous work in the areas of *WSN macroprogramming*, *service oriented architecture*, *general-purpose middleware solutions*, *software reconfiguration in sensor networks*, *mobile agent systems for WSNs*, *web service composition* and *constraint satisfaction problems.*

In Chapter 3, we briefly overview SHM and environmental and agricultural monitoring applications, and their requirements and challenges. Following the discussion of the different WSSN applications' characteristics, we elaborate their impact on system design. Specifically, we discuss the importance of *service selection and sharing* as well as *dynamic configuration* and the associated challenges. We use examples from our own experience in designing and implementing middleware services such as *reliable multihop communication* and *energy management*.

In Chapter 4, we introduce our high level architecture and its components. We leverage the Actor model of computation to represent services and their interactions. Our architecture is comprised of three main components, each of which is represented by an actor with specific responsibilities. The first component is a light-weight on-line *Monitor*. The goal of the Monitor is to provide the rest of the system with necessary information for an efficient configuration in response to environmental, application and hardware changes. We discuss the Monitor in Chapter 5. The Monitor has two main goals. First, it should find dynamic service properties using aggregate data from the deployment. Dynamic service properties include non-functional properties such as energy consumption and runtime and can greatly impact the network lifetime. Second, it detects constraint violations and finds their cause. Here again, the Monitor uses aggregate information to keep the logging overhead at minimum. In Chapter 6, we discuss our service selection component. We have designed S4 for automatic selection and parameterization of services that constitute a WSSN application. S4 allows application developers to configure component-based applications and their constituent middleware services in order to ensure that application constraints are satisfied. S4 solves application constraints to derive component configurations. The component configurations are then matched with particular parametrized services. We conclude the thesis in Chapter 7 and discuss future directions.

CHAPTER 2

RELATED WORK

Six areas are related to the research described in this thesis. In this chapter, we first describe related systems and will then discuss how they are different from our approach.

2.1 Wireless Sensor Network Macroprogramming

The goal of macroprogramming is to facilitate Wireless Sensor Network (WSN) programming by hiding low-level details of the distribute implementation. Specifically, macroprogramming enables programmers to specify the global behavior of a distributed system which may then be used to derive the behaviors of individual nodes automatically. Among the widely cited approaches to macroprogramming are database-like systems such as *Cougar* [3] and *TinyDB* [4]. The database-like approaches allow the user to use declarative SQL-like queries for accessing sensor data. The database-like approaches focus on efficient sensor data acquisition. These methods are usually static in terms of service selection and are optimized for processing users' queries from a collection of sensor nodes.

Kairos [5] and *Pleiades* [6] provide abstractions for expressing the global behavior of distributed computations. These systems allow the programmer to implement a central program that conceptually has access to the entire distributed system. *Macrolab* [7] provides a macroprogramming framework that offers a vector programming abstraction. Macrolab allows the user to write a single program for the entire distributed system, which is automatically decomposed to microprograms that run on individual nodes. MacroLab introduces a new data structure called a *macrovector*, each element of which corresponds to a different node in the network. The Macrolab provides deployment-specific code decomposition in that a central, distributed or hybrid decomposition is chosen based on the target deployment. *EnviroSuite* [8] proposes environmentally immersive programming, an object-based programming model in which individual objects represent physical elements in the external environment. The runtime system dynamically generates object instances when corresponding environmental elements are detected and destroys them when these elements leave the network.

ATaG data-driven macroprogramming language [9] and the Regiment macroprogramming system [10] are among compilation-based approaches. AtaG follows a hybrid approach in that communication among nodes is described in a declarative manner, whereas the local computation is expressed using an imperative language. Regiment compiles queries into stream-processing dataflow graphs, and thus is not suitable for highly dynamic applications. Moreover, Regiment is based on specific communication assumptions which hurts its flexibility and generality.

Finally, authors in [11] argue that node-level microprogramming can be made easier by using the right set of programming abstraction. They present *μSETL*, a programming abstraction for sensor networks based on *set theory*. The main purpose in the design of μSETL is to allow programmers to write event-driven programs from a node-level viewpoint while offering a high level of abstraction.

Similar to macroprogramming approaches, we aim to facilitate software development for WSSN applications. Towards this aim, we focus on three main design principles, namely *separation of concerns*, *sharing and reuse*, and *dynamicity*. Macroprogramming addresses separation of concerns by hiding low level details from application developers. However, macroprogramming approaches do not target component sharing. Our proposed framework better addresses modular functionality and component sharing and reuse by following a service-oriented approach in which services may be shared between applications. Services can be automatically selected based on application requirements and service specifications. Service selection is done in a module called *S4* which maps the service selection problem to a constraint satisfaction problem.

2.2 Service Oriented Architecture

Service oriented architecture aims to address challenges of WSSN application programming through the use of well-defined services that together compose an application [12, 13, 14]. The services provide a description, called an interface, of their inputs, outputs, and functionality, along with their non-functional aspects such as timeliness, resource requirements, etc. In this architecture, services are not tightly coupled with each other and do not need to know how the input data they require is provided. An important advantage of SOA for application development is that it enables separation of concerns [12]: application users are concerned only with application behavior and high-level requirements, while services and the middleware are implemented by service and systems programmers, respectively. The design of customizable services in this context promotes reuse as services for a given application domain can be adopted by a multitude of other applications.

We limit our discussion of SOA to those intended for sensor networks. A discussion of SOA for distributed systems in general can be found in [15].

SONGS [13] is a service-oriented programming model that is built on top of a hierarchical architecture consisting of sensors, field servers, and gateway servers. Field servers provide the library and manage the execution of semantic services, and gateway servers accept user tasks and derive optimal service composition plans. In SONGS, an application is composed of *semantic services* which are components that convert between elements in the information structure. A service description includes descriptions on data semantics required for the services to execute, called pre-condition, as well as the new semantics the service creates at its output ports, called post-condition.

Work in [14] proposes design principles and an SOA architecture for dynamic and concurrent execution of WSN applications. An application is comprised of a composition of middleware service requests, with *meta-actors* being responsible for handling the interaction among the services. Meta-actors are are control threads supervising deployment and execution of the services. Work in [16, 12] adopts SOA to improve portability for heterogeneous wireless sensor networks and cyber-physical systems. A two-level architecture is used separating the execution and controlling of the execution process. This model uses actor model of computation [17] for service interaction. In this architecture actors represent services, and meta-actors supervise deployment and execution of the services. The aforementioned approaches on SOA for WSNs lack a method for selecting services based on specific application needs. Our automatic service selection module can be used with any of these SOA architectures to facilitate service composition.

OASiS [18] follows an object centric, ambient-aware, and service-oriented approach. OASiS is different from our approach in that it is designed for data flow applications. A physical phenomenon of interest is represented by a finite state machine (FSM). Each FSM mode corresponds to a different physical state, and contains a service graph specifying the appropriate actions to take for the specific situation. Service discovery is passive in OASiS in that services are not advertised until there is a request for them. Requests are flooded a limited number of hops and service providers respond with their node ID, location, and power level. OASiS employs a globally asynchronous, locally synchronous (GALS) model for service communication.

2.3 General-purpose Middleware Solutions

Middleware has been devised to facilitate software development for distributed systems by masking problems of heterogeneity and distribution. There are many approaches to middleware with different requirements and performance characteristics. Middleware approaches for distributed software development can be divided into four categories based on the communication model they adopt: *transactional*, *message-oriented*, *procedural*, and *object or component* middleware [19].

Transactional middlewares, such as *TUXEDO* [20] and *CICS* [21] provide an interface to run transactions among different components of a distributed system. The two-phase commit protocol is used in transactional middleware to support distributed transactions. Transactional middleware can impose an undesirable overhead if there is no need to use transactions. Message-oriented middleware (MOM) allows communication between parties via message exchanges. MOM can be divided into two types: message passing/queuing, or message publish/subscribe [22]. Examples of message-oriented middleware are *MQSeries* from IBM [21], Sun's *Java Message Queue* [23], Tibco *Ren-*

dezvous (TIB/RV) [24], and *SonicMQ* from Progress [25]. In MOM, the marshalling code has to be written by the programmer, which complicates the use of message-oriented middleware. Procedural middleware is based on the Remote Procedural Call (RPC) protocol. It defines services as RPC programs, establishes synchronous communication, and provides marshalling and unmarshalling of parameters that are sent via messages. The primary example of procedural middleware is DCE (Distributed Computing Environment). Object and component middleware can be considered as an evolution of procedural middleware. This class of middleware supports distributed object requests and provides marshalling and unmarshalling of exchanged data. Examples of object and component middleware include Common Object Request Broker Architecture (CORBA) [26], Microsoft's Component Object (COM) [27], Enterprise Java Beans [28], Jini, and Java RMI. Service Oriented Architecture (SOA) is another step forward from component-based programming towards the development of dynamic, heterogeneous, and distributed applications using self-describing software components called *services* [29, 30, 31].

Traditional middlewares lack performance and memory optimizations [32] and are thus unsuitable for WSSNs. The incurred overhead in these middleware is due to heavy data copying, sharing of context data, object and service discovery, and internal message buffering strategies. Moreover, most traditional middleware for distributed systems aim at providing transparency abstractions by hiding the context information while WSN-based applications are usually context-aware [33].

2.4 Software Reconfiguration in Sensor Networks

Solutions to enable software reconfiguration in sensor networks range from full software image updates to programming models and middleware solutions that aim to enable lightweight and dynamic reconfiguration. Early approaches targeting dynamism in sensor networks such as *Deluge* [34] involved re-installing the full software image which imposes an unacceptable energy overhead especially in systems where frequent software updates are common. Modular software update during runtime, as provided by *Contiki* [35] can reduce the reconfiguration overhead.

Component-based reconfiguration approaches aim to provide dynamic composition of software modules capable of interaction with each other and the environment. *FlexCup* [36] provides a code update algorithm that allows exchanging only the components of a program that have changed. FlexCup has high memory overhead as each sensor node maintains metadata that is used to update the software. For updates, the compiled image of changed components along with the new symbol and relocation tables are transmitted. *OpenCOM* [37] is a lightweight reflective component model based on COM and can be used to construct a reconfigurable middleware platform. OpenCom has been extended by *GridKit* middleware [38] to implement a flood monitoring sensor network.

FiGaRo [39] provides a component and distribution model which enables control over what is configured and where. FiGaRo provides run-time support for dynamic reconfiguration using library functions that are linked against the Contiki kernel. *LooCI* [40] is another component based approach and provides a loosely-coupled, event-based binding model for Java devices. *Re-Wise* [41] is a component model for lightweight reconfiguration and adaptation in sensor applications. ReWise focuses on the *reconfiguration degree* of software modules and aims at providing fine-grained reconfiguration where the reconfiguration is narrowed to the part of a component that needs to be updated rather than the entire component. This is achieved by implementing an interface in a separate component, called TinyComponent, containing just the implementation of that interface rather than implementing an interface as a method within the component body.

Middleware architectures can facilitate software development for distributed systems by masking problems of heterogeneity and distribution. *Impala* [42] is one of the early middleware platforms targeting dynamic reconfiguration. It is a middleware system and API which aims at providing sensor network application adaptivity. Software updates are handled in a distributed fashion in Impala which imposes high memory and energy overhead. The event-based programming model of Impala is designed for a particular application, ZebraNet, and does not offer a mechanism for changing the triggering events. The *RUNES* [43] middleware is based on a two-level architecture: a lightweight component model above which there is a middleware layer providing dynamic reconfiguration. *DAVIM* [44] is an adaptable middleware for dynamic service management and targets simultaneously running applications in sensor networks. DAVIM uses virtual machines to isolate and run applications. It groups high-level operations in a virtual machine's instruction set as libraries and allows users to add, update, or remove them at runtime.

WiSeKit [45] provides a component-based middleware approach for dynamic adaptation and reconfiguration of sensor networks. WiSeKit offers local adaptivity at the node level by allowing parameter adaptation according to context information and adaptation policies. Component reconfiguration is provided at the cluster and the entire network level and requires predefined component interface implementations. WiSeKit follows a situation-action rules adaptation policy while our approach is goal oriented. In our view, a goal-oriented approach is better suited for resource-constrained domains and where policy changes are expected. *RemoWare* [46] is the run-time system supporting the in-situ reconfiguration of REMORA component model [47]. A reconfiguration middleware consisting of a set of dedicated services provides support for reconfiguration of REMORA components. These dedicated services act as a set of static components and rely on low-level system functionalities. For the dynamic parts of the system, the middleware exhibits an API that allows the programmer to use the reconfiguration features. One challenge with dynamic middleware solutions such as RemoWare is the binding model, and more specifically, the linking of dynamic components. RemoWare for example, uses a Dynamic Linking Table (DLT) for static components and a Dynamic Invocation Table (DIT) for dynamic components to resolve dynamic links. This approach however has a high memory overhead, especially with a large number of services. There is also an additional overhead for forwarding function calls to the correct service memory address and also for keeping the table up-to-date.

2.5 Mobile Agent Systems for Wireless Sensor Networks

Using proactive mobile agents can provide flexibility in reprogramming and operating WSNs and substantially reduce energy consumption by reducing the amount of communication required. *Mate* [48] is one of the first mobile agent platform designed for WSNs. It is specifically designed for highly memory restricted sensor nodes. Mate has high level instructions which result in a small code size and an efficient code migration. *Agilla* is another mobile agent platform for WSNs [49]. Like Mate, Agilla is a stack-based virtual machine with special instructions for code mobility. Additionally, Agilla supports multiple applications running on a single node and features a Linda-like tuplespace that decouples data from the spatial constraints [50]. However, in these systems, the programmability and the code maintainability pose a challenge due to the low level of language abstraction.

The mobile agent platform *SensorWare* [51] provides high level language abstractions for WSNs. Specifically, SensorWare supports an event-based Tcl-like script language. This not only improves increases the programmability but also reduces the code size. Currently, SensorWare is implemented for larger platforms like PDAs.

The *Melete* system is based on the Mate virtual machine and provides a method for on-the-fly deployment and concurrent execution of applications in a sensor network [52]. Each application is executed on a subset of nodes that form a group for the application. Compared to Melete, our method provides more flexibility in code updates as well as context awareness and adaptivity.

ActorNet [53] is a mobile agent platform for WSNs, designed to support multiagent applications on resource-limited systems. It provides services such as virtual memory, context switching and multi-tasking. ActorNet agents, are light-weight actors, and are based on the actor model of computation. Actors are concurrent active objects that interact via asynchronous message passing [17]. The actor model can be used as a formal programming model for agents [54, 55, 56, 57].

2.6 Web Service Composition

Automatic composition of Web services has been proposed to facilitated Web application development. Web services are self-contained and self-describing modular units [58] that are created and updated on the fly. Languages have been proposed to define a standard for service specification, discovery, and invocation. Such languages include Universal Description, Discovery, and Integration (UDDI) [59], Web Services Description Language (WSDL) [60], Simple Object Access Protocol (SOAP) [61]. Web service composition allows building applications from existing services. Dynamic composition methods have been proposed to facilitate the development of Web service applications. Workflow-based composition methods can be static or dynamic [58]. Static workflow-based composition only automate the selection and binding of atomic Web services [62], while dynamic workflow-based compositions create process model and select atomic services automatically. Web service composition has also been solved via AI planning approaches [63, 64, 65, 66].

Work in [67] considers end-to-end quality of service (QoS), and presents SLAng. SLAng defines Service Level Agreements (SLAs) that accommodate agreements between network services, storage services and middleware services. SLAng's reference model assumes the use of component oriented middleware and concentrates on application server technologies such as J2EE, and CORBA. *QUEST* [68] aims at managing generic quality-of-service (QoS) provisioning. QUEST provides initial service composition based on QoS constraints such as response time and availability as well as dynamic recomposition of services to recover from service outages during runtime. The service composition model in QUEST first maps each user request to a composite service template based on application specific requirements. The template is then mapped to an instantiated service path based on distributed performance (e.g., response time) and resource availability conditions. Work in [69] further improves dynamic QoS-aware service composition by supporting parallelism and branching. It presents *AgFlow*, a middleware platform that enables the quality-driven composition of Web services and provides an adaptive execution engine.

Constraint satisfaction problems have been widely used to reduce the complexity and time needed to generate a composition from the best possible available services. Work in [70] has reduced the dynamic composition of the web services to a constraint satisfaction problem which can be solved by any linear programming solver. Work in [71] encodes the problem of issuing requests to a composition of Web services as a constraint-based problem. [72] presents a constraint-based distributed framework for the provisioning of services. [73] considers two types of constraints: soft constraints and hard constraints. The constraints in [73] are mainly configuration parameters rather than full constraint consistency. Full constraint consistency is important for complex WSSN applications and ensures that all application requirements are satisfied. Work in [74] extends constraint based systems for service composition by addressing the need for a semantic candidate selection process. Composite Web service framework (CWSF) [75] uses semantic information of the Web services for composition. Thus, in CWSF, the selection of services that may satisfy the constraints does not rely on syntactic features, but uses the semantic model of the services.

Service selection and configuration for WSSN application development is different from Web service composition in a number of ways. First, in WSSNs there are many more axes and resources, along which the system should be optimized. This is due to hardware limitations of WSSNs as well as environmental effects. An additional abstraction layer is required to address low-level hardware characteristics. Second, unlike Web service composition, WSSN application development requires service parameter selection. Service parameterization can change the behavior of services and should address application constraints. Finally, the dynamically changing application requirements and the environmental characteristics requires adaptive modification of low-level behavior where service configuration and parameterization is no longer static.

2.7 Constraint Satisfaction Problems

Constraint satisfaction problems (CSP) involve finding a set of values for variables in the system such that specific constraints are satisfied. A large number of computer science problems can be considered as a special case of CSP. Examples include map coloring [76], scheduling [77], resource allocation [78], and satisfiability problem. Backtracking is a very common method for solving CSP [76]. Various methods have been proposed to improve the performance of backtracking, including constraint propagation, reason maintenance, intelligent backtracking, and variable ordering and instantiation [76]. CSP can be *unary*, involving a single variable, *binary* involving pairs of variables, or *n-ary* which involves n variables. The problem addressed in this work is an *n-ary CSP*, as it involves a large number of variables depending on application constraints. Distributed constraint satisfaction problem was introduced in [79]. As the name suggests, in distributed CSP variables and constraints are distributed among multiple agents. Other variations of CSP include *hierarchical Domain CSP* where the domain of variables are organized in a hierarchy [80], *dynamic CSP* [81] where the set of variables cannot be determined a priori, and *meta CSP* where the problem is decomposed into subproblems. Meta constraint satisfaction problems were originally designed to deal with the complexity of solving a problem by solving an equivalent problem, represented at a different level of abstraction, which can be solved more efficiently. Our CSP for selecting a set of services that address application needs is represented as a meta CSP, where each metavariable represents selection of a particular type of service.

2.8 Summary

In this section we summarize previous work on WSN programming approaches. Table 2.1 compares our approach with that of others. The comparison is based on WSSN application deployment challenges that we encountered during our real-world applications (Chapter 3), as well as our design principles. The main deployment challenges that we consider in this analysis are satisfying application requirements, efficient service selection, dynamic reconfiguration, and extracting dynamic service properties. Dynamic properties of services refer to service properties such as energy consumption and runtime that cannot be accurately determined prior to the deployment and may change over time. We follow three main principles: separation of concerns, sharing and reuse (modularity), and adaptivity. We also strive to provide resource efficiency and flexibility to address resource limitations of WSSNs.

Approaches such as Deluge [34] that provide full-image software updates provide flexibility but are not resource efficient. SOS [82] and Contiki [35] allow modular binary updates which improves modularity and resource efficiency but offers less flexibility. Component-based reconfiguration methods such as FlexCup [36], OpenCOM [37], FiGaRo [39], LooCI [40] and

Table 2.1: A comparison of WSSN programming approaches.

Approach	Resource Efficiency	Flexibility	Separation of Concerns	Modularity	Dynamic Reconfig.	Service Selection	Dynamic Properties
Full software image updates		✓					
Modular upgrades	✓	✓		✓			
Virtual machines	✓						
Mobile agents	✓				✓		
Component-based reconfig.	✓	✓		✓	✓		
Reconfig. middleware	✓	✓	✓	✓	✓		
I-AdMiN	✓	✓	✓	✓	✓	✓	✓

REMORA [47] provide flexible dynamic reconfiguration and modularity while addressing resource limitations of sensor networks. Reconfiguration middleware approaches such as Impala [42], RUNES [43], and WiSeKit [45] improve on component-based reconfiguration by providing separation of concerns. Dynamic reconfiguration of applications that are built using component composition requires an efficient method for service selection. Moreover, the system needs to automatically find dynamic service properties such as energy consumption and runtime that can not be accurately determined prior to system deployment. I-AdMiN provides automatic and efficient service selection by mapping the service selection problem to a constraint satisfaction problem which is solved in a module called *S4*. Another module, called *monitor*, derives energy characteristics of services by dynamic and on-line profiling of services during the WSSN deployment. The monitor follows a data-driven approach where coarse-grained data is used to derive service-specific information.

I-AdMiN provides modularity, separation of concerns, and dynamic reconfiguration by using a two-layer actor model (Chapter 4). The *base layer* is comprised of services that constitute an application, each of which is represented by an actor. The *meta layer* contains meta-actors that are responsible for automatic service selection (Chapter 6) as well as on-line monitoring (Chapter 5), and provide dynamic adaptation in response to application and environmental changes.

Variations of the two-layer actor model have previously been used to solve problems of distributed systems. [83] introduces a meta-architecture where objects of a real-time program are represented by a variant of the actor model. A high-level programming language called *RTsynchronizer* then defines the temporal constraints between actors. Work in [84] extends [85] and [83] by providing a formal treatment of the model. Distributed Connection Language (DCL) [86] uses the actor model to provide a two-layer architecture description language. At the meta-level, architectural policies are applied to a collection of actors. Each collection of actors (called module actors) implements a particular computational behavior. A meta-architectural framework based on the actor model, called the Two-Level Actor Model (TLAM), is presented in [87]. TLAM enables cost-effective QoS in distributed multimedia systems by providing a model for specifying and reasoning about the components.

CHAPTER 3

APPLICATION-CENTRIC APPROACH TO SYSTEM REQUIREMENTS

WSSN applications have unique requirements that are crucial to their successful deployment. These requirements vary widely from one application to another and can change over time. Unfortunately, finding application requirements is by nature an ad-hoc process and cannot be done automatically. Moreover, these requirements may change over time or in response to environmental changes. Application developers should therefore work closely with application users to have a clear understanding of the application and its specification.

3.1 Motivating Examples

Our research is motivated by two types of WSSN applications: applications with *bursty and high-throughput* data transfer, which we cal *data-intensive* applications, and applications with *low data rate* communication. High-throughput applications impose specific requirements such as high sampling rates, timely data collection and analysis, large volume of data, and reliable communication. This results in *data storms*-intense bursts of high-volume data transfer from multiple nodes in the network. Example applications include monitoring the structural health of civil infrastructure and rare event detection (earthquake, mudslide, etc.) [88, 89]. Low-throughput applications require relatively low-frequency data gathering, and do not come close to saturating the available network bandwidth. Example applications include medical monitoring, environmental observation and forecasting systems, and habitat monitoring [90, 91, 92].

We overview two applications that we have been investigating. The first application, *structural health monitoring*, is a high-throughput application with bursty traffic. The second application is *environmental and agricultural monitoring* and falls into the low-throughput category. Below, we will briefly introduce these applications and their deployments.

3.1.1 Structural Health Monitoring

Objective evaluation of structural performance can facilitate effective and efficient maintenance of aging infrastructure. Manual inspection of civil infrastructures can be costly and unreliable. Typical wired monitoring systems are also costly and are hard to deploy. Wireless smart sensor networks (WSSNs) offer the potential for dramatic improvements in the capability to capture structural dynamic behavior and evaluate the condition of structures. Sensing devices are becoming smaller, less expensive, more robust, and highly precise, allowing collection of high-fidelity data with dense instrumentation employing multi-metric sensors. SHM using WSSNs has been recognized as an important emerging applications of sensor networks [93, 94, 2, 95]. WSSNs can be employed for medium- and long-term monitoring of structural condition, as well as for shorter-term investigative structural monitoring campaigns to diagnose specific problems.

Vibration (accelerometer) and strain (strain gage) are common sensor modalities for SHM. Sensor measurements are collected periodically or in response to a high level of excitation, over the span of several months or even years. In the design of large-scale SHM systems, application-specific characteristics and requirements must be considered:

Large data size In SHM applications, each sensor node usually has multiple sensing channels, each of which samples surface vibration with frequencies as high as 1000 Hz. The measurement data must be divided into a large number of packets for transmission.

Dense deployment of sensor nodes in large spatial regions Civil infrastructure is typically extremely large and the communication protocol needs to support large and dense multi-hop networks.

Radio communication environment The radio communication environment on the bridge can be complex due to RF reflection, refraction, absorption, and other phenomena. Some bridge components may be between two WSS nodes, interrupting direct line-of-sight communication. Therefore, the communication range on a bridge will vary from place to place, and its estimation prior to on-site tests is challenging.

Nodes at fixed locations For most structural health monitoring applications, sensor installation locations are predetermined based on design drawings and structural considerations. These installation locations do not necessarily correspond with locations that are desirable with respect to RF communication.

Need for prompt data collection/analysis Structural vibration monitoring applications generally require prompt data collection and analysis, though data collection does not necessarily need to be in real-time. In particular, performance evaluation after extreme events such as earthquakes and

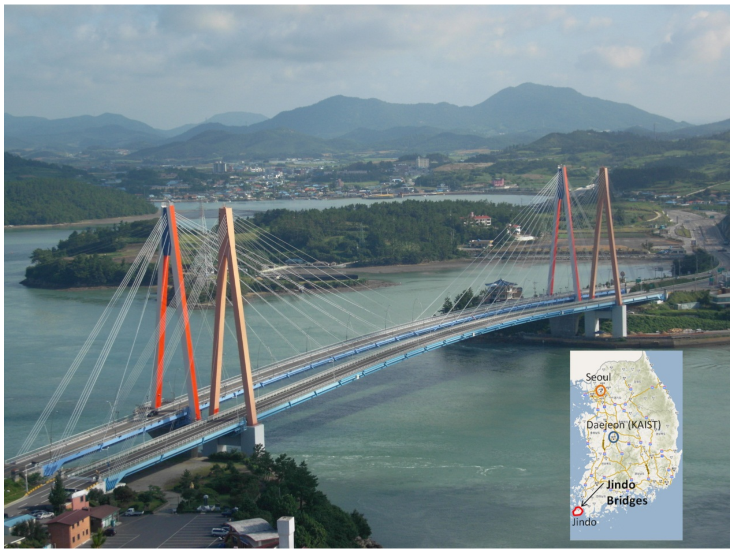


Figure 3.1: Twin Jindo Bridges connecting Jindo Island with the Korean Peninsula.

typhoons must be done as soon as possible to address safety concerns. As for monitoring campaigns where operation time is limited, data collection must be done in a timely manner.

High requirement on reliability Reliability of transporting acquired sensor data is vital. Most scenarios assume measurement data is available from all the nodes without intermittent loss. For example, many damage detection algorithms require sensing data from predetermined locations, while only a few algorithms so far have been extended to provide robustness against node failure.

The software developed in our research has been used in a SHM deployment in Jindo Bridge, Korea (Figure 3.1). This deployment is part of the Illinois Structural Health Monitoring Project (ISHMP) [2] and serves as an international testbed for wireless smart sensor network technology. The primary goal of the Jindo Bridge SHM project has been to realize the first large-scale, autonomous WSSN for structural health monitoring, taking advantage of the advanced computational capabilities of the Imote2 nodes. The monitoring application is built from statically-linked middleware services for a variety of application-independent as well as domain-specific tasks. All sensor nodes are equipped with a solar energy harvesting system and report their available voltage, charging current and temperature periodically. In total, 113 Imote2 leaf nodes have been installed on the Jindo Bridge measuring acceleration. At the time of this writing, the sensor network remains in operation on the bridge.

3.1.2 Environmental and Agricultural Monitoring

Wireless and sensor networks can be used in agricultural and environmental applications to study nitrate uptake and run-offs. Fertilizer is applied in the Fall and a significant fraction of it ends up in water streams due to run-offs before it can be taken up by plants in the Spring. The run-offs result in the release of Nitrous Oxides, Greenhouse Gases that are approximately a hundred times more potent than Carbon Dioxide. By developing methods to understand the nitrous uptake in a field at a fine spatial and temporal granularity, fields could be instrumented to minimize such run-offs.

Increasingly, researchers are turning to empirical observation of the holistic environmental system using networks of remote and embedded sensors. Given the scale of the observation area and the large number of potential events to be monitored, sensing resources will necessarily be limited and must be managed intelligently to achieve an acceptable level of network sensing performance. In environmental and agricultural monitoring applications, measurements of soil moisture, temperature, and humidity are taken by the wireless network sensors as often as every 15 minutes over the span of sev-

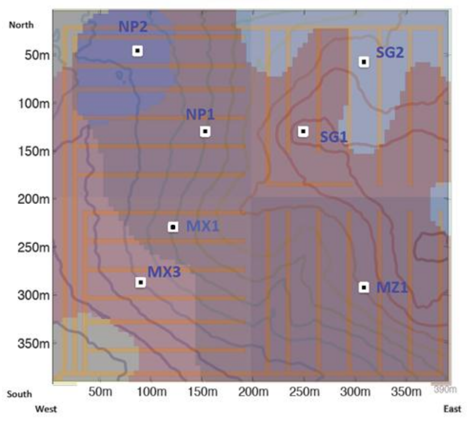


Figure 3.2: Sensor locations for the environmental observation application.

eral months or even years. In each interval, if the data is interesting in some respect, e.g. it contains a deviation from the model-predicted behavior, it is stored and periodically aggregated at an Internet-connected base station, where it can be further processed and categorized. Unlike SHM, environmental and agricultural applications do not require large numbers of samples or high sampling rates, and the requirement on reliability is not strict. The data collected from a WSSN that monitors environmental characteristics is typically used for estimating crop yield which does not require prompt data collection and analysis. However, environmental monitoring applications require very frequent data collection which is a challenge due to energy limitations. Moreover, the large areas that need to be monitored, and the field vegetation complicate wireless communication. Similar to SHM applications, node locations are determined based on soil characteristics, and not radio communication quality.

We have used our software for a deployment of an environmental monitoring application that is part of the IACAT Virtual Environmental Observatory project. Currently, two types of sensors (soil moisture and nitrate) are deployed in a 40 acre field that has 4 types of vegetation (Figure 3.2). The

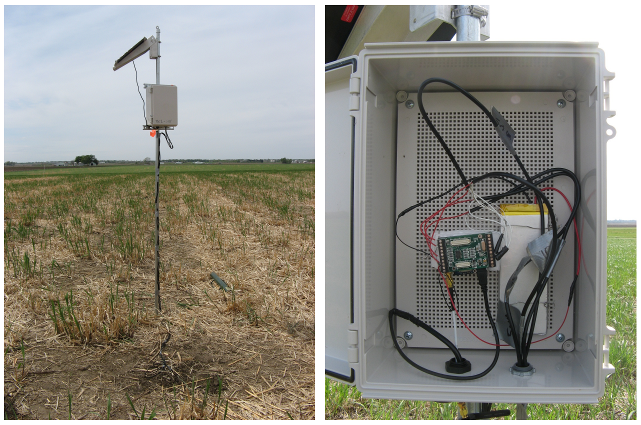

(a) A sensor node powered by a solar panel. (b) Each node supports four soil moisture sensors.

Figure 3.3: Sensor network deployment for environmental and agricultural monitoring.

sensor network includes 7 nodes, each equipped with 4 soil moisture sensors(Figure 3.3). The sensors measure soil moisture at four depths: 5cm, 10cm, 20cm, and 50cm. The measurements are taken every 15 minutes and are transfered to an Internet-connected base station, where it can be further processed and categorized.

3.2 Goals and Challenges

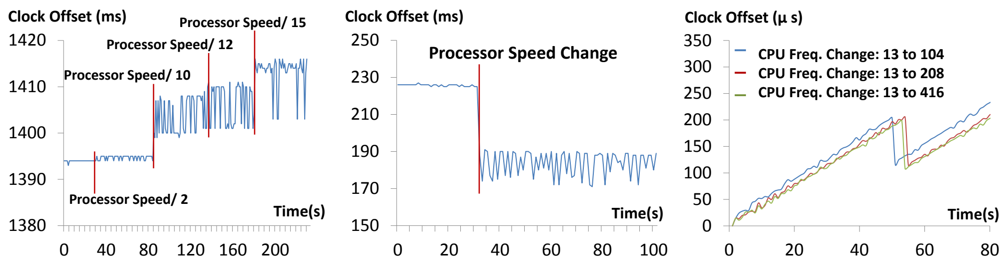
Successful realization of real-world WSSN applications relies on the development of scalable, robust, and efficient software capable of operating in this environment. Moreover, WSSN applications face varying environmental conditions and their requirements may change over time. During our experience with SHM and environmental monitoring applications we have learned that even a deep understanding of application and its requirements cannot ensure efficient and long-term network operation. This is because many requirements, interactions and specifications cannot be determined until the full-scale deployment of the application. Issues of this kind are the focus of this section and reflect the challenges we encountered during the course of software development for SHM and environmental monitoring applications. In the next chapters we discuss the design and implementation of components that address these challenges.

The applications we have worked on are built as a composition of services. Services are self-contained software components with self-describing interfaces that represent their inputs and outputs as well as their non-functional properties. Our discussions are therefore centered around challenges of developing reliable WSSN applications by linking together a set of services.

3.2.1 Service Interaction

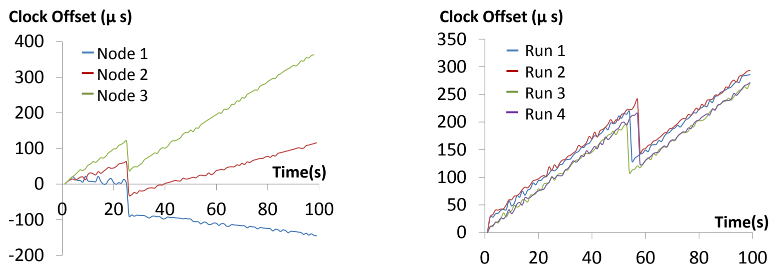
Service interaction can affect the overall behavior of the system. The WSSN is treated as a collaborative distributed computing platform, with policy goals acting on the entirety of the system, including multiple nodes and middleware services. Dependency relationships between different services and between the hardware platform and services must be taken into account in the course of service selection.

An example of service and hardware interdependencies is the relationship between time synchronization and an energy management service that uses processor frequency switching. In the course of designing an energy management service for SHM applications we discovered that the CPU frequency switching actions of a dynamic voltage and frequency (DVFS) service has an adverse effect on processor-based clocks. In other words, for embedded architectures with variable frequency CPUs that use the processor tick counter as a clock source, the act of changing the frequency can momentarily disrupt the local clock, causing a time lapse. Thus, changing the processor frequency also



(a) MICAz (b) Telosb (c) Imote2

Figure 3.4: Effect of changing CPU Frequency on different sensor platforms. Frequency change causes a sudden change in clock offset.



(a) Effect of CPU frequency change for three nodes

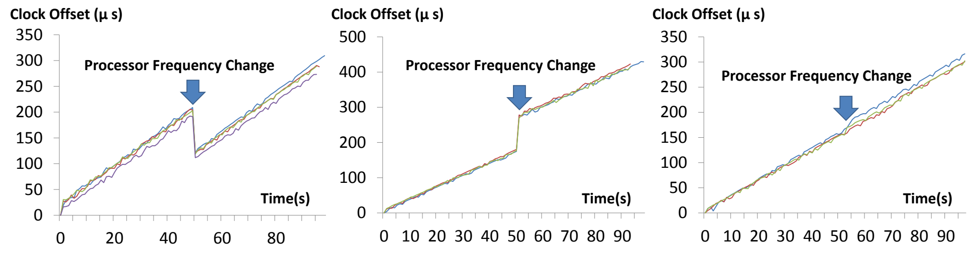
(b) Effect of CPU frequency change on the same pair.

Figure 3.5: Variance in clock offset change due to CPU frequency change.

changes the clock offset, measured as the difference between local clocks of different nodes. We call this effect the *time-keeping anomaly* [96]. The time-keeping anomaly is observed in three widely used sensor platforms: namely MICAz, Telosb, and Imote2 (Figure 3.4).

The new clock offset is not accurately predictable. Different offset changes are measured for different instances of the experiment for the same node pair. Using different nodes for multiple instances of identical frequency changes likewise leads to variable magnitude changes. Figure 3.5a shows the effect of frequency change on the relative offset for three different Imote2 nodes and Figure 3.5b shows this variance for different runs on the same node pair.

Note that, the amount of mean offset jump and mean offset error introduced due to DVFS also depends on the source and target states. It is possible to quantify the offset change and its variance for each frequency



(a) 13 to 104 MHz (b) 208 to 13 MHz (c) 208 to 416 MHz

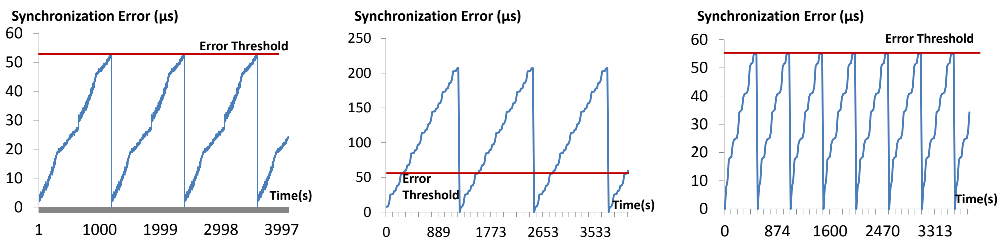
Figure 3.6: Effect of CPU frequency change on clock offset. The measurement is repeated multiple times on the same pair of nodes.

Table 3.1: Offset change due to frequency change on Imote2.

Frequency Change	**Median Offset Change (μs)**	**Offset Change Range (μs)**	**Frequency Change**	**Median Offset Change (μs)**	**Offset Change Range (μs)**
13 - 104	83	13	104 - 13	-95	11
13 - 208	92	8	208 - 13	-98	9
13 - 416	93	5	416 - 13	-102	14
104 - 208	3	2	208 - 104	-8	4
104 - 416	4	3	416 - 104	-8	8
208 - 416	-	-	416 - 208	-	-

pair through multiple runs and CPU frequency changes. Figure 3.6 and Table 3.1 summarize the results of offset change due to frequency switching for Imote2. The results show that the most significant change in the relative clock offset happens when switching CPU frequency to or from 13 MHz. The greater difference in the PLL divisor (8x, compared to 2x and 4x for the other frequency pairs) explains this behavior.

The disruption has a significant impact on the accuracy of local clocks, thereby greatly increasing the need for frequent resynchronization to maintain clocks within a fixed error bound. Traditional DVFS techniques have not considered the adverse effect of DVFS on clock synchronization and therefore are not suitable for sensor network applications that need tightly synchronized data. Fig. 3.7b shows the growth of synchronization error if DVFS



(a) Constant CPU frequency, $T_{Resync} = 20$ min (b) Classical DVFS, $T_{Resync} = 20$ min (c) Sync-aware DVFS determines T_{Resync}

Figure 3.7: Comparison of time synchronization error growth under different frequency switching policies. Synchronization error threshold is violated under classical DVFS.

is implemented without concern for the effect of CPU frequency changes on time synchronization. Note that the maximum possible error jumps with each frequency switch, and exceeds the maximum allowable value well before the 20 minute resynchronization period. In this case, while energy consumption may be lower than a synchronization-aware DVFS algorithm, synchronization requirements for SHM are violated. A synchronization-aware DVFS algorithm can address this deficiency by embedding the synchronization cost of frequency switching in the algorithm. Figure 3.7c confirms that our method presented in [96] meets SHM requirements: the overall energy consumption is significantly lower than that at constant CPU frequency, while the synchronization error is always within the acceptable bound.

3.2.2 Environmental Conditions

Environmental conditions affect WSSN application development in a number of ways. First, service configuration should reflect environmental conditions and deployment characteristics. This means that laboratory and small-scale experiments cannot ensure efficient service configuration once the sensor network is deployed. This is challenging due to the large gap between high-level application requirements and low-level details of service implementation. Manual and heuristic service configuration can be cumbersome and error-prone.

An example of complex service configuration, one that we have experienced is the deployment of a reliable multi-hop communication service for the SHM deployment on the Jindo Bridge. For this service deployment we selected parameters for the reliable multihop communication services (e.g. delay timers, number of route request packets, randomized wait times, etc.) based on laboratory and small-scale field experiments. Our experience with selecting these parameters showed that first, experimental parameter values can be far from optimal values; and second, lab-scale tests cannot provide proper parameters for full-scale deployments.

Another effect of environmental conditions becomes apparent when the availability of resources such as energy is affected by the surrounding conditions. Consider the environmental and agricultural monitoring application. The system is deployed for long-term monitoring and thus each node is equipped with a solar panel for the energy source. An efficient energy management system for this network should be able to adopt to environmental changes in two levels. At the local level, it should adjust node activities (sensing, data transport and processing, etc.), and middleware parameters (e.g., routing around a node) based on local parameters such as the available energy reserves. At the global level, the system should be able to respond to conditions (environmental, hardware-related, etc.) that have a system-wide effect. An example would be consecutive cloudy days which can change frequency of data collection.

For this application, any statically configured parameters for energy management in the various components of the application (routing, sensing, data processing, etc.) would necessarily have to be *overly conservative*. Otherwise, over-exploitation of the energy supply of the entire network or some select nodes would lead to failure as the energy supply is prematurely exhausted.

CHAPTER 4

ARCHITECTURE OVERVIEW

In this chapter we present a framework that embodies components we developed to facilitate software development for WSSNs while addressing application requirements and the dynamism in environmental conditions. We call our framework *I-AdMiN*, Illinois Adaptive Middleware for wireless smart sensor Networks. [1]

Service sharing and reuse can significantly reduce the overhead of software development in WSSNs. In order to ensure that the services that compose an application satisfy its varying requirements and address the dynamism in environmental conditions the following challenges should be addressed:

1. How to derive dynamic network information with low overhead.
2. How to efficiently select services in a way that application requirements are satisfied.
3. How to represent services and their interaction to allow efficient and dynamic service selection and parameterization.
4. How to apply dynamic network information to service selection.

We first introduce the components we have developed to enable 1 and 2. We then discuss how services are represented in I-AdMiN to address 3 and 4.

[1]Parts of this Chapter are done in collaboration with Kirill Mechitov and will appear in a subsequent paper.

4.1 Deriving Dynamic Network Information

Services that form an application can have varying properties which are used in deriving a service selection. It is important to derive service properties such as energy consumption during network runtime for two main reasons. First, many service properties such as service run time and energy consumption highly depend on the deployment characteristics such as network size and radio communication quality and cannot be accurately determined until the full-scale deployment of the service. Second, dependency relationships between different services and between the hardware platform and services can affect the overall behavior of the system and must be taken into account in the course of service selection. Many of such dependencies are unknown and depend on the environment and run time characteristics.

We have designed and implemented a light-weight monitoring system that uses aggregate information to derive dynamic service properties. For this component we focus on energy consumption and leave investigation of other properties as future work. The monitoring system is discussed in detail in Chapter 5 and has two main goals. First, it attributes aggregate energy consumption from logged data to individual services. Second, it detects energy anomalies and finds and isolates the cause. This information is used in service selection to ensure efficient network operation.

4.2 Service Selection

Service sharing and reuse requires a systematic method for service selection. Otherwise, application developers should investigate all available services without being able to efficiently verify if a service selection meets application requirements in the long term.

However, because of the diversity in available software components, selecting and sharing the right components to meet application requirements can be challenging. Service selection is challenging for two main reasons. First, current programming approaches in WSSNs lack a universal language for describing non-functional service properties. This is because programming in WSSNs has traditionally been done statically and manually. The need for a universal language for describing non-functional service properties is only recognized when a part of the code is to be automatically generated. The functional descriptions of services and their inputs and outputs are not sufficient to determine whether a service can satisfy certain requirements. For example, in order to determine the energy footprint of a service, detailed timeliness and resource consumption information are required. A universal description language is required to facilitate the interpretation of these specifications. The second challenge is due to the large number of available services and their varying requirements and parameters. Doing the selection manually makes the process not only complex and error prone, but defeats the goal of facilitating incremental change.

We provide a systematic method to support the use of component-based customizable and adaptive services in wireless sensor network application development. We have developed *S4* to enable sensor network application developers exploit a pool of existing services, while satisfying application requirements. Application requirements and service specifications form a constraint satisfaction problem to select a set of services. S4 takes advantage of dynamic data provided by the monitoring system. The monitor continuously profiles the deployed services and provides S4 with dynamic service specifications and dependency relationships. The details of S4 are discussed in Chapter 6.

4.3 Service Representation

We follow the principles of service-oriented architecture (SOA) [12] and consider applications that can be fully represented as a composition of services. Services are self-contained software components with self-describing interfaces that represent their inputs and outputs as well as their non-functional properties. A set of services can therefore be linked together to build an application with services interacting with one another through their well-defined interfaces. Services can be customized to the requirements of different applications through the assignment of specific values to their parameters, or service parameterization. A service invocation is called a service request and is what allows sharing of services between different applications.

We use the Actor model of computation [17] to represent service instances and their interactions. *Actors* are concurrent active objects interacting via asynchronous message passing (Figure 4.1). Compared to the component model, actors are a better fit for highly dynamic applications operating in open and changing environments. Actors may be created and destroyed dynamically, they can change their behaviors, and migrate to different physical locations. Following this model, we represent service instances as actors. A service interface is the set of messages the actor representing the services sends and receives. Service instances connect services to each other and to the application. Note that only the interactions between services need to follow the actor model; we do not restrict the internal implementation or semantics of the services themselves, or their tight coupling to the underlying operating system.

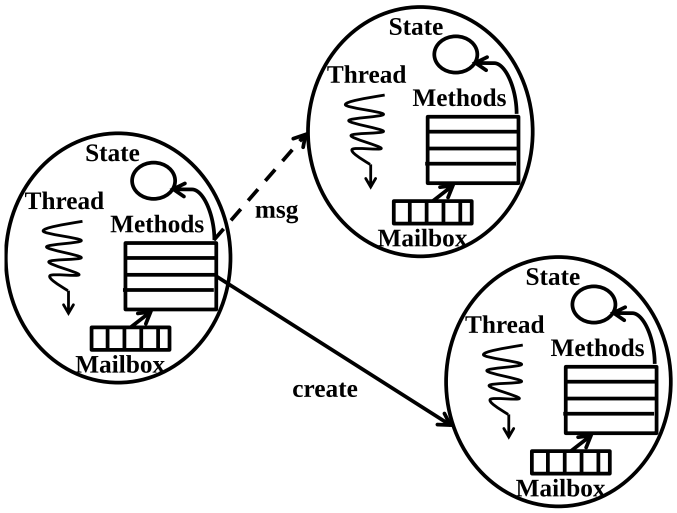


Figure 4.1: Actor model of computation. Actors are concurrent objects that communicate through message-passing and may in turn create new actors. An actor has its own thread of control, a mailbox, and a globally unique immutable name.

4.3.1 Service Description

Currently, a universal language for non-functional descriptions of service properties has not been adopted by the WSSN programming community. Without these specifications automatic service selection cannot be achieved and the application developer should go over service implementations in order to verify that they do not violate application constraints. In many cases, the programmers choose to develop the required services from the beginning instead.

Our proposed service interface specification (Figure 4.2) allows for the definition of *service properties* which describe non-functional service specifications. Non-functional service specifications can be specified either statically, or dynamically. Static descriptions correspond to those that are determined by the service developer prior to system runtime and are viewed as estimates that can be different from actual specifications depending on the scale of the deployment and environmental characteristics. Dynamic specifications are

```
service Sensing(modality, numSamples, sensingFreq)
{
    implementation SensingC;

    parameters {
        modality = enum {"acceleration", "strain", "temperature", "humidity"},
        numSamples = range {100, 700000},
        sensingFreq = enum {25, 50, 100, 280}
    }

    static properties {
        energy = func1(modality, numSamples, sensingFreq),
        duration = func2(modality, numSamples, sensingFreq)
    }

    dynamic properties {
    }

    command startSensing(startTime);
    command abortSensing();

    event dataReady(sensorData);
    event sensingFailed(reason);
}
```

Figure 4.2: An abbreviated interface specification for a sensing service. To facilitate automated service selection and configuration, service properties can optionally be specified.

derived dynamically (Chapter 5) and reflect the environment and runtime characteristics. When available, dynamic service specifications overwrite the static properties for service evaluation and selection.

4.3.2 Service Composition

Service invocation may be subject to interactions with other services and the corresponding dependencies. Service dependencies can be considered in two main categories: *causal* and *temporal*. Causal dependencies include *data* and *logical* dependencies. For example, the data aggregation and flash storage services have data dependency with the sensing service that generates the sensor data. An example of a temporal dependency is network clock resynchronization, which is enabled after a certain interval after the preceding synchronization event.

In the actor model, the dependencies between services are resolved by message transfers between actors. In other words, services are initially blocked, and then enabled based on message arrivals. For instance, in the example depicted in Figure 4.3, the *Flash Storage* service waits for a *data ready* message from the actor representing the *Sensing* service before performing its task of storing data. In this context, a single service may enable (i.e., send a message) to a number of other services (actors). Also, one service may require a message from more than one other service in order to be enabled. A service can send messages to itself to enable the next round of operation.

We also use a service composition language to define the control flow between services, which is based on their dependencies and a sequence of invocations by the application. Figure 4.3 shows service dependencies and invocation for the SHM application described above using our service composition language. Each service is represented by an actor and arrows show message exchanges between actors. In this example, the *TimeSync* service is enabled after a certain time from the previous invocation, the *Sensing* service waits for the *TimeSync* service and enables the *LocalStorage* and *DataAggregation* services. As illustrated in Figure 4.3, concurrent execution of services is allowed provided that the corresponding dependencies are satisfied.

4.4 Adaptive Middleware Framework

In this section we discuss how our framework integrates the discussed components to enable dynamic service selection and configuration to address

```
// system initialization
System.init() -> TimeSync.startSync()[GatewayNode];

// main sensing cycle
TimeSync.syncAchieved()[SensorNodes]
    -> Sensing.startSensing(TIME + 10 s)[LeafNodes];
Sensing.dataReady(sensorData)[NODE]
    -> LocalStorage.store(sensorData)[NODE],
       DataAggregation.aggregate(sensorData)[NODE];
DataAggregation.complete()[LeafNodes] -> TimeSync.stopSync()[SensorNodes];

// periodic sensing
DataAggregation.complete()[LeafNodes]
    (delay = 60 m)-> TimeSync.startSync()[SensorNodes];

// fault handling
TimeSync.syncLost()[NODE] -> Sensing.abortSensing()[NODE];
```

Figure 4.3: Simplified service interaction specification for a structural health monitoring application.

changes in application requirements and environmental conditions. It is critical that the middleware framework must be implemented in a way that only incurs overhead costs when service interactions happen (service invocation by the application, data transfer between services, etc.) and not for the native implementation of the services and their interaction with the operating system (Figure 4.4). The latter would impose unacceptable resource use and latency costs on resource-constrained embedded systems.

4.4.1 Adaptation

We propose a model that breaks the static linkage and pre-determined customization of the service parameters in favor of *dynamic service composition*, where such actions take place at runtime. We combine *system-wide adaptation* in response to changing global conditions or application requirements with efficient, low-latency *local adaptation* in response to location-specific

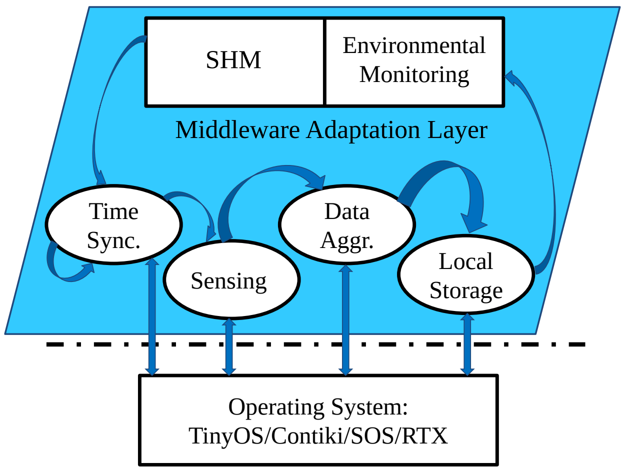

SHM
Environmental Monitoring
Middleware Adaptation Layer
Time Sync.
Sensing
Data Aggr.
Local Storage
Operating System: TinyOS/Contiki/SOS/RTX

Figure 4.4: An adaptive middleware framework. This framework mediates low frequency interactions between coarse-grained application components and middleware services, but not low-level interactions with the operating system.

events and conditions.

By *local adaptation* we refer to changes in behavior decided at an individual node or within a single middleware service. This is needed when local parameters and environmental conditions promote localized changes behavior. Some examples of local adaptation would be the change of resynchronization frequency (TimeSync service) and sensing parameters such as length of sensing and sampling rate (Sensing service) based on the energy reserves available at a node, or a change in routing metrics for certain links to route around a congested node (Routing service). Such changes are often in response to transient and ephemeral events or localized phenomena, and as such a centralized, coordinated response is neither needed or feasible with a low latency.

System-wide adaptation, on the other hand, is global and is required when the entire middleware layer must make coordinated changes that are neither localized nor limited to a single service. Global constraints are not known by any individual service or node and can change the aggregate function of the system. Global constraints can change system policies which in turn can change the acceptable range for parameters. An example requiring global adaptivity is adjusting the frequency of data collection for the environmental monitoring application, and thus affecting all of the middleware services, in response to lower than expected solar panel output due to a stretch of cloudy weather.

While diverse requirements exist on when and how middleware services need to adapt their behavior, we follow a minimalistic approach in specifying these requirements from the point of view of the middleware adaptation layer. Each service, when it is instantiated by the system, is assigned a *policy*, which gives it a range for one or more of its configuration parameters. The service is then free to make independent decisions about selecting the appropriate configuration value within that range, and can adjust it at any point due to local adaptation decisions. An example would be the DataAggregation service, which may be allowed to vary the bandwidth usage within a certain limit, trading off throughput for increased energy draw.

4.4.2 Two-level Actor Model

We want to maintain a separation of concerns between the principal functionality of the system represented by the composition of actors and the functionality of the adaptive middleware framework. The adaptive middleware framework is concerned with monitoring the relevant system state, exercising adaptive control over service selection as well as the configuration parameters of the deployed services, and the deployment and reconfiguration of service instance actors.

To achieve this objective, we develop a *two-level actor model*. At the base

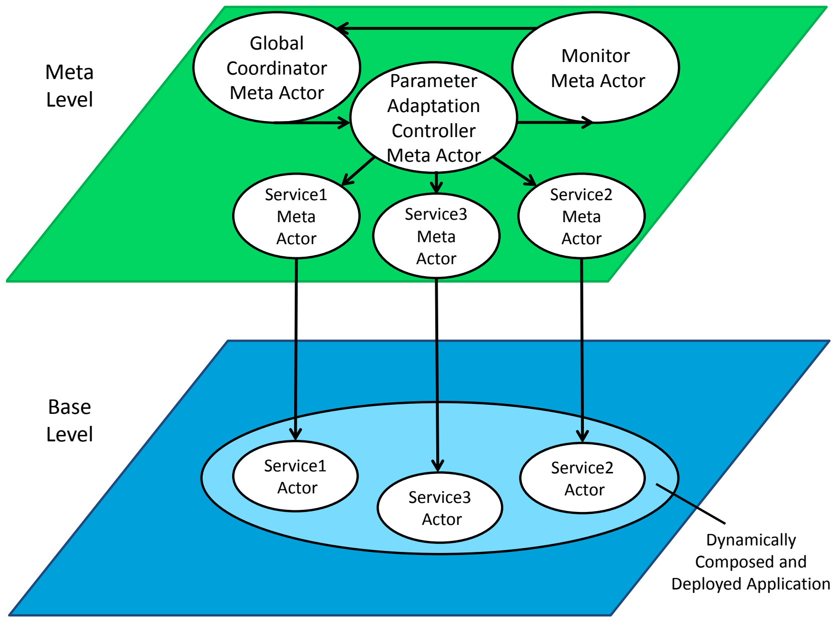


Figure 4.5: Two-level actor model for adaptive sensor network middleware.

level, the principal functionality of the application and middleware services is represented by a collection of *base-actors*, which implement the functional behavior of the system. The adaptation is realized through the meta-level, represented by a meta-actor associated with each base-actor, as well as additional specialized actors responsible for coordination, monitoring, and distributed control (Figure 4.5). All actor types communicate with each other exclusively through message passing.

4.4.3 Actor Roles

As shown in Figure 4.5, actors can be of two types: base-actors and meta-actors. Base-actors represent the services that constitute an application. Several types of meta-actors exist in the system. Below, we briefly introduce each meta-actor.

Monitor The responsibility of the Monitor is to periodically provide measurements of important system values (energy levels, load, etc.) to other meta-actors. It does so by collecting information from the system about the behavior of each node and the deployed services. In order to keep the network monitoring overhead at minimum, the Monitor uses aggregate information from relatively long durations of network operation. In this work, we focus on energy management and leave investigation of other system values for future work.

The Monitor has two main goals. First, it attributes aggregate energy consumption to individual services that are running in the network. Second, it detects energy anomalies and finds their cause. An energy anomaly can occur due to selection of high energy services, anomalous interaction of different services, or hardware issues. In Chapter 5 we discuss how the Monitor achieves these goals in detail.

The Monitor shares dynamic service specifications and the detected energy anomalies with Global Coordinator meta-actor to ensure efficient service selection and parameterization.

Global Coordinator The Global Coordinator meta-actor (GC) is responsible for deploying applications and services within the WSSN. It uses application requirements and service specifications to derive an optimal service selection and parameterization and provides it to the Parameter Adaptation Controller meta-actor. Service selection is done using a CSP solver module within the GC meta-actor. For service selection, the GC meta-actor uses service properties provided by service developers, as well as dynamic properties derived by the Monitor meta-actor. The service selection module of the GC meta-actor is called *S4* and is discussed in detail in Chapter 6.

Parameter Adaptation Controller The Parameter Adaptation Controller meta-actor (PAC) is responsible for distributing updates received from the Global Coordinator meta-actor regarding service and parameter selection as well as dynamic service properties to the corresponding Service meta-actors. PAC provides Service meta-actors with acceptable range of parameter values which are derived from information provided by the GC meta-actor.

PAC is responsible for the initial instantiation of all of the services and also for the dissemination of global policy changes. In actor semantics, actors can only communicate with a set of actors that they *know*, that is they have been given the unique *actor name* for them. Since PAC creates all the other actors in the system, it knows their addresses and can disseminate them as needed to the service actors. This would include the names of the actors from which inputs and dependency satisfaction notifications can be delivered to a service actor, as well as the list of actors that the service needs to notify upon completion or some other event. When new service actors are added to the system, PAC can also send notifications to existing services to update the dependency relationship graph.

Once the service instances are deployed and notified of their dependencies by PAC, all subsequent interactions between services are direct, without going through the mediator actor. This allows for significant savings in network traffic volume, particularly for service instances collocated at the same node, where message sends are translated into direct function calls.

Service meta-actors Service meta-actors are tightly coupled to the base level service actors and are responsible for enacting low-latency, localized control over the service parameters, within the range dictated by the PAC.

4.4.4 Runtime Support System

Besides static configuration and parametrization at instantiation time, dynamic service composition and system-wide adaptation require additional support from the middleware system at runtime. For our actor-based approach, this involves the asynchronous message passing functionality for service interactions, initial service deployment and configuration, and the runtime monitoring system.

The runtime system can be though of as a graph, with each node representing a service interface, or an actor. The edges define service dependencies (causal and temporal) and are executed by message transfers between actors. The services send asynchronous messages that may contain parameters and the inputs required for service invocation. This model allows for concurrent execution of services while satisfying data and temporal constraints. In our implementation, links between service instances are determined at run time.

We note that there is not a single centralized location where the graph data structure exists in the system. Rather, the data structure is fully distributed, with parts of it being stored with each service instance (e.g., service configuration parameters) and others being part of the messages in transit (invocations, dependency notifications, etc.). In this way, the graph can be evaluated recursively and concurrently without requiring expensive round-trip message exchanges with a centralized entity for every service interaction.

4.5 Discussion

In this chapter we presented an overview of our architecture. I-AdMiN has a two-layer architecture. The base layer is comprised of services that constitute an application. Service instances and their interactions are represented by actors.

The meta layer is the main focus of this work. This level contains meta-actors that are responsible for automatic service selection and parameterization and provide dynamic adaptation in response to application and environmental changes. It is worth noting that the main distinction between base-actors and meta-actors and between the different meta-actors is only their responsibilities. Each meta-actor is given distinct responsibilities that together ensure the efficient operation of the sensor network application.

Let us demonstrate I-AdMiN's work flow. Assume that a pool of services is available and a set of application requirements are provided. Each service is associated with initial specifications. These specifications are generally provided by the service developer and do not reflect the effect of the surrounding environment. In the beginning, the application developer uses the service selection module from the Global Coordinator to select a set of services that satisfy application constraints. Once the sensor network application is deployed, the Monitor collects runtime information that are used in deriving dynamic service specifications. Dynamic service specifications are used to update the initial service descriptions provided by service developer and replace the static specifications. Another role of the Monitor is to detect instances where application constraints are violated and find the cause. The cause of a constraint violation can be an anomalous service or service interaction, or hardware issues. The derived service specifications and constraint violations are shared with the Global Coordinator to update service selection and/or parameterization if necessary. Any change in service selection or parameterization is shared with the Parameter Adaptation Controller to be distributed in the system. If there is a change in the range of acceptable parameters of a service, PAC sends a message to the corresponding Service meta-actors informing them of the change. Similarly, PAC sends Service meta-actors messages to indicate new service instantiation or deactivation. During system runtime, the Monitor updates its information on service specifications and constraint violations. This allows GC and PAC to adjust service selection and parameterization in response to changes in application requirements or the environment.

In the next chapters we will discuss the modules that implement each meta-actor's functionality. The Monitor is comprised of two main modules. The first module attributes dynamic service properties to individual services using aggregate information. The second module detects constraint violations during sensor network runtime and finds the cause. Both these modules are discussed in detail in Chapter 5. The Global Coordinator's main module is called S4 and is responsible for automatic service selection in a way that application constraints are satisfied(Chapter 6). S4 uses dynamic information from the sensor network deployment made available by the Monitor. Finally, the Parameter Adaptation Controller meta-actor and Service meta-actors work together to dynamically adapt service configuration in response to information from GC.

CHAPTER 5

MONITORING SERVICE ENERGY CONSUMPTION .

In this chapter we discuss our monitoring system. Due to the importance of energy consumption on the lifespan of WSSN applications, we focus on deriving energy characteristics of services. In the WSSN domain, static properties such as energy consumption that are derived by service developers may not reflect the exact behavior of services once they are deployed. This is due to varying hardware platforms and environmental conditions. We thus propose dynamic and on-line profiling of services during the WSSN deployment. The need for an on-line monitoring system is motivated by the following factors.

Hardware platform. There are many different hardware platforms used in the deployment of WSSNs. TelosB [97], MICAz [98], and Imote2 [99] are a few popular off-the-shelf platforms. These platforms have different processor, memory, radio, and energy characteristics which affect their behavior. To add to the variability, each of these platforms can be used with different sensors (e.g. sensor boards that mount on the platform such as ISM400 sensor board [100] versus analog sensors such as soil moisture sensors [101]). Exact specification of services such as their energy consumption depend on the underlying platform which is not known at the time of service implementation. Figure 5.1 compares the energy consumption of a service called *Remote Sensing* on Imote2 when two different sensors are attached to it. The Remote Sensing service collects sensor data from multiple sensors and comprises seven sequential phases: setup, communication, computation, net-

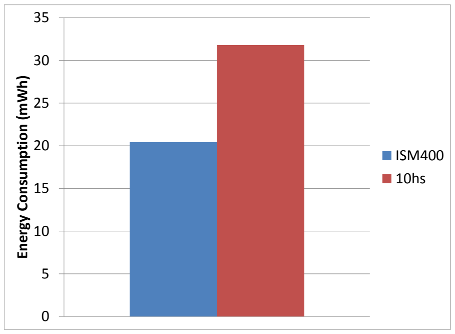


Figure 5.1: Comparison of energy consumption of RemoteSensing service deployed on Imote2 platform with two different sensors. Soil moisture sensor Decagon 10hs shows higher energy consumption compared to ISM400 sensor that measures acceleration.

work initialization, idle, sensing and data storage. Service parameters and runtime are exactly the same in the two cases and the only difference is in the sensor board.

Environmental characteristics. Deployment and environmental characteristics can affect service specifications. Properties such as runtime and energy consumption can be very different in an indoor lab test and a full-scale outdoor deployment. For example, the time needed for route construction in a multi-hop routing algorithm increases as the number of nodes, their distance and radio communication obstacles increase.

Non-functional dependencies. Functions performed by services that comprise an application may affect the non-functional properties of other employed services. This phenomena is often platform dependent and changes service properties. For example, according to the time-keeping anomaly discussed in Chapter 3, changing the frequency of an embedded processor can negatively impact time synchronization [96]. The impact on time synchronization has to be considered if a service that changes the frequency for energy conservation is to be used in an application requires tight time synchronization. It is often the case that such dependencies and interactions are not known in advance.

Resource limitations of WSSNs imply that any monitoring system should be light weight and energy efficient. A monitoring system that can estimate service properties without needing information about their parameters and environmental conditions is desired. Such a system should provide frequent updates on service properties as deployment conditions change. We should note that the monitoring system under discussion is different from on-line monitoring of program properties which investigates temporal logic and causal dependencies.

We follow a data-driven approach for the monitor where coarse-grained data is used to derive service-specific information. The proposed monitoring system comprises of two parts. The first part estimates service energy consumption and the second part identifies and isolates energy anomalies. In both of these parts our focus is on energy consumption and will use our ISHMP Jindo Bridge deployment [2] as the basis of our analysis. We leave estimation of service properties other than energy consumption and the detection of other constraint violations for future work.

5.1 Analysis Data

In an effort to design a monitoring system with the least possible overhead, we leverage existing data logging services of the ISHMP Jindo Bridge deployment [2]. Therefore, no additional overhead is incurred for data collection. The deployed Illinois SHM Toolsuite stores output data from multiple services in the form of text files. When scheduled to run, each service collects information from the network and sends it to the base station where it is stored. Examples of such services include *RemoteSensing* and *AutoUtil*. RemoteSensing is a distributed service used to collect measurements of acceleration from all nodes. AutoUtil collects measurements of battery voltage, temperature and light. The output from each service is a text file showing data collected from each node. The output files of different services can be easily differentiated by their size and output format.

For our analysis, we focus on energy consumption and exploit existing measurements to deduce the effect of each service on energy consumption. The battery voltage measurement from the AutoUtil service is a viable option for this purpose. We can read battery voltage measurements before and after a set of services were running and estimate total energy consumption. The challenge is finding the effect of each individual service on energy consumption from aggregate data that shows energy consumption when a combination of services were running. In the following sections we discuss how we use a large number of such data points to find per-service estimates of energy consumption.

5.2 Estimating Service Energy Consumption

In this section we provide a method to attribute aggregate energy consumption to individual services. For this purpose, we analyze each node individually and then use the mean of estimated energy across all nodes as the final estimate. The reason for considering each node individually is that in the deployment from which we collected the data each node is equipped with a solar panel. The pattern in the battery voltage draw of a node can vary based on the direction of its solar panel and weather conditions. Once estimates are gathered across all nodes, we combine them to fortify the final result.

In order to get necessary data for estimating service energy consumption, the monitor reads all output files from the services that are running. Output files from AutoUtil service are the only files that contain battery voltage information. For each node, the monitor reads battery voltage in two consecutive AutoUtil files and takes the difference as the voltage draw. It also counts the number of AutoUtil and RemoteSensing services that ran during this time on the node under consideration. The battery voltage reading is an 8 bit value that does not have the required accuracy to read small changes in battery voltage and may return the same value when battery voltage draw is small. For this reason, the monitor continues on reading the output files when the battery voltage draw is zero until it reaches an AutoUtil output file that shows battery voltage drain. In such cases, all AutoUtil and RemoteSensing occurrences are counted for the participating nodes. Data points that include durations of battery charge from the solar panel are discarded. Table 5.1 shows a sample of data generated in the monitor for service energy estimation.

Table 5.1: Sample processed data from Jindo Bridge deployment. Each data point shows node id, the amount of battery voltage draw, number of RemoteSensing occurrences and number of AutoUtil occurrences.

Node Id	**Battery Voltage Draw**	**AutoUtil Count**	**RemoteSensing Count**
3	0.093	2	6
3	0.011	2	2
3	0.083	2	2
69	0.062	2	1
69	0.031	1	0
86	0.041	2	4

5.2.1 Approach

Let's assume the application under consideration has N services $S = \{s_1, s_2, ..., s_N\}$. Each data point d_i from the set of all data points D includes the following:

- A duration t_i at which the data is collected.
- A node ID n_i for which the data is collected.
- A set of services s_k, $k \in 1..M$ that were running during t_i.
- For each service s_k, a count c_k of the number of times s_k was running during t_i.
- Total battery voltage draw $VDrop_i$ during t_i.

The goal is to find $VDrop_{s_k}$ for each service in $S = \{s_1, s_2, ..., s_N\}$ using the aggregate energy consumption data in D. In other words, we have equations of the form

$$VDrop_i = \sum_{i=1}^{N}(c_k * VDrop_{s_k}) \tag{5.1}$$

where $VDrop_{s_k}$s are unknown. We are not able to solve this as a system of linear algebraic equations as the noise in the data creates a conflicting system of equations (e.g. rows 2 and 3 in Table 5.1). Instead, we use regression to fit a linear line with $VDrop_i$ as the dependent variable, c_k as the independent variables and $VDrop_{s_k}$s as the unknown parameters.

Let us investigate data characteristics before discussing our method. Since AutoUtil is the only service that measures battery voltage, we can find many data points with only AutoUtil as the active service (i.e. $c_k = 0$ for $s_k \neq AutoUtil$). This means that we can analyze AutoUtil separately in advance and find its battery voltage draw estimate. We can then subtract the estimated battery voltage attributed to AutoUtil from the dependent variable and continue our analysis for the rest of the independent variables. Note that we follow the same method in analyzing AutoUtil and all other services but do so in two separate steps for increased accuracy.

We have observed a non-negligible amount of noise in data points with small AutoUtil and RemoteSensing count. The high variance of battery voltage draw for these data points, shown in Figure 5.2 and Table 5.2, is an indicator of this noise. This phenomenon is due to battery characteristics. Accurate battery voltage readings require a period of *battery rest time* following a duration of activity [102]. During the rest time, the battery will gain some of its charge back allowing a more realistic reading of battery drain. Therefore, consecutive battery voltage readings that are within a short amount of time are subject to a high amount of noise and in many cases show an inaccurate spike in battery voltage draw. The smaller the AutoUtil and RemoteSensing counts, the shorter the amount of time between battery voltage readings, and the less reliable the data is.

Table 5.2: Variance of battery voltage drain for different AutoUtil counts. Groups with smaller values of AutoUtil count show higher variance. Smallest amount of variance is observed for AutoUtil count values of 7 and more.

AutoUtil Count	**Variance**
1	0.0047
2	0.0028
3	0.0021
4	0.0017
5	0.0023
6	0.0032
≥7	0.00041

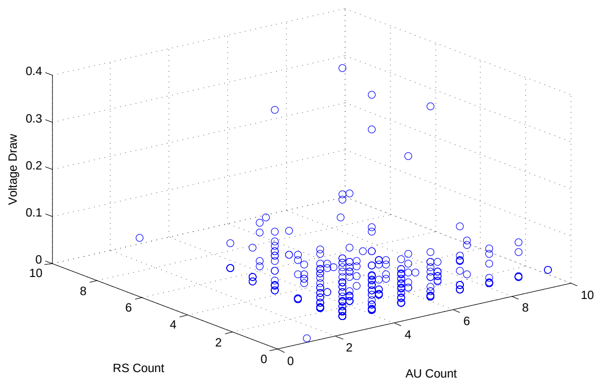


Figure 5.2: Scatter plot showing AutoUtil count, RemoteSensing count and voltage draw.

The non-constant amount of variance for different values of the independent variable, as observed in Figure 5.2 and Table 5.2, suggests the presence of *heteroscedasticity* in the data. Let us investigate this property more thoroughly. Consider the regression model:

$$Y_i = \beta_0 + \beta_1 Xi + \epsilon_i \tag{5.2}$$

Let residual (observed error) $e_i = Y_i - \widehat{Y_i}$ and true error $\epsilon_i = Y_i - E\{Y_i\}$. For the above regression model, the ϵ_is are assumed to be independent normal random variables with mean 0 and constant variance σ_2. If the model is appropriate for the data at hand, the observed residuals, e_i, should then reflect the properties assumed for the ϵ_i [103]. However, when the error variance is not constant over all cases, we have *heteroscedasticity* in the dataset. Heteroscedasticity is inherent when the response in regression analysis follows a distribution in which the variance is functionally related to the mean [103]. When heteroscedasticity prevails but other conditions of regression model are met, the estimated regression coefficients are still unbiased and consistent, but they are no longer minimum variance unbiased estimators [103]. Table 5.3 shows the error variance, the variance of the residuals for the unweighted least squares regression, for groups of data, each with a value of AutoUtil count. Data is grouped according to the value of the independent variable (AutoUtil count in this example). The data shown in this table confirms that heteroscedasticity is present for our data set.

When the variance of the observed values are unequal (i.e. heteroscedasticity is present), but no correlations exist among the observed variances, weighted least squares (WLS) regression can be used. Weighted least squares regression is a special case of generalized least squares (GLS) regression. Unlike simple linear regression which weights each Y observation equally, WLS criterion assigns different weights:

$$Q_w = \sum_{i=1}^{n} w_i (Y_i - \beta_0 - \beta_1 X_i)^2 \tag{5.3}$$

where w_i is the weight of the ith observation. WLS regression is a good fit for our problem due to the high noise observed in data points with small values of AutoUtil and RemoteSensing count. It will allow us to give a higher weight to data points that show more accurate values while not ignoring the less reliable data points completely.

In order to use WLS regression, we should determine the appropriate weights. We first investigate if the error term variance has a simple relationship with each group of the independent variable. Table 5.4 shows possible relationships between AutoUtil count groups and error term variance. Since none of these relationship show the required stability, we will use the reciprocal of the variance as the weight. Once the weight of each group

Table 5.3: Variance of error for different values of AutoUtil count. Error variance is the variance of the residuals for the unweighted least squares regression on each group.

Group	**AutoUtil Count**	**Error Variance**
1	1	0.0029
2	2	0.0026
3	3	0.0027
4	4	0.0039
5	5	0.0017
6	6	0.0019
7	$\geq$7	0.0044

Table 5.4: Analysis of the relationship between error term variance σ^2 and X_j for groups $1..j$. If a group has more than one value for the independent variable the midpoint is used as X_j.

Group	σ_j^2/X_j	σ_j^2/X_j^2	$\sigma_j^2/\sqrt{X_j}$
1	0.0029	0.0029	0.0029
2	0.0013	0.00065	0.0018
3	0.0009	0.003	0.0015
4	0.000975	0.00024	0.00195
5	0.00034	0.00068	0.000729
6	0.000316	0.0000527	0.000778
7	0.000624	0.0000897	0.016

is determined, it is given as one of regression parameters to determine the unknown variables $VDrop_{s_k}$.

5.2.2 Experimental Results

In order to evaluate our method of attributing aggregate battery voltage draw to individual services, we use data gathered from ISHMP Jindo Bridge deployment. We compare our results with measurements of battery voltage draw on several nodes.

Table 5.5 shows characteristics of the data used in our analysis.

Table 5.5: Dataset characteristics for data gathered from ISHMP Jindo Bridge deployment.

Dataset	**Number of Files**	**Number of Nodes**	**Total Number of Data Points**
Jindo Deck 2010	1241	33	7424
Jindo Deck 2012	3805	32	23845

We first separate data points at which AutoUtil is the only active service. For Jindo Deck 2010 dataset, this includes 5758 data points which is 77% of all data. The Jindo Deck 2012 dataset contains 20721 data points which is 87% of all data. For the datasets with only AutoUtil as the active service, we run weighted least squares regression crossing the origin. In order to determine the wieghts for the WLS regression, we first group the data based on the value of AutoUtil count. Table 5.6 shows these groups and their characteristics. For the data points in each group, the reciprocal of the variance is considered as the wieght. We discard data points with AutoUtil count of 1 in this analysis due to their high noise. We first do the analysis on a per-node basis and then take the mean of the per-node estimates as the final result. Table 5.7 shows the estimated value for battery drain attributed to AutoUtil service using the discussed method.

Table 5.6: Dataset groups based on AutoUtil count value.

Group	**AutoUtil Count**	**Group Size**	
		Jindo Deck 2010	Jindo Deck 2012
1	1	3247	11949
2	2	1128	3447
3	3	613	2313
4	4	367	1272
5	5	148	656
6	6	84	455
7	≥7	171	629

Table 5.7: Estimated battery voltage drain for AutoUtil service using WLS regression.

	Jindo Deck 2010	**Jindo Deck 2012**
Mean of Estimated AutoUtil Battery Voltage Draw Across all Nodes (V)	0.0021	0.0048
Variance of Estimated AutoUtil Battery Voltage Draw Across all Nodes	7.3841e-007	6.11e-006

Table 5.8 shows the mean and median of battery voltage draw for data points with only AutoUtil as the active service and compares them with results from WLS regression. For this comparison we take data points with only AutoUtil as the active service and divide the battery voltage draw value by the AutoUtil count. As shown in this table, the mean an median have much higher values that the estimated value using WLS regression. This is because in calculation of mean and median all data points, including data points with high levels of noise, are given the same weight. As discussed previously, battery voltage reading inaccuracies cause a spike in battery drain for data points with small AutoUtil counts, resulting to a high value for average battery drain readings.

Table 5.8: Mean and median of battery voltage draw values for data points with only AutoUtil as the active service compared to results from WLS regression.

	Jindo Deck 2010	**Jindo Deck 2012**
Mean Battery Voltage Draw for AutoUtil	0.024	0.0331
Median Battery Voltage Draw for AutoUtil	0.01	0.0121
Battery Voltage Draw Estimate for AutoUtil	0.0021	0.0061

Once the battery voltage drain estimate for AutoUtil service is determined, we subtract it from the aggregate battery voltage draw in the rest of the data and follow the same approach to determine battery voltage drain attributed to RemoteSensing. Table 5.9 shows data grouping according to RemoteSensing values and Table 5.10 shows the estimated battery voltage draw attributed to RemoteSensing using WLS regression. The estimate for battery voltage drain of RemoteSensing is higher for the Jindo Deck 2010 dataset. This is expected since the 2012 version of RemoteSensing service included energy optimizations which included changing CPU frequency in different phases of the application.

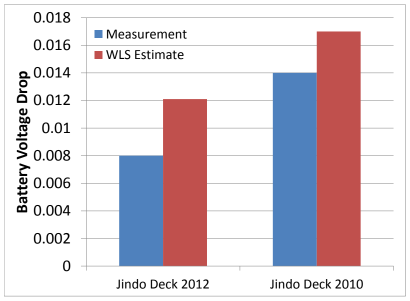
Table 5.9: Dataset groups based on RemoteSensing count value.

Group	**RemoteSensing Count**	**Group Size**	
		Jindo Deck 2010	Jindo Deck 2012
1	1	1049	2217
2	2	302	612
3	$\geq$3	60	313

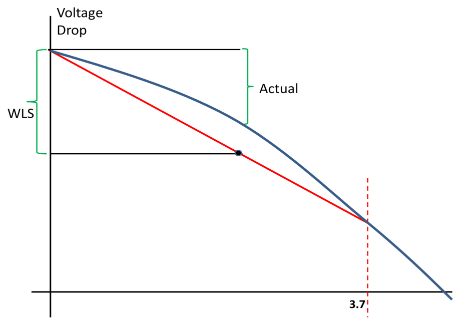
Figure 5.3a compares our results with lab measurements of battery voltage drain when running RemoteSensing service. The measurements are not conducted on the Bridge where the analysis data is gathered due to difficulty in getting node access and may thus include an error that is caused by differ-

Table 5.10: Estimated battery voltage drain for RemoteSensing service using WLS regression.

	Jindo Deck 2010	**Jindo Deck 2012**
Mean of Estimated RemoteSensing Battery Voltage Draw Across all Nodes (V)	0.0170	0.0121
Variance of Estimated RemoteSensing Battery Voltage Draw Across all Nodes	5.2721e-005	7.1259e-005



(a) Laboratory measurements and battery voltage draw estimates of RemoteSensing service



(b) Effect of linearization.

Figure 5.3: Comparison of battery voltage draw estimates.

ence in environmental characteristics. However, in an attempt to fortify our comparisons for the services under investigation, we have chosen the same parameters and settings as the Jindo Bridge deployment. Another source of error is linearization. As discussed previously, we have assumed that battery voltage drop is linear which means that the WLS method overestimates the battery voltage drop (Figure 5.3b). We have reduced the effect of linearization by limiting the battery voltage readings (nodes with battery voltage less than 3.7V are disregarded).

5.3 Detecting and Isolating Energy Spikes

In this section we discuss our method of detecting energy spikes and isolating the cause. When new data from the sensor network becomes available to the Monitor, it first detects and diagnoses instances of high energy consumption. This information is then used in a Bayesian belief revision system where each hypothesis states the probability of having high energy consumption when a set of services are running. The probability of the hypotheses which is inferred using Bayesian methods is used at the time of service selection by the S4 module.

We focus on three classes of issues: hardware issues, service issues and interaction issues. Interaction issues can happen between services or between services and hardware.

Our analysis is based on the assumption that samples are drawn from a normal distribution. We will confirm this assumption in Section 5.3.4. Each data point that is investigated in the detection and isolation steps includes node ID n_i, duration t_i, service counts c_k for services s_k ($k \in 1..M$) and total energy consumption E_i.

5.3.1 Detecting Energy Spikes

We follow a straight forward approach in detecting energy spikes. Assume the independent variable (energy consumption) follows a normal distribution $N(\mu, \sigma^2)$ with μ and σ^2 estimated from the data. A data point with total energy consumption E_i is detected as an energy anomaly if the probability of E_i coming from $N(\mu, \sigma^2)$ is less than a threshold θ. We will see in Section 5.3.4 how the choice of θ affects the detection of energy anomalies.

5.3.2 Isolating the Cause of Energy Spikes

Once an energy spike is detected, the monitor determines what caused it. Energy anomalies are categorized as hardware issues, service issues, service interaction issues and service and hardware interaction issues.

Given a data point with high E_i according to the detection step, we examine all services s_k $(k \in 1..M)$ that were running during t_i as well as the hardware, represented by node ID, n_i. We call each of the services s_k and the node ID, n_i a factor and aim to find the factor that contributes to high energy consumption. In order to take into account the effect of service interactions we add another factor for each j-combination c_j of the set of services s_k $(k \in 1..M, j \in 1..M)$ that were active during t_i. We also add a factor for the interaction between node n_i and each j-combination of the set of services.

For each factor f_i we examine all data points and divide them into two classes: one that contains f_i, denoted by D_i, and one without f_i, denoted by $D_{\bar{i}}$. Specifically, we use two normal distributions $N_i(\mu_1, \sigma_1^2)$ and $N_{\bar{i}}(\mu_2, \sigma_2^2)$ to approximate D_i and $D_{\bar{i}}$, and estimate the means and the variances from data. We adopt *Student's t-test* to determine whether D_i and $D_{\bar{i}}$, with unequal sizes and unequal variances, are drawn from the same distribution, with 95% confidence error bounds. If the two distributions are different and $\mu_1 - \mu_2 \geq \epsilon$, then factor f_i is considered to be significant with respect to the detected high energy consumption. Here, ϵ is an adjustable threshold and is determined based on the desired granularity of isolation.

If the factor f_i that is identified as the cause of high energy consumption represents a service, the monitor classifies it as a *high energy service*. Similarly, if a combination c_j of services is identified during the isolation step, a selection with services in c_j is classified as high energy. High energy factors that include the node n_i are reported to the system administrator for further analysis. A high energy consumption instance that is correlated to the sensor node can be caused by a variety of issues such as hardware platform problems, solar panel direction and connection, connection to the sensor, etc.

5.3.3 Integrating Energy Anomaly Information

Optimized service selection necessitates sharing of information on high energy services and anomalous interactions as identified in the detection and isolation steps with the service selection module. We have designed a *Bayesian belief revision* system for this purpose that is continuously updated as the monitor receives new information from the deployed sensor network. Bayesian methods provide probabilistic predictions on a set of hypotheses which become more or less probable as more data is observed [104]. Below, we will describe our set of hypotheses and how they are revised to address new information from the detection and isolation steps.

Assume the service repository includes N services $s_1..s_N$. For each j-combination of services $c_j, j \in 1..N$, hypothesis h_j states the belief that running this set of services has high energy consumption. We will therefore have $H = 2^N$ hypotheses. We assign each hypothesis a probability value $P(h_j)$ that quantifies the degree of confidence in the hypothesis. We use Bayes theorem to dynamically derive $P(h_j)$ as new data becomes available from the sensor network deployment and the detection and isolation steps. In other words, we aim to calculate $P(h_j|D)$ where D is the newly available data:

$$P(h_j|D) = \frac{P(D|h_j).P(h_j)}{P(D)} \tag{5.4}$$

where $P(h_j|D)$ is the posterior probability of hypothesis h_j that c_j has high energy consumption and reflects our confidence that h_j holds after we have seen the training data D. $P(h_j)$ is independent of D and is the prior probability of h_j and may reflect background knowledge about h_j. $P(D|h_j)$ is the probability of observing D if hypotheses h_j holds. Finally, $P(D)$ is the prior probability that training data D will be observed [104]. Consider event A_j, where the set of services in c_j are running. We can assume that events $A_1..A_N$ are mutually exclusive with $\sum_{i=1}^{n} P(A_i) = 1$. We thus have:

$$P(D) = \sum_{i=1}^{N} P(D|A_i).P(A_i) \tag{5.5}$$

According to 5.4, the probability of a hypothesis is determined by combining prior knowledge with observed data. Prior knowledge can be provided by: (1) stating a prior probability of each hypothesis and (2) stating a probability distribution over observed data for each possible hypothesis [104].

In the beginning of system runtime, prior probabilities for the hypotheses can be determined by either using static service properties provided by service developers or setting the same prior probability for all hypotheses. We use the latter approach for two reasons. First, many already developed services lack a description of their non-functional properties which impedes the ability to derive prior probabilities. Second, assigning equal prior probabilities to services allows a fair comparison of services as more information becomes available from the deployment. Therefore, in the beginning we have $P(h_i) = P(h_j) = p$ for all $i, j \leq H$.

The posterior probability $P(h_i)$ of each hypothesis h_j is recalculated as the isolation step identifies service combinations that cause energy anomalies. We use Jeffrey's rule ([105], and [106]) to account for any uncertainty in the new coming information. Assuming that the new data D is attached with a probability α, we have:

$$P(h_j|(D, \alpha)) = \alpha P(h_j|D) + (1 - \alpha) P(h_j|\bar{D}) \tag{5.6}$$

For each hypothesis h_j, $P(D|h_j)$ is 1 if the isolation step has identified c_j as a source of high energy consumption and is 0 otherwise. $P(h_j)$ is the prior probability (with respect to D) of having high energy consumption when running c_j. $P(D)$ is calculated using equation 5.5.

As the probabilistic predictions of hypotheses are modified the monitor sorts them in descending order which means services or service interactions with the highest predicted probability of having high energy consumption are at the top of the list. This ranking of services and their interactions based on their energy consumption is shared with the service selection module for optimization of future service selection and is mostly helpful when the service pool includes multiple service implementations for the same functionality. We will discuss service selection in detail in Chapter 6.

5.3.4 Experimental Results

For the evaluation of our detection and isolation methods we use a simplified SHM application that is comprised of three main services: a *sensing* service, a *time synchronization* service and an *energy management* service. The collected data is from two sources. We use the battery voltage data used in the previous section for the sensing service. Data for time synchronization and energy services is from our laboratory measurements of the amount of current draw in Imote2 when these services are running. We have done extensive measurements for all combinations of available time synchronization and energy management services and calculated the corresponding energy consumptions. We have considered a sensing service called *RemoteSensing*, a time synchronization service called *TimeSync*, and two energy management services called *DVS* and *TS-DVS*. RemoteSensing is the same sensing service discussed in the previous section. TimeSync is an implementation of the *Flooding Time Synchronization Protocol*. DVS and TS-DVS both use dynamic voltage and frequency scaling to reduce the total energy consumption when the application is comprised of different phases. The energy consumption data for TimeSync, DVS and TS-DVS consists of aggregate energy consumption (in mWh) for approximately every 15 minutes of system runtime. It includes more than 200 energy consumption measurements that range between 56.22 mWh and 83.23 mWh. The mean value of energy consumption is 67.08 mWh and the standard deviation is 5.08.

As discussed previously, our analysis is based on the assumption of normality. In order to confirm this assumption, we perform the *JarqueBera test* which is a goodness of fit test for normal distribution. [107] shows that Jarque-Bera test is superior to Kolmogorov-Smirnov and Cramervon Mises type, Shapiro-Wilk test, and Kuiper test for symmetric distributions with medium up to long tails and for slightly skewed distributions with long tails.

JarqueBera test investigates the null hypothesis that the sample data has the skewness and kurtosis of a normal distribution (samples from a normal distribution have an expected skewness of 0 and an expected excess kurtosis of 0). In other words, the null hypothesis is the joint hypothesis that both skewness and excess kurtosis are zero. The Jarque-Bera test statistic is defined as:

$$JB = \frac{n}{6}(S^2 + \frac{1}{4}(K-3)^2) \tag{5.7}$$

where n is the number of observations, S is the sample skewness, and K is the sample kurtosis. S and K are defined as:

$$S = \frac{\widehat{\mu}_3}{\widehat{\sigma}_3} = \frac{\frac{1}{n}\sum_{n}^{i=1}(x_i - \overline{x})^3}{(\frac{1}{n}\sum_{n}^{i=1}(x_i - \overline{x})^2)^{3/2}} \tag{5.8}$$

$$K = \frac{\widehat{\mu}_4}{\widehat{\sigma}_4} = \frac{\frac{1}{n}\sum_{n}^{i=1}(x_i - \overline{x})^4}{(\frac{1}{n}\sum_{n}^{i=1}(x_i - \overline{x})^2)^{2}} \tag{5.9}$$

For the null hypothesis h that our sample data comes from a normal distribution, the Jarque-Bera test returns $h = 0$ which means that it cannot reject the null hypothesis at 5% significance level.

We first examine our anomaly detection method. Our data has normal distribution $N(\mu, \sigma^2)$ with $\mu = 67.08$ and $\sigma^2 = 25.85$. Let P_i denote the probability that a data point D_i with total energy consumption E_i is drawn from this normal distribution. D_i is identified as an energy spike if $E_i > \mu$ and $P_i < \theta$. Table 5.11 shows the effect of θ on the percentage of data points that are considered as energy anomalies. We consider $\theta = 5\%$ for our analysis.

Table 5.11: Effect of threshold θ on the percentage of data points regarded as energy anomalies.

θ	**Energy Anomaly Percentage**
5%	11%
3%	5%
1%	1%

Let us investigate the cause of the detected energy anomalies. In our measurement data for TimeSync, DVS and TS-DVS, the node and the sensing service were kept constant. Also, the two energy management services cannot run together. We thus have a total of 5 factors that should be analyzed for their significance towards the energy spike: 3 individual factors, one for each of the services *TimeSync*, *DVS* and *TS-DVS* and 2, two-way interaction factors. The Jindo Bridge data for RemoteSensing includes data points were either RemoteSensing, AutoUtil or both were running. Since here we are only interested in RemoteSensing, we will have one factor for this service. We therefore have a total of 6 factors to consider.

For each factor f_i, we divide the entire dataset to two classes D_i and $D_{\bar{i}}$ and perform the Student's t-test. Tables 5.12 and 5.13 show the results of the t-tests. We have separated RemoteSensing results from the rest of the services because their data source is different. The energy data for RemoteSensing is from node battery drain measurements in Jindo Bridge which are done at irregular intervals. Data for the other services shows energy consumption of the entire application for durations of 15 minutes. Based on these results and a critical value of 0.05, the TimeSync service's effect on high energy consumption is significant. This is confirmed by our laboratory measurements for the energy consumption of the TimeSync service. Our measurements show that every run of TimeSync consumes 1.89 mWh which is high for a simple time synchronization service. RemoteSensing also shows high energy consumption. Laboratory measurements for RemoteSensing shoe and energy consumotion of 11.6 mWh.

Another more interesting result is that the DVS service does not have a significant effect on high energy consumption but when it runs with the TimeSync service, the energy consumption significantly increases. Running the alternative energy management service TSDVS with the TimeSync service does not show such an increase in energy consumption. At a first glance, this result seems unexpected since and energy management service is designed to reduce total system energy consumption. We have investigated this phenomenon in [96]. We have found that in certain embedded platforms, the

CPU frequency switching actions of a dynamic voltage scaling (DVS) service has an adverse effect on processor-based clocks causing the time synchronization service to run more frequently to maintain clocks within a fixed error bound. Since the time synchronization service has high energy consumption, running it more frequently causes energy spikes.

Table 5.12: Student t-test results for three services: TimeSync, DVS and TS-DVS.

Factor	D_i		$D_{\bar{i}}$		**T-test**
	Mean	**Variance**	**Mean**	**Variance**	
TimeSync	67.80	19.69	61.88	40.58	1.14e-4
DVS	67.36	25.26	66.52	26.92	0.27
TS-DVS	66.52	26.92	67.36	25.26	0.27
TimeSync and DVS	68.88	7.14	65.01	39.55	1.5e-07
TimeSync and TS-DVS	67.25	23.91	67.01	26.76	0.76

Table 5.13: Student t-test results for RemoteSensing.

Factor	D_i		$D_{\bar{i}}$		**T-test**
	Mean	**Variance**	**Mean**	**Variance**	
RemoteSensing	0.034	0.0016	0.028	0.0017	3.52e-5

The energy anomaly isolation results show that from the 11% detected energy spikes, 4% are false positive. The false positives are related to instances where only DVS or TS-DVS services were running. In the rest of 96% energy spikes either TimeSync service, RemoteSensing service, or both Time-Sync and DVS were running.

We now discuss how the detected energy anomalies are used in a Bayesian belief revision system to achieve optimized service selection. Assume that the service pool includes the four discussed services: *RemoteSensing*, *TimeSync*, *DVS* and *TS-DVS*. We will therefore have $H = 16$ hypotheses, including the following:

- h_1: TimeSync has high energy consumption.
- h_2: DVS has high energy consumption.
- h_3: TS-DVS has high energy consumption.
- h_4: RemoteSensing has high energy consumption.
- h_5: Interaction of TimeSync and DVS has high energy consumption.
- h_6: Interaction of TimeSync and TS-DVS has high energy consumption.
- h_8: Interaction of RemoteSensing, TimeSync, and TS-DVS have high energy consumption.

In the beginning, $P(h_i) = P(h_j) = p$ for all $i, j \leq 16$. Let $p = 1/16$. We recalculate these probabilities as soon as the isolation step identifies services or service interactions with high energy consumption. From Tables 5.12 and 5.13 we can conclude that TimeSync, RemoteSensing and the interaction of TimeSync and DVS have high energy consumption. We recalculate $P(h_i)$ assuming a degree of uncertainty $\alpha = 0.95$. Using Equations 5.4 and 5.5, $P(h_1)$, $P(h_4)$, and $P(h_5)$ will change from 0.0625 to approximately 0.32. The probability $P(h_i)$ for $i \notin \{1, 4, 5\}$ changes from 0.0625 to approximately 0.004. The monitor shares the modified probabilities with the service selection module. An example of the effect of this analysis to service selection is that when TimeSync is selected and both DVS and TS-DVS satisfy the constraints, TS-DVS will be selected instead of DVS.

CHAPTER 6

SERVICE SHARING AND SELECTION

Much of the effort in wireless sensor network research aims at overcoming the challenges of software development to build applications that exploit the capabilities of these systems while satisfying the underlying constraints. We aim to accomplish this goal by exploiting reuse and adaptability provided by *service-oriented architecture* (SOA) [16] while at the same time selecting services that satisfy application requirements. SOA allows a wide variety of services (possibly of different implementation languages) to be selected and shared between different applications and modules [108].

The contributed code in the TinyOS-2.x repository is a good example to illustrate the importance of service sharing. TinyOS is an open source operating system that has been widely used in low-power wireless devices such as Imote2, MICAz, and TelosB. The TinyOS-2.x index of contributed code [109] includes 16 applications, 32 libraries, 5 system components, and 21 tools. It supports 8 different chips, 17 different platforms, and 5 sensor boards. It is highly desirable for a WSSN application developer to be able to exploit the developed components to build a new application.

This is a challenging problem since services for sensor networks face varying application requirements [110]. The diversity in the available components results in a dilemma for sensor network application developers. On the one hand, they are eager to exploit the opportunities offered by available services. On the other hand, the increasing heterogeneity and complexity of the available services for sensor networks jeopardizes their use.

Consider the selection of a multi-hop communication protocol, a time synchronization service, and an energy management scheme for a structural health monitoring application under strict energy, memory, synchronization error, and packet loss budgets. Consider a per-hour energy consumption budget B_E, memory budget B_{Mem}, time synchronization error budget B_{TS}, and packet loss budget B_{Pckt}. Also consider three sets of available services S_{MH}, S_{TS}, and S_E for multi-hop communication, time synchronization, and energy management respectively. S_{MH}, S_{TS}, and S_E sets include 35 (considering multihop communication services discussed in [111] and [112]), 9 (services chosen from schemes discussed in [113]), and 50 (using energy management services presented in [114]) services respectively and at least one service from each of these sets should be chosen. The selection of each service changes the available budgets non-deterministically, introducing non-linearity in service selection. For example, selection of service E_i that uses frequency scaling for energy management changes (increases) the available energy budget B_E by ϵ_i, and decreases the time synchronization error budget B_{TS} by δ_i, and memory budget B_M by μ_i. Another energy management service E_j that works based on periodically putting nodes to sleep would only affect the energy and memory budgets by ϵ_j and μ_j respectively. The selection of the multi-hop and time synchronization services can thus have varying constraints based on which energy management service has been chosen. The large number of available services and their varying requirements render the manual selection of services computationally intractable.

Currently, a suitable selection of any of the aforementioned services for a specific application needs knowledge on the details of each of these protocols. Efficient service selection from the large number of available services and their varying requirements and parameters can greatly facilitate the application development task. In this chapter, we first elaborate on the requirements of automatic service selection for application requirement satisfaction. We then describe S4, a system for service selection for large sensor network application development which allows automatic service selection while satisfying application constraints.

6.1 System Requirements

We support service sharing and reuse by allowing services to be used in the development of a multitude of applications. S4 enables sensor network application developers exploit a pool of existing services, while satisfying application requirements. For this purpose, the followings are required:

1. Specification of application requirements.
2. Specification of service properties.
3. A match between available services and application requirements.
4. Generation of a configuration file that links the selected services.

We aim to maintain separation of concerns and provide a suitable level of abstraction for application users, application developers and service developers. Application users will submit high level requirements of the application such as maximum allowed energy consumption, maximum packet drop, time synchronization error, etc. The high-level requirements vary widely from one application to the other. Specifying these requirements should not require detailed information about software toolsuite and the underlying hardware platform. Application developers can use S4 to take advantage of the pool of available services in a way that application requirements are satisfied. Service developers provide the implementation of services that constitute a pool of available services. They also provide an estimate of non-functional properties of services.

In addition to service specifications provided by service developers, S4 takes advantage of dynamic data provided by the monitoring system (Chapter 5). The monitor continuously profiles the deployed services and provides S4 with updated service specifications and dependency relationships.

6.2 Service Selection

We consider the service selection as a constraint satisfaction problem (CSP). A constraint satisfaction problem consists of a set of n variables, $\{x_1, ..., x_n\}$; a domain D_i of possible values for each variable $x_i, 1 \leq i \leq n$; and a collection of m constraints $\{c_1, ..., c_m\}$. Each constraint $c_i, 1 \leq i \leq m$, is a constraint over some set of variables called the *scheme* of the constraint. The size of this set is known as the *arity* of the constraint. A solution to a CSP is an assignment of a value $a_i \in D_i$ to $x_i, 1 \leq i \leq n$, that satisfies all the constraints [76].

We cast the service selection problem as a CSP in terms of variables, values and constraints. Each of the underlying services provides a set of specifications, defining the requirements it satisfies. These specifications will be used in the CSP as variable domains, and the constraints are generated from application requirements.

6.2.1 CSP Formulation

Let us consider the high-level requirements of the SHM application. An important constraint on many WSSN applications is the memory requirement and energy consumption of the system. SHM applications have additional constraints on maximum time synchronization error and maximum data packet loss [115].. To formulate the requirements as a CSP, we define the variables to be the employed policies: *MaxEnergy*, *MaxMem*, *MaxTSError*, and *MaxPcktLoss*. Variable domains are determined by values listed in service descriptions.

Let us consider a maximum energy consumption of 20K, maximum time synchronization error of 40, maximum packet loss of 3 and maximum memory requirement of 4K. These constraints are unary and limit the value of *MaxEnergy*, *MaxTSError*, *MaxPcktLoss*, and *MaxMem* to a number less than 20K, 40, 3, and 4K respectively:

$$\begin{aligned}
C_{MaxEnergy} = \quad & \{MaxEnergy \in D_{MaxEnergy} | MaxEnergy < 20K\} \\
C_{MaxTSError} = \quad & \{MaxTSError \in D_{MaxTSError} | MaxTSError < 40\} \\
C_{MaxPcktLoss} = \quad & \{MaxPcktLoss \in D_{MaxPcktLoss} | MaxPcktLoss < 3\} \\
C_{MaxMem} = \quad & \{MaxMem \in D_{MaxMem} | MaxMem < 4K\}
\end{aligned} \tag{6.1}$$

The constraint graph includes four nodes: *MaxEnergy*, *MaxTSError*, *MaxPcktLoss*, and *MaxMem* with no edges.

6.2.2 Service Selection Module

At this stage the CSP formulation is complete and the task is to find a set of services, from the available pool of services, that satisfy all application constraints. We will exploit *meta constraint satisfaction* [116] for the purpose of service selection. Meta constraint satisfaction problems were originally designed to deal with the complexity of solving a problem by solving an equivalent problem, represented at a different level of abstraction, which can be solved more efficiently. In our case, the nature of the problem calls for a meta constraint satisfaction problem. This is because an application requires different types of services, each of which can be provided by a multiple of implementations. For example, an application may require a time synchronization service, a multi-hop communication service, and an energy management service. We need to choose a service for each of these required services such that:

1. Each service satisfies application constraints that include variables presented in the service's specifications.
2. The interaction between all selected services conform with the detailed application requirements.

At the meta level, we can decompose the problem into subproblems, each for selecting one of these services. Each subproblem includes a subset of the variables in the original problem, together with the values for these variables and the constraints relating variables within this subset [80]. The subproblems are represented by *metavariables.* The domain of a metavariable is the set of solutions to the subproblem (i.e. a set of services satisfying the first item above). Metavariables can overlap by sharing common variables. These common variables define the interactions between services. A *metaconstraint* between two metavariables must enforce all the original constraints, involving variables from the corresponding subproblems. This ensures that the second item above is satisfied. Furthermore, if the same variable appears in both subproblems, the metaconstraint must ensure that this variable receives the same value in the solution chosen as *metavalue* for each of them.

We can think of each service as a meta-variable in the CSP, containing a subset of variables present in the constraints. The variables within each service, as well as variables from different services may be connected via the derived constraints and different services can share variables. Additional constraints can be used to relate variables in a single service.

The CSP solver may return more than one solution that satisfies application constraints. In the next section we will discuss how S4 chooses a solution from the available options.

6.2.3 Applying Dynamic Deployment Data

The Monitor provides on-line data regarding dynamic properties of services. S4 leverages this data in two ways. First, whenever available, dynamic service properties derived from the sensor network deployment replace static service properties determined by the service developer. The dynamic properties of each service define the CSP variables that correspond to that services and their values are used in the CSP solver.

We allow the CSP solver to return all solutions that satisfy application constraints. The second use of data from the Monitor is in choosing one solution from the available options. The Bayesian belief revision system of the Monitor is used for this purpose (Chapter 5). The Bayesian belief revision system provides a list of hypotheses that state the probability of having high energy consumption when a set of services are running. The probability of each hypotheses is updated as new information becomes available from the sensor network.

Let us consider a set $R = \{R_1, R_2, ..., R_N\}$ of N possible CSP solutions. For each solution R_i with services $\{s_1, s_2, ..., s_m\}$, we calculate the probability

$P_h(R_i)$ of having high energy consumption. $P_h(R_i)$ is calculated as follows.

$$P_h(R_i) = \sum_{forall \omega_j \in P(R_i)} P_h(\omega_j) \tag{6.2}$$

where $P(R_i) = \{\omega_1, \omega_2, ..., \omega_{2^m}\}$ is the power set of R_i. A solution with the minimum P_h is finally chosen as the service selection. This approach of choosing a service selection provides flexibility in the degree of sensitivity to dynamic data from the network.

6.2.4 Configuration File Generation

We generate the configuration file in nesC language for embedded applications that use TinyOS. TinyOS [117] is a component-based operating system and platform for wireless sensor networks, written in the nesC programming language [118] as a set of cooperating tasks and processes. nesC has a C-like syntax with support for the TinyOS concurrency model.

Building embedded applications in nesC involves linking software *components*. Each component *provides* and *uses* a number of interfaces. There are two types of components in nesC: *modules* and *configurations*. Modules provide application code, implementing one or more interface while configurations are used to assemble other components together, connecting interfaces used by components to interfaces provided by others. Our code generation module generates the nesC configuration file with the information given by the CSP solver output, and leaves the implementation of the module to the application developer. The interfaces that the application provides are given a priori with its requirements specification. The interfaces that are used are derived from the CSP solver. The information for the used and provided interfaces are given to the configuration generator module to provide the nesC configuration file. Figure 6.1 shows this process.

6.3 Implementation

We have implemented our system for a simplified SHM application to collect distributed sensor data from multiple sensors. This application requires *timestamped* data to be *reliably* collected from the sensor nodes. The high-level requirements for this application are the same as the SHM application, and Equation 6.1 shows the high-level CSP. The SHM application requires four services: *Sensing*, *Time Synchronization*, *Remote Invocation*, *Multi-hop Communication*, and *Energy Management*.

Current implementations of services for WSSNs do not provide their non-functional properties. In order to evaluate our system we have chosen 2 sensing, 10 time synchronization, 2 remote invocation, 42 multihop communication, and 30 energy management services from the previously developed schemes [112, 111, 119, 113, 114]. We have derived service specification for the considered services by evaluating their *memory consumption*, *time synchronization error*, *computation* and *packet loss* in a fixed amount of time, and for a fixed network size of 10 nodes. Due to the lack of detailed non-functional specifications for these services, only rough estimates for each of these parameters were derived. In the first step, each parameter was evaluated as *extra low* (XL), *low* (L), *medium* (M), *high* (H), and *extra high* (XH). Next, these estimates were translated to exact numbers based on the type of the parameter. For example, memory consumption is translated to Bytes of memory, time synchronization is translated to micro seconds of error, etc. Table 6.1 shows a subset of these values for the considered services.

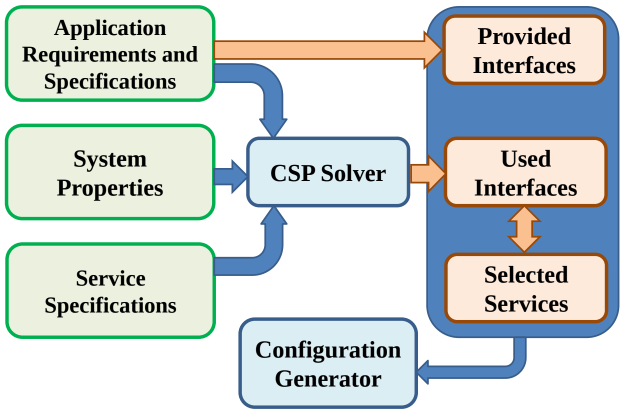


Figure 6.1: The process of generating a configuration file from application specifications.

The pool of services, their interfaces, and specifications are given to the CSP solver module. We have five metavariables: *Sensing*, *TimeSynchronization*, *RemoteInvocation*, *MHCommunication*, and *EnergyManagement* in this meta constraint satisfaction problem. The variables in this CSP are *memory*, *tsError*, *energy*, and *pcktLoss* with the following ranges.

$$\begin{aligned} memory &\in \{250, 300, 350, 400, 450\} \\ tsError &\in \{10, 20, 50, 100, 200\} \\ energy &\in \{100, 120, 150, 200, 300\} \\ pcktLoss &\in \{1, 5, 10, 20, 50\} \end{aligned} \tag{6.3}$$

Note that a service description may not contain all variables. In such cases, the variable takes a *don't-care* value in the CSP. Moreover, for some variables the cumulative value represented in each of the services should satisfy the constraints. This is different from traditional CSPs where the problem is to find a single value, from the provided range, that satisfies the constraints. For example, consider the *runTime* variable. In order to satisfy the maximum energy consumption policy (*em-PolicyMaxEnergy*), the sum of all

Table 6.1: Estimated non-functional properties for a subset of services considered in the service selection.

Service Name	**Memory**		**Time Sync Error**		**Energy**		**Pckt Loss**	
	Est.	Value	Est.	Value	Est.	Value	Est.	Value
Sensing	XH	450	-	-	M	150	-	-
SensingUnit	H	400	-	-	M	150	-	-
RemoteSensing	H	400	-	-	L	120	-	-
FTSP	L	300	XL	10	L	120	-	-
Gradient Clock	M	350	H	100	M	150	-	-
RemoteInvoc	XL	350	-	-	L	120	M	10
ReliableComm	M	250	-	-	L	120	XL	1
ReliableInvoc	M	250	-	-	L	120	L	3
AODV	XL	250	-	-	XL	100	M	10
Agile AODV	XL	250	-	-	L	120	XL	1
Rumor	XL	250	-	-	H	200	H	20
OLSR	L	300	-	-	XH	300	L	5
DVS	L	300	-	-	XL	100	-	-
TS-DVS	L	300	-	-	H	200	-	-

values of *runTime* variable in the five metavariables should be considered. Similarly, in order to satisfy the maximum time synchronization error policy (*em-PolicyMaxTSError*) the maximum value of *tsError* variable should be considered. Thus, in order to take into account the cumulative values of variables, the constraints are accompanied by a suitable operator from the set $\{min, max, \sum, \prod\}$.

For solving the CSP we use the Constraint Class from Microsoft's *Solver-Foundation* library. The variables, their domains and the constraints are given to the *ConstraintSystem* which can provide the solution using its *Solve* method. The output from the solver module (i.e. the selected services) are given to the configuration generator.

Table 6.2 shows results from the CSP solver. The CSP solver can return more than one service selection that satisfy the constraints. In this case

Table 6.2: Results of service selection for an SHM application using CSP solver.

Service	**Selection**		
	R_1	R_2	R_3
Sensing	RemoteSensing	RemoteSensing	RemoteSensing
Time Sync.	FTSP	FTSP	FTSP
Remote Invocation	ReliableInvoc	ReliableComm	ReliableComm
Multihop Comm.	Agile AODV	Agile AODV	AODV
Energy Mng.	TS-DVS	DVS	TS-DVS

information from the Bayesian belief revision module of the Monitor is used to break the tie. In the beginning, $P_h(R_i) = P_h(R_j)$, for all $i, j \in \{1..3\}$ and any of the solutions can be selected. The service selection can change in response to information from the Monitor on the probability of having high energy consumption for each of the solutions. From results in Chapter 5 we know that the Monitor has derived the following probabilities after receiving data from the sensor network:

- $P_h(RemoteSensing) = 0.32$
- $P_h(FTSP) = 0.32$
- $P_h(FTSPandDVS) = 0.32$
- $P_h(DVS) = 0.004$
- $P_h(TS - DVS) = 0.004$
- $P_h(RemoteSensingandTS - DVS) = 0.004$
- $P_h(RemoteSensingandDVS) = 0.004$
- $P_h(RemoteSensingandFTSP) = 0.004$

```
configuration RemoteSensingC
{

 provides {
        interface RemoteSensing;
        interface RetrieveData;
        }
}

implementation
{
 components Main,
        RemoteSensingM,
        SensingC,
        TSDVSC,
        FTSPC,
        AgileAODVC,
        ReliableInvocC;

 Main.StdControl -> RemoteSensingM;

 RemoteSensingM.Sensing -> SensingC;
 RemoteSensingM.FTSP -> FTSPC;
 RemoteSensingM.TSDVS -> TSDVSC;
 RemoteSensingM.AgileAODV -> AgileAODVC;
 RemoteSensingM.ReliableInvoc -> ReliableInvocC;
}
```

Listing 6.1: Configuration file for a simplified SHM application. The configuration file is generated in nesC.

In calculating $P_h(R_i)$, terms for which the Monitor does not have data are ignored. We thus have: $P_h(R_2) > P_h(R_1)$ and $P_h(R_1) = P_h(R_3)$. This results in selecting either R_1 or R_3 for the SHM application under consideration.

Listing 6.1 shows the generated configuration file. The arrows bind interfaces (on the left) to implementation (on the right). This configuration file shows that *Sensing*, *FTSP*, *TSDVS*, *Agile AODV* and *Reliable Invoc* services are chosen and linked to build the application.

6.4 Experimental Results

In order to assess the impact of dynamic policy-based service adaptation and global policy changes, we consider the energy consumption of a WSN system with rechargeable batteries supplied by solar panels. For this purpose we use data gathered from a long-term continuous monitoring deployment of over 100 Imote2 sensor nodes on the Jindo Bridge in South Korea.

We have used the charging current and voltage data collected in the course of 28 consecutive days for modeling the energy supply and consumption in our system.

We first assess the effect of dynamic policy changes at the node level. Each node is able to adapt service parameters within the range dictated by the global policy. We then discuss the effect of global policy changes such as service selection change in response to dynamic information from the Monitor.

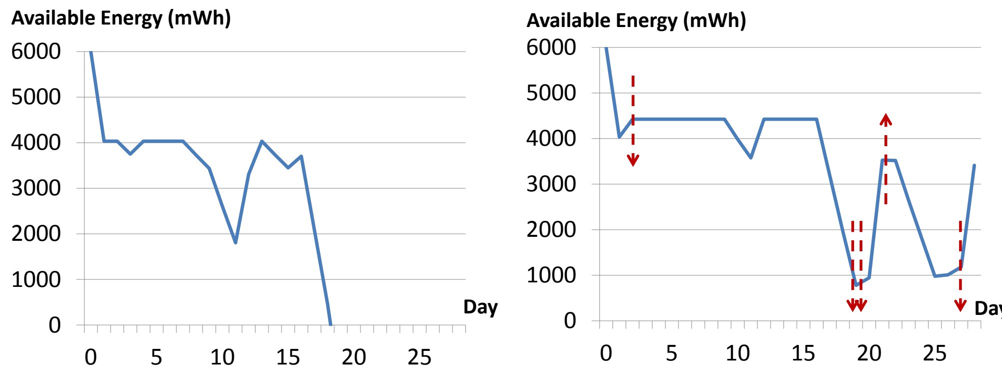
$$\begin{aligned}&\textit{Estimated power charging per day} = \\ &\textit{Daily charging current} \times 4.1V \times \textit{Estimated Hours of Sunlight}\end{aligned} \tag{6.4}$$

6.4.1 Local Policy-based Adaptation

First, we compare the energy level on two nodes, one employing a static monitoring scheme and the other employing our policy-driven adaptation method. The nodes are programmed with a data acquisition application with 4 phases: *wakeup*, *sensing*, *data processing*, *data transfer*, and are in deep sleep mode when inactive. Energy consumption on the nodes is calculated based on current draw measurement and duration of each phase, as depicted

Table 6.3: Current draw measurements and duration of phases for a data acquisition application. Nominal input voltage for the Imote2 is 4.5 V.

Phase	**Current(mA)**	**Power(mW)**	**Duration(s)**	**Energy Consumption (mWh)**
Wake-up	48	216	30	1.8
Sensing	169	760.5	900	190.125
Data Processing	80	360	10	1
Data Transfer	55	247.5	40	2.75
Sleep	0.1	0.45		



(a) Static (b) Policy-driven Adaptation

Figure 6.2: Comparison of energy saving using a static method and dynamic, policy-driven adaptation. Arrows indicate adaptation actions.

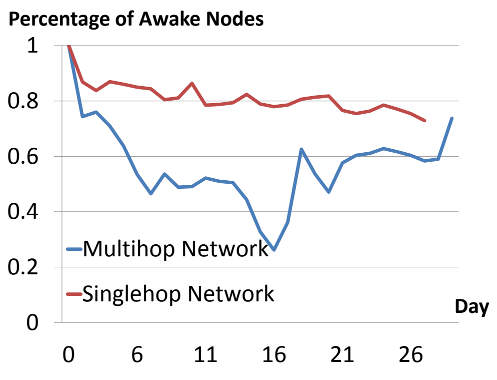
in Table 6.3. On each node, the available energy is derived based on the energy consumption of running the data acquisition application, and the provided daily solar charging power as described above. The static scheme runs the application 10 times per day. The policy-based adaptation model adjust the number of remote sensing events based on the average of available daily charging current in the last five days. The range of allowed sensing events per day is determined by the system-wide policy and in this case is between 1 and 10. Figure 6.2 compares these two approaches. In the static scheme, the node runs out of available energy after only 18 days of operation, while the policy-based scheme maintains an acceptable level of available energy at all times by adjusting the number of samplings per day. The arrows in Figure 6.2b show adjustments in the number of sensing events in a day based on the average charging current in the last 5 days. Arrows pointing down show a decrease in the number of sensing events, while upward arrows show an increase.

6.4.2 Global Policy Changes

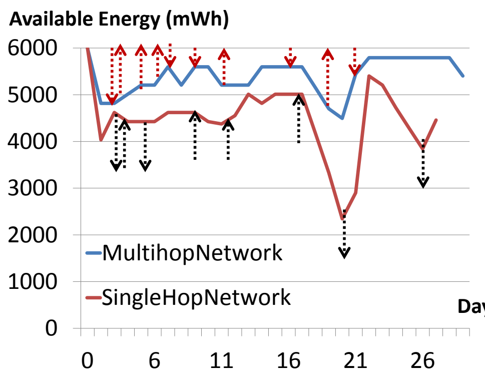
We consider the effect of dynamic global policy changes at two levels. First, we evaluate the effect of system-wide policy changes that alter the acceptable range of parameters across the network. Next, we evaluate a more aggressive global policy change which triggers a change in service selection.

In order to evaluate the effect of system-wide policy changes to the acceptable range of parameters, we consider two separate networks: one single-hop and the other multi-hop with up to a 5-hop radius. The multi-hop network consumes more energy, which results in several nodes running out of power and becoming unresponsive. An increase in the number of unresponsive nodes complicates routing and can even cause network fragmentation. Thus, in the adaptive scheme, the system-wide policy is changed based on the percentage of responsive nodes. The policy change mechanism alters the range of acceptable sensing events in a day based on the reported number of responsive nodes. As the number of responsive nodes decreases, the adopted policies become more conservative to ensure long network lifetime. At the node level, each node uses this range and the available charging current to choose the number of sensing events in a day.

Figure 6.3a shows the percentage of responsive nodes in the two networks. The multi-hop network generally has a smaller number of responsive nodes



(a) Fraction of awake nodes.



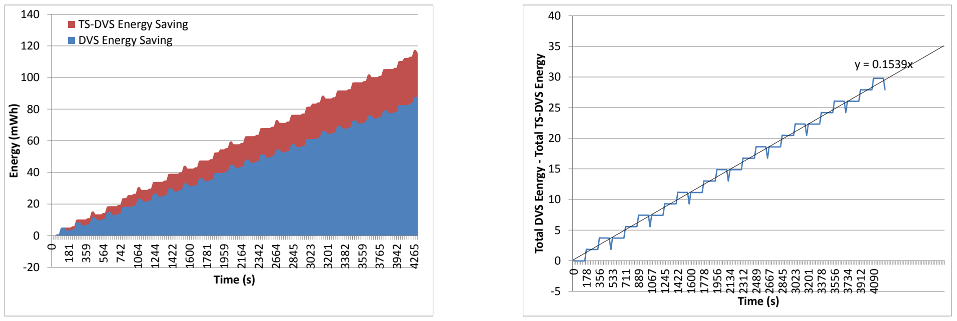
(b) Effect of policy changes.

Figure 6.3: Effect of global policy changes in multi-hop and single-hop networks. Arrows indicate adaptation actions.

and shows a larger variance. Figure 6.3b shows the effect of system-wide policy adjustments on the available level of energy. The multi-hop network has a larger number of policy changes, which is due to the large variance in the number of responsive nodes. It also adopts a more conservative policy to maintain network connectivity at all times.

To evaluate the effect of the adaptive middleware framework on service selection, we consider the SHM application with an initial service selection corresponding to R_2 from the CSP solver. As discussed in the previous section, the information from the Bayesian belief revision module of the Monitor leads to the conclusion that this service selection has a higher probability of having high energy consumption compared to a selection that replaces DVS service with TS-DVS. This is due to a time-keeping anomaly of embedded devices that use the CPU tick counter as their clock source [96]. In these embedded devices, the CPU frequency switching actions of the DVFS service disrupt the local clock, causing a time lapse.

The time synchronization error problem under classical DVFS can be mitigated by running the time synchronization service more frequently. However, the time synchronization service itself has relatively high energy consump-



(a) Energy consumption (b) Energy difference

Figure 6.4: Comparison of energy consumption between a classical DVS service and frequent resynchronization with a modified TS-DVS service. The TS-DVS service takes resynchronization cost into account.

tion, causing an overall increase in the energy consumption due to more frequent resynchronizations. This excessive energy consumption is detected by the adaptive middlware framework, triggering a new service selection.

Figure 6.4 compares the energy consumption of the classical DVS scheme with that of the modified service selection. In the calculation of total energy consumption for the new service selection we have considered the cost of service reconfiguration. Table 6.4 shows the reconfiguration cost calculation.

Table 6.4: Reconfiguration cost for new service selection.

Cost Item	**Time (s)**	**Energy (mWh)**
Transmission	0.03	0.0019
Processing	0.02	0.0013
Total	0.05	0.0031

These results confirm that even a relatively simple rule-based adaptive policy can greatly benefit a resource-limited system, increasing its longevity and robustness. While we focus on energy in this paper, similar effects can be seen in network congestion, load balancing, and other facets of middleware services.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

In this section we provide a summary of this dissertation. We will also discuss its limitations and future directions.

7.1 Summary

In this dissertation we studied the problem of software development for large-scale wireless smart sensor networks (WSSNs). Energy management is important to increase the lifespan of sensor nodes. We therefore focus on optimizing energy consumption of the sensor network in the face of varying application requirements and environmental conditions.

Our work is motivated by the requirements of two real-world WSSN applications, namely structural health monitoring (SHM) and environmental and agricultural monitoring. It was during the course of developing and deploying middleware services for these applications that we recognized the need for a framework to facilitate software development for dynamic and complex WSSN applications. We follow three main principles for this purpose: *separation of concerns*, *sharing and reuse*, and *adaptivity*. The result is I-AdMiN: Illinois Adaptive Middleware framework for wireless smart sensor Networks.

In the design of I-AdMiN, we consider applications that can be fully represented as a composition of services. In this model, a set of services can be linked together to build an application with services interacting with one another through their well-defined interfaces. We use the Actor model of computation to represent service instances and their interactions. A service interface is thus the set of messages the actor representing the services sends and receives. Service instances connect services to each other and to the application. For service interface representation, we define *service properties* which describe non-functional service specifications. Service properties can be either static or dynamic. Static service properties are determined by service developer at the time of its implementation and therefore do not reflect the effect of the environment. Environmental conditions can greatly affect service properties such as energy consumption which in turn affects the efficiency and lifetime of the sensor network. Dynamic service properties reflect the effect of the environment and are determined during the runtime of the sensor network.

The architecture of I-AdMiN has two layers. The base layer is comprised of service actors which together constitute the main functionality of the WSSN application. The meta layer is the main focus of this dissertation and is comprised of different meta-actors that enable dynamic service configuration. The *Monitor* meta-actor is responsible for deriving energy characteristics of services as energy anomalies. It does so by leveraging coarse-grained periodic data from the sensor nodes.

The monitor uses weighted least squares (WLS) regression to attribute aggregate energy consumption to individual services that are running in the sensor network. Another responsibility of the Monitor is to detect energy spikes and isolating the cause. The cause of an energy anomaly can be service or service interaction issues, or hardware problems. When new data from the sensor network becomes available to the Monitor, it first detects and diagnoses instances of high energy consumption. This information is then used in a Bayesian belief revision system where each hypothesis states the probability of having high energy consumption when a set of services are running. The probability of the hypotheses which is inferred using Bayesian methods is shared with the *Global Coordinator* meta-actor to be used at the time of service selection. The Global Coordinator meta-actor is responsible for automatic service selection based on dynamic service properties and the detected energy anomalies in a way that application requirements are satisfied. We represent service selection as a meta-constraint satisfaction problem and choose a solution based on dynamic network information from the Monitor. Updates on service selection and parameterization are sent to the *Parameter Adaptation* meta-actor to be distributed to *Service* meta-actors on individual nodes. Service meta-actors are tightly coupled to the base level service actors and are responsible for enacting low-latency, localized control over service parameters.

I-AdMiN facilitates software development for WSSNs by allowing efficient and automatic service selection. It also finds dynamic properties of services which cannot be accurately determined off-line and can change over time. The actor model in turn allows dynamic reconfiguration and parameterization of the selected services based on application requirements as well as environmental conditions. The approaches taken in I-AdMiN are vastly applicable to other areas of distributed systems. For example, this framework can be used in the HomeOS [120] project to allow dynamic selection of different modules of a home deployment. As another example, many mobile distributed systems can use I-AdMiN for the monitoring of their components from coarse-grained data, which can then optimize component selection and operation.

7.2 Limitations and Future Work

WSSN applications impose unique requirements that are crucial to their successful deployment. We therefore focus on satisfying application constraints in the design of I-AdMiN. Our service selection module, S4 satisfies all application constraints submitted by application user. However, in deriving dynamic service properties and constraint violations our focus is only on energy. We devised a method to attribute aggregate energy consumption to individual services. This can be extended to other service properties such as runtime, packet loss, etc. We also only discussed the detection and isolation of energy anomalies while other constraint violations can be investigated during system runtime.

The adaptive middleware framework employs a policy-based adaptation where each service instance is assigned a policy, which gives it a range for one or more of its configuration parameters. The service is then free to make independent decisions about selecting the appropriate configuration value within that range, and can adjust it at any point due to local adaptation decisions. Our work does not specify how parameter ranges are derived and how a service can choose a parameter that improves the efficiency of the sensor network. An area which can be investigated in future work is parameter derivation based on dynamic network properties. In the beginning, a set of parameters can be assigned to each service in a way that application constraints are satisfied. A monitoring scheme can be used to attributed total energy consumption to different service parameter values and a selection that optimizes energy consumption can be picked.

Another area for future work is optimizing when service selection and parameterization updates are distributed in the network. Service updates can be distributed either periodically or in the event of specific occurrences. We followed an event-driven approach where service configuration updates are distributed as soon as new dynamic information becomes available. We leave a thorough investigation of this optimization for future work.

In this work we employed an implicit enforcement of the assigned policies to services by changing service selection when an energy anomaly is detected. Another module that can be added to our framework is a regulator that imposes the assigned policies and ensures that service properties reflect their true behavior during network runtime. Such a system can also enforce failure semantics to improve fault tolerance. Ideally, each component has its own set of failure semantics for additional flexibility. Such behavior can be realized by the use of a language such as DIL [121], which allows per-component protocol specifications to transparently enforce failure semantics.

REFERENCES

[1] J. Li, T. Nagayama, K. Mechitov, and B. F. Spencer, “Efficient campaign-type structural health monitoring using wireless smart sensors,” in *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems, vol. 8345*, 2012.

[2] “Illinois structural health monitoring project (ishmp) service toolsuite.” [Online]. Available: http://shm.cs.uiuc.edu/

[3] Y. Yao and J. Gehrke, “The cougar approach to in-network query processing in sensor networks,” *SIGMOD Rec.*, vol. 31, no. 3, pp. 9–18, Sep. 2002. [Online]. Available: http://doi.acm.org/10.1145/601858.601861

[4] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “Tinydb: an acquisitional query processing system for sensor networks,” *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, Mar. 2005. [Online]. Available: http://doi.acm.org/10.1145/1061318.1061322

[5] R. Gummadi, O. Gnawali, and R. Govindan, “Macro-programming wireless sensor networks using kairos,” in *Distributed Computing in Sensor Systems*, ser. Lecture Notes in Computer Science, V. Prasanna, S. Iyengar, P. Spirakis, and M. Welsh, Eds. Springer Berlin / Heidelberg, 2005, vol. 3560, pp. 466–466.

[6] N. Kothari, R. Gummadi, T. Millstein, and R. Govindan, “Reliable and efficient programming abstractions for wireless sensor networks,” in *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, ser. PLDI ’07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1250734.1250757 pp. 200–210.

[7] T. W. Hnat, T. I. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse, "Macrolab: a vector-based macroprogramming framework for cyber-physical systems," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, ser. SenSys '08. New York, NY, USA: ACM, 2008. [Online]. Available: http://doi.acm.org/10.1145/1460412.1460435 pp. 225–238.

[8] L. Luo, T. F. Abdelzaher, T. He, and J. A. Stankovic, "EnviroSuite: an environmentally immersive programming framework for sensor networks," *Transactions on Embedded Computer Systems*, vol. 5, pp. 543–576, August 2006. [Online]. Available: http://doi.acm.org/10.1145/1165780.1165782

[9] A. Pathak, L. Mottola, A. Bakshi, V. K. Prasanna, and G. P. Picco, "Expressing sensor network interaction patterns using data-driven macroprogramming," in *Proceedings of the 5th IEEE International Conference on Pervasive Computing and Communications Workshops*, ser. PERCOMW '07. Washington, DC, USA: IEEE Computer Society, 2007. [Online]. Available: http://dx.doi.org/10.1109/PERCOMW.2007.46 pp. 255–260.

[10] R. Newton, G. Morrisett, and M. Welsh, "The Regiment macroprogramming system," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, ser. IPSN '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1236360.1236422 pp. 489–498.

[11] M. Hossain, A. Alim Al Islam, M. Kulkarni, and V. Raghunathan, "μsetl: A set based programming abstraction for wireless sensor networks," in *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, april 2011, pp. 354 –365.

[12] K. Mechitov and G. Agha, "An architecture for dynamic service-oriented computing in networked embedded systems," in *Software Service and Application Engineering*, M. Heisel, Ed., 2012, pp. 147–164.

[13] J. Liu and F. Zhao, "Towards semantic services for sensor-rich information systems," in *Broadband Networks, 2005. BroadNets 2005. 2nd International Conference on*, oct. 2005, pp. 967 –974 Vol. 2.

[14] K. Mechitov, R. Razavi, and G. Agha, "Architecture design principles to support adaptive service orchestration in wsn applications," *SIGBED Rev.*, vol. 4, no. 3, pp. 37–42, July 2007. [Online]. Available: http://doi.acm.org/10.1145/1317103.1317110

[15] M. P. Papazoglou and W.-J. Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The VLDB Journal*, vol. 16, no. 3, pp. 389–415, 2007. [Online]. Available: http://dx.doi.org/10.1007/s00778-007-0044-3

[16] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan, "An efficient service oriented architecture for heterogeneous and dynamic wireless sensor networks," in *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, oct. 2008, pp. 740 –747.

[17] G. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[18] M. Kushwaha, I. Amundson, X. Koutsoukos, S. Neema, and J. Sztipanovits, "Oasis: A programming framework for service-oriented sensor networks," in *Communication Systems Software and Middleware, 2007. COMSWARE 2007. 2nd International Conference on*, jan. 2007, pp. 1 –8.

[19] W. Emmerich, "Software engineering and middleware: a roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000. [Online]. Available: http://doi.acm.org/10.1145/336512.336542 pp. 117–129.

[20] C. L. Hall, *Building client/server applications using TUXEDO.* New York, NY, USA: John Wiley & Sons, Inc., 1996.

[21] E. S. Hudders, *CICS: a guide to internal structure.* Somerset, NJ, USA: Wiley-QED Publishing, 1994.

[22] V. Issarny, M. Caporuscio, and N. Georgantas, "A perspective on the future of middleware-based software engineering," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.2 pp. 244–258.

[23] M. Hapner, R. Burridge, and R. Sharma, "Java message service specification," Sun Microsystems, Tech. Rep., 1999.

[24] Tibco Rendezvous, http://www.tibco.com/products/soa/messaging/rendezvous/.

[25] Progress SonicMQ, http://www.progress.com/en/sonic/sonicmq.html.

[26] R. Orfali, D. Harkey, and J. Edwards, *Instant CORBA.* New York, NY, USA: John Wiley & Sons, Inc., 1997.

[27] D. Box, *Essential COM*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[28] B. Burke and R. Monson-Haefel, *Enterprise JavaBeans 3.0 (5th Edition)*. O'Reilly Media, Inc., 2006.

[29] M. Singh and M. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley and Sons, 2005.

[30] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Krämer, "Service-oriented computing: A research roadmap," in *Service Oriented Computing (SOC)*, F. Cubera, B. J. Krämer, and M. P. Papazoglou, Eds., no. 05462, 2006.

[31] W. Tsai, "Service-oriented system engineering: A new paradigm," in *IEEE International Workshop on Service-Oriented Systems Engineering*, 2005, pp. 3–8.

[32] L. Jingyong, Z. Yong, C. Yong, and Z. Lichen, "Middleware-based distributed systems software process," in *Proceedings of the 2009 International Conference on Hybrid Information Technology*, ser. ICHIT '09. New York, NY, USA: ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1644993.1645058 pp. 345–348.

[33] M.-M. Wang, J.-N. Cao, J. Li, and S. Dasi, "Middleware for wireless sensor networks: A survey," *Journal of Computer Science and Technology*, vol. 23, pp. 305–326, 2008.

[34] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031506 pp. 81–94.

[35] W. Dong, C. Chen, X. Liu, J. Bu, and Y. Liu, "Dynamic linking and loading in networked embedded systems," in *Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on*, 2009, pp. 554–562.

[36] P. J. Marrón, M. Gauger, A. Lachenmann, D. Minder, O. Saukh, and K. Rothermel, "Flexcup: a flexible and efficient code update mechanism for sensor networks," in *Proceedings of the Third European conference on Wireless Sensor Networks*, ser. EWSN'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 212–227.

[37] 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 Trans. Comput. Syst.*, vol. 26, no. 1, pp. 1:1–1:42, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1328671.1328672

[38] D. Hughes, P. Greenwood, G. Blair, G. Coulson, P. Grace, F. Pappenberger, P. Smith, and K. Beven, “An experiment with reflective middleware to support grid-based flood monitoring,” *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 11, pp. 1303–1316, Aug. 2008. [Online]. Available: http://dx.doi.org/10.1002/cpe.v20:11

[39] L. Mottola, G. P. Picco, and A. A. Sheikh, “Figaro: fine-grained software reconfiguration for wireless sensor networks,” in *Proceedings of the 5th European conference on Wireless sensor networks*, ser. EWSN'08. Berlin, Heidelberg: Springer-Verlag, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1786014.1786039 pp. 286–304.

[40] D. Hughes, K. Thoelen, W. Horré, N. Matthys, J. D. Cid, S. Michiels, C. Huygens, and W. Joosen, “Looci: a loosely-coupled component infrastructure for networked embedded systems,” in *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*, ser. MoMM '09. New York, NY, USA: ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1821748.1821787 pp. 195–203.

[41] A. Taherkordi, F. Eliassen, R. Rouvoy, and Q. Le-Trung, “Rewise: A new component model for lightweight software reconfiguration in wireless sensor networks,” in *Proceedings of the OTM Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems: 2008 Workshops: ADI, AWeSoMe, COMBEK, EI2N, IWSSA, MONET, OnToContent + QSI, ORM, PerSys, RDDS, SEMELS, and SWWS*, ser. OTM '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 415–425.

[42] T. Liu and M. Martonosi, “Impala: a middleware system for managing autonomic, parallel sensor systems,” in *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, ser. PPoPP '03. New York, NY, USA: ACM, 2003. [Online]. Available: http://doi.acm.org/10.1145/781498.781516 pp. 107–118.

[43] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G. Picco, T. Sivaharan, N. Weerasinghe, and S. Zachariadis, “The runes middleware for networked embedded systems and its application in a disaster management scenario,” in *Pervasive Computing and Communications, 2007. PerCom '07. Fifth Annual IEEE International Conference on*, 2007, pp. 69–78.

[44] W. Horré, S. Michiels, W. Joosen, and P. Verbaeten, "Davim: Adaptable middleware for sensor networks," *IEEE Distributed Systems Online*, vol. 9, no. 1, pp. 1–, Jan. 2007. [Online]. Available: http://dx.doi.org/10.1109/MDSO.2008.2

[45] A. Taherkordi, Q. Le-Trung, R. Rouvoy, and F. Eliassen, "Wisekit: A distributed middleware to support application-level adaptation in sensor networks," in *Proceedings of the 9th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, ser. DAIS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 44–58.

[46] A. Taherkordi, F. Loiret, R. Rouvoy, and F. Eliassen, "Optimizing Sensor Network Reprogramming via In-situ Reconfigurable Components," *ACM Transactions on Sensor Networks*, vol. 9, no. 2, pp. 1–37, May 2013. [Online]. Available: http://hal.inria.fr/hal-00658748

[47] A. Taherkordi, F. Loiret, A. Abdolrazaghi, R. Rouvoy, Q. Le-Trung, and F. Eliassen, "Programming sensor networks using remora component model," in *Proceedings of the 6th IEEE international conference on Distributed Computing in Sensor Systems*, ser. DCOSS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 45–62.

[48] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," *SIGPLAN Notices*, vol. 37, pp. 85–95, October 2002. [Online]. Available: http://doi.acm.org/10.1145/605432.605407

[49] C.-L. Fok, G.-C. Roman, and C. Lu, "Mobile agent middleware for sensor networks: an application case study," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, ser. IPSN '05. Piscataway, NJ, USA: IEEE Press, 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1147685.1147747

[50] N. Carriero and D. Gelernter, "Linda in context," *Communications of the ACM*, vol. 32, pp. 444–458, April 1989. [Online]. Available: http://doi.acm.org/10.1145/63334.63337

[51] A. Boulis, C.-C. Han, and M. B. Srivastava, "Design and implementation of a framework for efficient and programmable sensor networks," in *International Conference on Mobile Systems, Applications, and Services.* USENIX Association.

[52] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting concurrent applications in wireless sensor networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1182807.1182822 pp. 139–152.

[53] Y. Kwon, S. Sundresh, K. Mechitov, and G. Agha, "ActorNet: an actor platform for wireless sensor networks," in *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1160633.1160871 pp. 1297–1300.

[54] G. Agha, N. Jamali, and C. A. Varela, "Agent naming and coordination: Actor based models and infrastructures," in *Coordination of Internet Agents: Models, Technologies, and Applications*, 2001, pp. 225–246.

[55] N. Jamali, P. Thati, and G. Agha, "An actor-based architecture for customizing and controlling agent ensembles," in *IEEE Intelligent Systems, vol. 14, no. 2, April*, 1999.

[56] A. Ricci, R. H. Bordini, and G. A. Agha, "Agere! (actors and agents reloaded): splash 2011 workshop on programming systems, languages and applications based on actors, agents and decentralized control," in *OOPSLA Companion*, C. V. Lopes and K. Fisher, Eds., 2011, pp. 325–326.

[57] A. Ricci, G. Agha, and R. H. Bordini, "Agere! (actors and agents reloaded): splash 2011 workshop on programming systems, languages and applications based on actors, agents and decentralized control," in *SPLASH Workshops*, C. V. Lopes, Ed., 2011, pp. 143–146.

[58] J. Rao and X. Su, "A survey of automated web service composition methods," in *Semantic Web Services and Web Process Composition*, ser. Lecture Notes in Computer Science, J. Cardoso and A. Sheth, Eds. Springer Berlin / Heidelberg, 2005, vol. 3387, pp. 43–54.

[59] T. Bellwood, L. Clement, D. Ehnebuske, A. Hately, M. Hondo, Y. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. Riegen, "Universal description discovery and integration (uddi) specification," Tech. Rep., 2005. [Online]. Available: http://www.oasis-open.org

[60] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (wsdl)," *W3C Web Site*, vol. 2008, no. 2008-01-07, pp. 1–32, 2001. [Online]. Available: http://www.w3.org/TR/wsdl

[61] D. e. a. Box, "Simple object access protocol (soap)," Tech. Rep., 2001. [Online]. Available: http://www.w3.org/TR/soap/

[62] F. Casati, S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, and M.-C. Shan, "eflow: a platform for developing and managing composite e-services," in *Research Challenges, 2000. Proceedings. Academia/Industry Working Conference on*, 2000, pp. 341 –348.

[63] S. McIlraith, T. Son, and H. Zeng, "Semantic web services," *Intelligent Systems, IEEE*, vol. 16, no. 2, pp. 46 – 53, mar-apr 2001.

[64] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing web services on the semantic web," *The VLDB Journal*, vol. 12, no. 4, pp. 333–351, Nov. 2003. [Online]. Available: http://dx.doi.org/10.1007/s00778-003-0101-5

[65] S. Narayanan and S. A. McIlraith, "Simulation, verification and automated composition of web services," in *Proceedings of the 11th international conference on World Wide Web*, ser. WWW '02. New York, NY, USA: ACM, 2002. [Online]. Available: http://doi.acm.org/10.1145/511446.511457 pp. 77–88.

[66] S. R. Ponnekanti and A. Fox, "Sword: A developer toolkit for web service composition," in *Proceedings of the 11th International WWW Conference (WWW2002)*, Honolulu, HI, USA, 2002.

[67] D. D. Lamanna, J. Skene, and W. Emmerich, "Slang: A language for defining service level agreements," in *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, ser. FTDCS '03. Washington, DC, USA: IEEE Computer Society, 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=795675.797134 pp. 100–.

[68] X. Gu, K. Nahrstedt, R. Chang, and C. Ward, "Qos-assured service composition in managed service overlay networks," in *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, may 2003, pp. 194 – 201.

[69] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 311 – 327, may 2004.

[70] N. Channa, S. Li, A. W. Shaikh, and X. Fu, "Constraint satisfaction in dynamic web service composition," in *Database and Expert Systems Applications, 2005. Proceedings. Sixteenth International Workshop on*, aug. 2005, pp. 658 – 664.

[71] A. Lazovik, M. Aiello, and R. Gennari, "Encoding requests to web service compositions as constraints," in *Principles and Practice of Constraint Programming - CP 2005*, ser. Lecture Notes in Computer Science, P. van Beek, Ed. Springer Berlin / Heidelberg, 2005, vol. 3709, pp. 782–786.

[72] E. Monfroy, O. Perrin, and C. Ringeissen, "Dynamic web services provisioning with constraints," in *On the Move to Meaningful Internet Systems: OTM 2008*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds. Springer Berlin / Heidelberg, 2008, vol. 5331, pp. 26–43.

[73] A. Ben Hassine, S. Matsubara, and T. Ishida, "A constraint-based approach to horizontal web service composition," in *The Semantic Web - ISWC 2006*, ser. Lecture Notes in Computer Science, I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, Eds. Springer Berlin / Heidelberg, 2006, vol. 4273, pp. 130–143.

[74] R. Thiagarajan and M. Stumptner, "Service composition with consistency-based matchmaking: A csp-based approach," in *Web Services, 2007. ECOWS '07. Fifth European Conference on*, nov. 2007, pp. 23 –32.

[75] E. Karakoc and P. Senkul, "Composing semantic web services under constraints," *Expert Systems with Applications*, vol. 36, no. 8, pp. 11 021 – 11 029, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417409002462

[76] V. Kumar, "Algorithms for constraint satisfaction problems: A survey," *AI MAGAZINE*, vol. 13, no. 1, pp. 32–44, 1992.

[77] V. Dhar and N. Ranganathan, "Integer programming vs. expert systems: An experimental comparison," *Commun. ACM*, vol. 33, no. 3, pp. 323–336, Mar. 1990. [Online]. Available: http://doi.acm.org/10.1145/77481.77485

[78]

[79] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara, "The distributed constraint satisfaction problem: formalization and algorithms," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 10, no. 5, pp. 673 –685, sep/oct 1998.

[80] D. Sabin and E. C. Freuder, "Configuration as composite constraint satisfaction," in *in Proc. Artificial Intelligence and Manufacturing. Research Planning Workshop.* AAAI Press, 1996, pp. 153–161.

[81] S. Mittal and B. Falkenhainer, "Dynamic constraint satisfaction problems," in *Proceedings of the eighth National conference on Artificial intelligence - Volume 1*, ser. AAAI'90. AAAI Press, 1990. [Online]. Available: http://dl.acm.org/citation.cfm?id=1865499.1865503 pp. 25–32.

[82] C.-C. Han, R. K. Rengaswamy, R. Shea, E. Kohler, and M. Srivastava, "Sos: A dynamic operating system for sensor networks," in *Third International Conference on Mobile Systems, Applications, And Services (Mobisys)*, 2005, pp. 163–176.

[83] S. Ren, G. A. Agha, and M. Saito, "A modular approach to programming distributed real-time systems," *Journal of Parallel and Distributed Computing*, vol. 36, no. 1, pp. 4 – 12, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731596900866

[84] B. Nielsen, S. Ren, and G. Agha, "Specification of real-time interaction constraints," in *ISORC*, 1998, pp. 206–214.

[85] S. Ren and G. Agha, "Rtsynchronizer: Language support for real-time specifications in distributed systems," in *Workshop on Languages, Compilers, and Tools for Real-Time Systems*, R. Gerber and T. J. Marlowe, Eds., 1995, pp. 50–59.

[86] M. Astley and G. A. Agha, "Customization and composition of distributed objects: Middleware abstractions for policy management," in *Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. SIGSOFT '98/FSE-6. New York, NY, USA: ACM, 1998. [Online]. Available: http://doi.acm.org/10.1145/288195.288206 pp. 1–9.

[87] N. Venkatasubramanian, C. L. Talcott, and G. Agha, "A formal model for reasoning about adaptive qos-enabled middleware," *ACM Trans. Softw. Eng. Methodol.*, vol. 13, no. 1, pp. 86–147, 2004.

[88] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, april 2007, pp. 254 –263.

[89] T. Nagayama, B. F. Spencer, K. Mechitov, and G. Agha, "Middleware services for structural health monitoring using smart sensors," *Smart Structures and Systems*, vol. 5, no. 2, 2008.

[90] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc.of the 1st ACM Intl. Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002, pp. 88–97.

[91] W. Kim, K. Mechitov, J. Choi, and S. Ham, “On target tracking with binary proximity sensors,” in *Proc. of the 4th Intl. Symp. on Information Processing in Sensor Networks (IPSN)*, 2005.

[92] L. Luo, T. Abdelzaher, T. He, and J. Stankovic, “EnviroSuite: An environmentally immersive programming framework for sensor networks,” *ACM Trans. Embed. Comput. Syst.*, vol. 5, pp. 543–576, 2006.

[93] A. Eisenberg, “Keeping tabs on the infrastructure, wirelessly,” The New york Times, March 2011. [Online]. Available: http://www.nytimes.com/2011/03/13/business/13novel.html?_r=1

[94] “Superstructures,” The Economist, December 2010. [Online]. Available: http://www.economist.com/node/17647603?story_id=17647603&fsrc=rss

[95] J. Rice, K. Mechitov, S. H. Sim, B. F. Spencer, and G. Agha, “Enabling framework for structural health monitoring using smart sensors,” *Structural Control and Health Monitoring*, vol. 15, p. 574587, 2011.

[96] P. Moinzadeh, K. Mechitov, R. Shiftehfar, T. Abdelzaher, G. Agha, and B. Spencer, “The time-keeping anomaly of energy-saving sensors: Manifestation, solution, and a structural monitoring case study,” in *9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, ser. SECON ’12, 2012.

[97] *TelosB Hardware Reference Manual*, MEMSIC, 2003. [Online]. Available: http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=152%3Atelosb

[98] *MicaZ Hardware Reference Manual*, MEMSIC, 2005. [Online]. Available: http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=148%3Amicaz

[99] *Imote2 Hardware Reference Manual*, MEMSIC, 2007. [Online]. Available: http://web.univ-pau.fr/ cpham/ENSEIGNEMENT/PAU-UPPA/RESA-M2/DOC/Imote2_Hardware_Reference_Manual.pdf

[100] J. Rice and B. Spencer, “Structural health monitoring sensor development for the IMote2 platform,” in *SPIE Smart Structures/NDE*, 2008.

[101] I. Decagon Devices, *10HS Soil Moisture Sensor*, Decagon Devices, Inc., 2010. [Online]. Available: http://www.decagon.com/assets/Manuals/10HS-Manual.pdf

[102] J. Aylor, A. Thieme, and B. Johnso, "A battery state-of-charge indicator for electric wheelchairs," *Industrial Electronics, IEEE Transactions on*, vol. 39, no. 5, pp. 398–409, 1992.

[103] J. Neter, W. Wasserman, and M. H. Kutner, *Applied linear regression models*, 2nd ed. Boston, MA, USA: Irwin (Homewood, IL.), 1989.

[104] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[105] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann, 1988.

[106] H. E. Kyburg Jr, "Bayesian and non-bayesian evidential updating," *Artificial Intelligence*, vol. 31, no. 3, pp. 271–293, 1987.

[107] T. Thadewald and H. Büning, "Jarque–bera test and its competitors for testing normality–a power comparison," *Journal of Applied Statistics*, vol. 34, no. 1, pp. 87–105, 2007.

[108] K. Mechitov and G. Agha, "Building portable middleware services for heterogeneous cyber-physical systems," in *Third International Workshop on Software Engineering for Sensor Network Applications (SESENA'12), pp. 31-36*, 2012.

[109] "Tinyos 2.x index of contributed code." [Online]. Available: http://docs.tinyos.net/tinywiki/index.php/TinyOS_2.x_index_of_contributed_code

[110] K. Mechitov, W. Kim, G. Agha, and T. Nagayama, "High-frequency distributed sensing for structure monitoring," in *in Proc. First Intl. Workshop on Networked Sensing Systems (INSS 04*, 2004.

[111] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, no. 3, pp. 325 – 349, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870503000738

[112] J. Al-Karaki and A. Kamal, "Routing techniques in wireless sensor networks: a survey," *Wireless Communications, IEEE*, vol. 11, no. 6, pp. 6–28, 2004.

[113] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization for wireless sensor networks: a survey," *Ad Hoc Networks*, vol. 3, no. 3, pp. 281 – 323, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870505000144

[114] G. Anastasi, M. Conti, M. D. Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537 – 568, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870508000954

[115] T. Nagayama, P. Moinzadeh, K. Mechitov, M. Ushita, N. Makihata, M. Ieiri, G. Agha, B. F. Spencer, Y. Fujino, and J.-W. Seo, "Reliable multi-hop communication for structural health monitoring," *Smart Structures and Systems*, vol. 6, no. 5, pp. 481–504, 2010.

[116] E. C. Freuder, "2. constraint solving techniques," vol. 131, pp. 51–74, 1992.

[117] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "Tinyos: An operating system for sensor networks," in *Ambient Intelligence*, W. Weber, J. M. Rabaey, and E. Aarts, Eds. Springer Berlin Heidelberg, 2005, pp. 115–148, 10.1007/3-540-27139-2-7. [Online]. Available: http://dx.doi.org/10.1007/3-540-27139-2-7

[118] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC language: A holistic approach to networked embedded systems," *ACM SIGPLAN Notices*, vol. 38, no. 5, pp. 1–11, 2003.

[119] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *Network, IEEE*, vol. 18, no. 4, pp. 45 – 50, july-aug. 2004.

[120] J. Scott, A. Brush, and R. Mahajan, "Augmenting homes with custom devices using. net gadgeteer and homeos," in *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM, 2012, pp. 213–214.

[121] D. Sturman and G. Agha, "A protocol description language for customizing failure semantics," in *Reliable Distributed Systems, 1994. Proceedings., 13th Symposium on*, 1994, pp. 148–157.