

SAMSON:
Strong Multi-Agent Simulation of Wireless
Sensor Networks

Alexis Morris



Master of Science
School of Informatics
University of Edinburgh
2008

Abstract

Distributed systems have increased in scope and complexity and have brought new challenges in interaction design and management of decentralized components to the forefront of both industry and research (Kephart and Chess (2003)). One solution to handle this new scope is to develop systems that are ever more dynamic, self-aware, adapting, and optimizing (Wolf and Holvoet (2003)). Such “autonomic systems” merge the research advancements in the field of multi-agent software design, dynamic analysis, and decentralized control (Wolf and Holvoet (2003)) in order to assist designers in constructing complex distributed systems. In this new approach, interaction management is a central element of problem solving, (Jennings (2000)), and study of designs require studying interactions of decentralized and self-interested agents; commonly through simulation techniques.

One domain that is particularly suited to autonomic designs is the wireless sensor network (WSN), as it consists of many distributed nodes in diverse and changing environments, that must interact over a common communication channel. Researchers have only just begun to apply multi-agent system design techniques in order to study improvements gained on such networks (Kho et al. (2007)), (Ruiz et al. (2005)), (Rogers et al. (2005)), (David Marsh (2004)), etc. Agents used in current research do not contain fully deliberative (or strong) reasoning systems, due to limitations of the current sensor hardware. Since hardware increases rapidly, it is expected that such systems will eventually be viable. This work aims to provide a generic and extensible simulator for further testing of interactions in strong multi-agent systems for WSN's.

To achieve this the belief, desire, intention (BDI) agent of Rao, (Rao and Georgeff (1995)), is used, as well as the Agentspeak language, (Rao (1996)), and the JASON framework (Bordini et al. (2007)). Additionally a model of current sensor hardware, (Sentilla (2008b)), is presented to show that strong agents are able to easily reflect common uses of WSN's in practical scenarios. The STATUS project of ArsLogica (ArsLogica (2008b)), and the classical WSN Flooding protocol, (Akkaya and Younis (2005)), are used as testcases for the simulator concept and models.

Acknowledgements

Thanks to all of the people who have provided guidance to me in producing this thesis. In particular, thanks to my supervisors in Trento (Dr. Paolo Giorgini), and Edinburgh (Dr. Philip Wadler), for giving insight on the workings of a Master's project, and for the many discussions on the topic, especially for providing the flexibility to modify the directions when needed. Special thanks to Sameh Abdel-Naby for his excellent optimism and enthusiasm; to Jomi Hubner, Rafael Bordini, and the Jason Newsgroup for dedicating time to answer all my questions; to the members of ArsLogica (Luca Debiasi, Fabrizio Stefani, and Fernando Pianegiani) for their interest, feedback, and time spent discussing the project; to Andy Hunt for his excellent book on research project management. Finally, thanks to those who encouraged me along the way.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Alexis Morris)

To my family and friends.

Table of Contents

1	Introduction	1
1.1	Overview of Autonomous System for WSN's	2
1.2	Problem Definition	3
1.2.1	Expected Results	3
1.3	Stakeholders	4
1.4	Thesis Structure	5
2	Motivation	6
2.0.1	Key Factors	7
3	Autonomic Computing Systems	8
3.1	Autonomic Computing's Vision	8
3.2	Designing Autonomic Systems	10
3.3	Discussion about Autonomous Computing	11
4	Agent Systems	12
4.1	Agents and Programs	12
4.2	Agents Defined	13
4.3	Agent Benefits	14
4.4	Agent Classifications	14
4.4.1	Agent Internal States	15
4.5	Weak Agents	16
4.6	Strong Agents	17
4.6.1	Platforms for Strong Agency	18
4.7	Multi-Agent Systems	19
4.7.1	Emergence in Multi-agent Systems	20

5	Wireless Sensor Networks	22
5.1	Classifying Sensor Networks	23
5.2	WSNs as Autonomous Systems	23
5.2.1	Testing Autonomous WSN Systems	25
6	Agent Implementations in Wireless Sensor Networks	27
6.1	Weak Agents in WSNs	27
6.2	Strong Agents in WSNs	29
6.3	Simulating Agents for WSNs	30
6.4	Discussion	31
7	Proposed Solution	32
7.1	Objectives	32
7.1.1	Assessment Focus	33
7.2	Methodologies	33
7.2.1	Agent Platform Selection	34
7.2.2	Agent Methodology Selection	34
7.2.3	Simulation Framework Selection	35
7.3	TMote Sky Hardware Platform	36
7.3.1	Key Features and Components	37
7.3.2	TMote Sky vs Other Sensors	38
8	SAMSON Simulator Designs	39
8.1	SAMSON Architecture	40
8.1.1	JASON	40
8.1.2	Agentspeak	42
8.2	Hardware Models	42
8.2.1	Micro Controller Unit (MCU)	44
8.2.2	Sensors and actuators	45
8.2.3	Radio Transceiver	47
8.2.4	Power Supply and Consumption	47
8.3	Environment Model	49
8.3.1	Wave-based Propagation	50
8.4	View Component	51
8.4.1	Tileworld	51
8.4.2	Dialogs	52

8.5	Controller	52
9	Testing Scenarios	55
9.1	Agent Based WSN Flooding	55
9.1.1	Designs	56
9.2	Agent Based Server Monitoring	56
9.2.1	WSN Serverfarm Management	57
9.2.2	Agent-based WSN Serverfarm Management	59
10	Results and Evaluation	64
10.1	Simulation Environment	64
10.2	Expected Results	64
10.3	Results	65
10.3.1	Agent-based Flooding	65
10.3.2	Agent-based Server Management	65
10.4	Discussion and Evaluation	66
11	Related Work	70
11.1	T-Mass	70
11.2	AgentFactory/J-Sim	71
12	Conclusions	72
12.1	Possibilities for Future Work	73
A	Appendix A - Implementation Details	74
B	Detailed Agent Design	79
	Bibliography	83

List of Figures

1.1	SAMSON Simulator for BDI-WSNs.	4
3.1	An architecture for Kephart's autonomic element, as seen in (Kephart and Chess (2003)).	9
4.1	Agent environment and dimensions, taken from (Chrisman et al. (1991)).	16
4.2	An overview of multi-agent systems, from (Jennings (2000)).	20
5.1	Vinyals taxonomy for WSN's, from (Vinyals et al. (2008)).	24
7.1	The SENS simulator design, from (Sundresh et al. (2004)).	36
7.2	The TMote Sky sensor used as the hardware model; image from (Sentilla (2008b)).	37
8.1	The simulation architecture, following the MVC pattern.	40
8.2	The JASON/Agentspeak interpreter cycle for BDI, from (Bordini et al. (2007)).	41
8.3	Class hierarchy for SAMSON, inspired by SENS (Sundresh et al. (2004)).	44
8.4	Sensor Network World view in SAMSON (shows nodes and random obstacles).	52
8.5	Dialogs and charts in SAMSON for Tile and Node properties, and Average power.	53
9.1	Simple flooding scenario with 50 nodes.	56
9.2	Prometheus design showing the system analysis overview - goals, agents, percepts, and actions.	59
9.3	Prometheus design showing the system goals.	60
9.4	Prometheus design showing the system roles.	60
9.5	Prometheus design showing the system roles and actions.	61

9.6	Prometheus design showing the system roles and actions.	61
9.7	Prometheus design detail for a data collector agent.	62
9.8	Prometheus design detail for a clusterhead agent.	63
9.9	Prometheus design detail for a basestation agent.	63
10.1	Results gained from running the Flooding testcase.	66
10.2	Network reformation test consisting of 15 randomly distributed clusterheads and 1 basestation.	67
10.3	Power consumption levels of a datacollector agent.	67
10.4	Averages of messages sent and received, and power remaining after testing.	68
A.1	Class diagram overview.	75
A.2	Full system class diagrams.	76
A.3	Sequence diagram for sending a message.	76
A.4	Sequence diagram for receiving a message.	77
A.5	Sequence diagram for sensing the environment.	77
A.6	Sequence diagram for changing the radio transmission power.	78

List of Tables

7.1	Several simulation platforms considered at design time.	34
8.1	MCU States according to (Sentilla (2008b))	45
8.2	Sensors included in the model of the TMote Sky unit (Sentilla (2008b)).	46
8.3	Radio based parameters that are used in SAMSON, from (Sentilla (2008b)).	48
8.4	Overall System states, and power consumption levels taken from TMote nominal behaviors (Sentilla (2008b)).	49
8.5	Attenuation coefficients for obstacles of different thicknesses, according to (Karl and Willig (2005))	50
8.6	Actions for SAMSON agents.	54
10.1	Table showing average results from the Server scenario.	69

Chapter 1

Introduction

Wireless sensor networks are built from small, low powered computing devices that measure physical factors of the environment, through sensor technology, and relay this information to an interested party via wireless communication. This type of distributed system platform is now considered one of the hot topics in computer science, as it has great potential to embed computing power in everyday objects (Sentilla (2008a)). As the technology becomes smaller, and more power efficient, it will eventually be used in every facet of pervasive computing. The techniques involved in operating and maintaining such a system requires having a deployment of sensor devices, programmed correctly to perform a task, or many tasks, and a means of querying the system for desired data from a remote location.

There are many uses for WSN's, and they have been shown useful in measuring volcanoes, borders, building structures, amongst other uses. Ambient intelligence systems, for instance, make use of data harvested from sensor networks in order to configure the proper temperature to a building, or proper lighting, based on a host of sensor data. Currently the range of physical sensing components small enough for use on the tiny sensor node is vast, from accelerometers to sound sensors, to temperature. Communication for these devices proves to be absolutely important, as it often takes many hundreds of sensors spread over a large field in order to adequately monitor the types of dynamism found in a real environment.

Sensor networks are undoubtedly important, and will become even more so as the technology improves, allowing for better power efficiency, processing, and storage capabilities on a single device. With this added ability comes added challenges for designers. Complexity in the environment combines with the dynamic nature of the many varied problem domains, and limited intelligence of the sensor device often make

it difficult to deploy even a simple WSN. Testing of WSN deployments also becomes problematic, as conventional methodologies are not able to compensate for the number of variables that the device can encounter. One solution to these problems is to use cleverly designed, complex, distributed algorithms to make the system more robust. The other is to create an autonomous wireless sensor network.

1.1 Overview of Autonomous System for WSN's

Autonomous systems are designed to face precisely the problems mentioned, making it a good fit, and a natural combatant for the increase in complexity found in many WSN deployment domains. Systems that use such an approach are flexible, adaptable, self-organizing, and self-optimizing (Kephart and Chess (2003)). This allows designers to solve distributed problems using interaction design policies rather than algorithms (Wolf and Holvoet (2003)). In fact, researchers are already taking advantage of the approach in order to solve common distributed system problems such as routing, active sensing, information processing, and collaborative sensing strategies (see (Vinyals et al. (2008)) for a complete survey). The autonomous WSN (AWSN), as a platform is able to achieve results such as improved system longevity (Rogers et al. (2005)), and coverage density (Yadgar and Kraus (2006)), two of the most common WSN problem areas. This leads to the conclusion that autonomous systems are more fitting, and designs with autonomous properties have become common in the research (Vinyals et al. (2008)).

Autonomous systems rely heavily on the research in the multi-agent system domain, as well as dynamic analysis and decentralised behaviour protocols. Hence, the problem is largely one of multi-agent system design. In this work, the use of agents in wireless sensor networks is discussed, and the focus is on the class of agents used for WSN design. The majority of current approaches present agents that are non-deliberative, meaning that they do not have a fully functioning reasoning system, and hence are not reactive to the environment, or “non-rational agents.” These agents are considered as weaker than “rational” agent designs primarily because of a lack of deliberative components. In particular, the belief-desire-intention (BDI) style of agent, as shown by Rao (Rao and Georgeff (1995)), is commonly held to be the standard model of strong agency (Jennings (2000)). Unfortunately, with added deliberative ability comes extra operational costs, which prevent the implementation of BDI agents on present hardware.

Little has been discovered about strong agents in wireless sensor networks, despite the benefits a weaker agency have brought to the platform. Since hardware designs improve according to Moore's law, it will soon be plausible for systems to accommodate fully deliberative agents. At present, however, there is no established medium for studying strong agents in WSN's through simulation techniques. In fact, based on this research there are only two existing simulation environments that are BDI-like, although not directly based on the work of Rao (Rao and Georgeff (1995)), and neither approach has shown results in a realistic WSN scenario. This paper aims to address this area by providing a third simulator, based directly on Rao's BDI agent model, and also by testing it on two WSN problems.

1.2 Problem Definition

Several robust simulation environments like TOSSIM, (Levis et al. (2003)), NS2, (Riley (2003)), and J-Sim, (Sobeih et al. (2005)), exist for testing WSN's but they are generally inadequate for representing agent oriented designs. This has led researchers of the AWSN field to build various test environments for a number of different scenarios and agents (such as (Rogers et al. (2005)), (David Marsh (2004)), (Kho et al. (2007)), (Ruiz et al. (2005)), (Yadgar and Kraus (2006))). What is noticed from the literature is that there is little agreement on the class of agents used for WSN's, and less agreement on simulators designed. This project has aimed to design a general purpose simulator that is based on rational agents and that also closely models current sensor network hardware.

1.2.1 Expected Results

The result is the SAMSON simulator for wireless sensor networks. SAMSON is composed of a hardware model that simulates the workings of the TMote Sky sensing unit, as well as its radio operations, and wave propagation through the environment. The environment itself is represented as a TileWorld (Pollack and Ringuette (1990a)), having cells with environmental properties. A user is provided with interfaces that allow for moving nodes, adding obstacles with various settings (such as radio signal attenuation factors), editing environment and node parameters such as power settings and radio states, and viewing charts for consumption levels and average system battery life. A snapshot of the environment is seen in the figure 1.1.

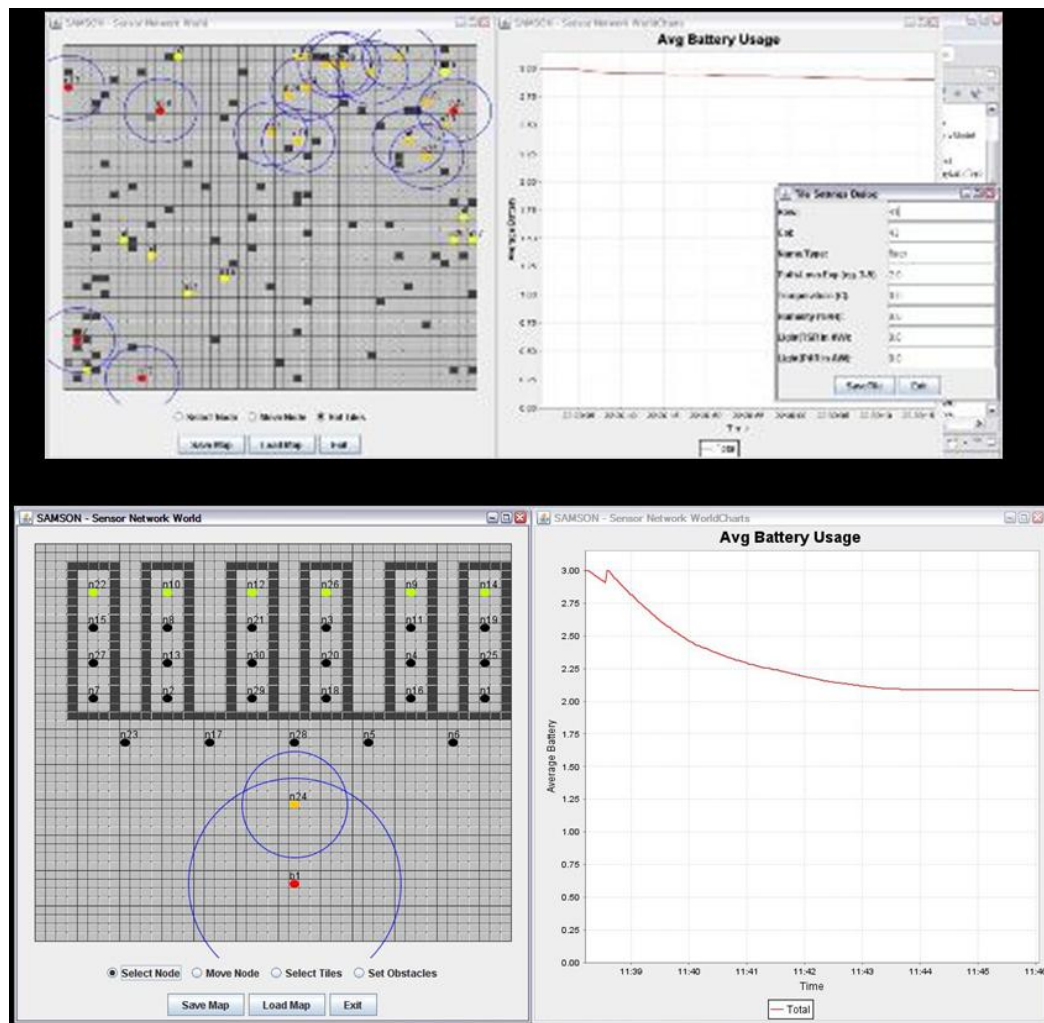


Figure 1.1: SAMSON Simulator for BDI-WSNs.

The agent based component of SAMSON relies on the work of Rao for the Agentspeak language (Rao (1996)), and the work of (Bordini et al. (2007)) for the JASON/AgentSpeak interpreter. JASON provides SAMSON with a procedural reasoning system that especially incorporates agent design based on Beliefs, Desires, and Intentions (BDI). This control layer adds to its grid-based view, and general, object-oriented model of TMote Sky sensor components, and the environment to create a generic AWSN simulator.

1.3 Stakeholders

In particular, this work has enjoyed support from an Italian WSN development and research company, ArsLogica, (ArsLogica (2008a)), which is affiliated with the University of Trento, and performs research as well as actual WSN deployment efforts. The

company is largely interested in the simulator development, and its usefulness in modeling scenarios, with a particular focus on its hardware settings. Support for the work has involved providing useful information about the hardware, the TMote Sky sensor, (Sentilla (2008b)), and a selected implementation scenario, (ArsLogica (2008b)) for study. Several demos of the simulation solution have also been successfully conducted with the company during the course of the work.

1.4 Thesis Structure

- Chapter 2 of this paper provides a look at the key motivations in the project.
- Chapter 3 discusses the concept of autonomous computing systems and how they are characterized.
- Chapter 4 introduces agent based systems, discussing the benefits and categorizations of these, as well as multi-agent systems.
- Chapter 5 overviews the wireless sensor network concepts, and discusses the WSN's fit as an autonomous system.
- Chapter 6 reviews some of the work found in the literature for implementations and simulations of agent oriented WSN's.
- Chapter 7 presents the proposed solution, and the justification for the methodologies used.
- Chapter 8 gives the details with regard to the SAMSON simulator design components, architecture, models and interfaces.
- Chapter 9 shows the two main testcases for the simulator, highlighting a basic scenario, and a realistic industry scenario, along with agent designs and implementation details.
- Chapter 10 evaluates the results of the testing.
- Chapter 11 provides a comparison with related work.
- Chapter 12 concludes the paper.

Chapter 2

Motivation

There are many reasons in favor of studying agent designs in wireless sensor networks. For instance, it is useful to be able to study properties of negotiation or coordination in such networks through simulation. Testing protocols that can have power saving properties, or data allocation scenarios are only two examples that benefit from being able to simulate agents in wireless sensor networks. At present, however, there is no suitable simulator that is both general purpose, and flexible enough for a host of different application criteria, and it is common for researchers to attempt to find an existing, non-agent based WSN simulator (such as TOSSIM or NS2), or to build one for the application. However, when this is done the result is tightly coupled to the problem domain. The aim in this paper is to provide a simulation environment for general use.

In designing a generic agent oriented wireless sensor network simulation tool there are three main problems, namely the hardware modeling, software utility, and the environment modeling. These all factor into the accuracy of the simulation, improving its usefulness in industry and research. In particular the hardware model should be reasonably accurate as a reflection of existing hardware. Software should be readily adaptable to actual implementation designs for real hardware (ideally it should be easy to convert from simulation code to actual code, as is done by some traditional WSN simulators like TOSSIM), finally the environment model should be a reflection of the actual properties, in this case obstacles and propagation of radio waves for communication. These all present a significant challenge in the design of a simulator for autonomic wireless sensor networks.

2.0.1 Key Factors

The primary motivating factor for the project is that a simulation environment for agent based wireless sensor networks is needed in order to make future studies on collaboration, agent based routing, and negotiation. This environment must be able to quantify results that are practical and near to actual hardware. The system should be generic, flexible for handling future cases, and future hardware architectures. It must also be useful for “agentifying” existing WSN scenarios.

Secondly, the motive is to show the benefits of strong agents/deliberative, or rational agents, in a realistic environment. This also entails discovering the problems that may be found with agents in WSN’s, if they arise. Furthermore the work of software simulation will allow for testing the BDI mechanism . At present the highly acclaimed Agentspeak language (Rao (1996)), and the JASON interpreter (Bordini et al. (2007)), have not yet been used in this context. It is not the first BDI WSN simulation, however, but (at the time of writing) the first to use Agentspeak.

A further motivation for this work comes directly from industry. ArsLogica, as mentioned earlier, is looking for a general tool to assist with testing and deployment exercises, with good reason. WSN Deployment scenarios and actual implementations are significantly diverse and involve a number of different parameters (Vinyals et al. (2008)). In ArsLogica’s case two examples have to do with traditional ”sense and send” applications (Karl and Willig (2005)). In a sense and send application the sensor nodes are deployed with an objective of gathering and making summary data and then forwarding this data to a basestation node through some efficient routing protocol. This is the typical sensor network usage case, and the company would like to be able to test before deployment. Especially important is the scalability of the network. Developers would like to simulate the topology of the network, along with general algorithms in order to gain a global view of the network. Furthermore, the company is looking to understand what agents are, exactly, and how adding agents to their applications can benefit the designs. Finally, the simulation environment would allow the company to show demos of sensor network plans and application designs to clients; which is a value add for having such a tool. So we see that having a simulation environment that is flexible, and agentified or autonomous, is helpful for the goals of Arslogica, which represent industry practitioners.

Chapter 3

Autonomic Computing Systems

3.1 Autonomic Computing's Vision

Autonomic computing is discussed by Kephart and Chess, of IBM labs, (Kephart and Chess (2003)), as being systems that are able to function much like the body's autonomous nervous system, which takes care of the majority of the small details in normal operation scenarios of its own, allowing the brain to have the resources to focus on higher level tasks. These systems are designed to be self-managing to a large degree, in order to allow system architects to focus solely on policy-making, and goal design. In (Kephart and Chess (2003)), self management entails features of:

- Self configuring - Such systems are able to automatically arrange itself and its components into a reasonably coherent organizational structure based on high level directives given by the system designers.
- Self optimizing - The system should be able to seek out ways able to improve itself where it is possible to do so.
- Self healing - The system should be able to anticipate failures and instantiate recovery policies without the intervention of outside forces or human actors. This could be in both software or hardware related problems.
- Self protecting - The system should be able to anticipate security threats and take measures to avoid them. It should also actively seek out the latest updates if it needs to, and install and reconfigure itself as needed.

It needs no saying that such systems are a long long way off from any active use or even any valid design. However, the authors of the paper make it reasonably clear that

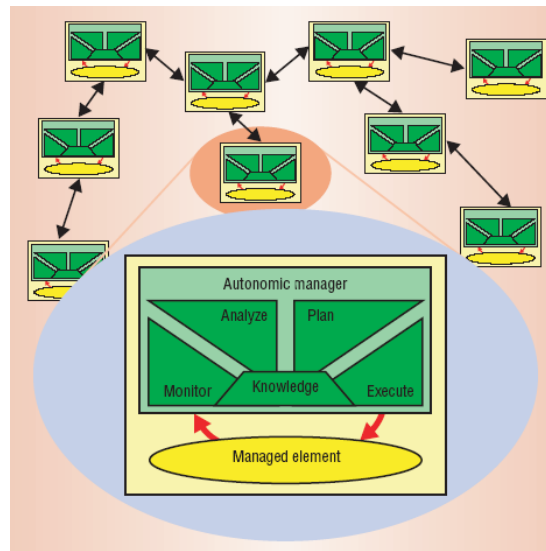


Figure 3.1: An architecture for Kephart's autonomic element, as seen in (Kephart and Chess (2003)).

such systems are essential, since the degree of complexity in systems design is growing rapidly, and program designers and information architects are faced with increasing difficulties in deploying, testing, debugging, and maintaining distributed systems. It is expected that as systems become more interactive, and network oriented, interactions among them will make complexity unmanageable with present approaches (Kephart and Chess (2003)). This complexity snowball led IBM to push for a new direction in research, calling for a joint effort to solve the challenge of building and designing autonomous systems. Today, this challenge remains, although there have been efforts to classify, and study the concept.

The vision for such systems defines computing that is goal oriented, and interactive, based on layers of active “autonomic elements.” These autonomic elements are software systems that have an architecture that continuously senses the environment in order to react to changes as it performs its functions. Each autonomic element is goal driven, and responsible for a specific function; an architecture which is very similar to those discussed in the agent-oriented software engineering circles (Wooldridge and Ciancarini (2000)). These elements consist of a component to be managed, and an autonomic manager that has a knowledge base and functions for analysis, planning, monitoring, and executing various actions. In such systems it is highly important that the manager be able to select its actions, and responsively handle problems; perhaps even generating its own behaviour plans, depending on how it may achieve its goals.

3.2 Designing Autonomic Systems

De-Wolf and Holvoet (Wolf and Holvoet (2003)) have outlined what they consider to be the key components of autonomous systems engineering, building on existing software engineering paradigms. In particular they discuss three components; agent modelling, in order to model the dynamics of the system; dynamical systems theory in order to analyze the interaction dynamics; and decentralized control systems in order to provide the necessary policies and mechanisms for the system to produce expected results at global and local levels. These three building blocks are suggested as the starting point for the process of engineering such systems. For modelling, the agent oriented paradigm provides an ease of representation, as it has advanced methodologies that are adaptable for interaction design, autonomy engineering through goal analysis, and is inherently modular. Additionally, multi-agent systems are easily converted to distributed systems and may be simulated. Dynamic systems allow for analysis of the system itself over time, based on obtaining metrics about the system state space, trajectory, and attraction behaviors. Tracking the movement of the system through the state space allows for making decisions that control the system, and gives an understanding of attraction elements which can cause the system to move permanently into a sub-space of the possible states. This analysis information is used in performing decentralised control mechanisms to make the system fit requirements through emergent properties, such as the work on computational mechanism design (Dash et al. (2003)). Some of the approaches for controlling such systems (as found in (Wolf and Holvoet (2003)))are:

- Hierarchical control - systems that have operations of control based on roles. One layer does data handling, for instance, while the other does data transportation.
- Market based control - systems that are controlled by economic theories such as maximizing a utility function based on the value of some market variable.
- Environmental blueprint control - systems that are constrained by following the environmental landmark as a guide beacon, as found in localization problems (Karl and Willig (2005)).
- Constraint-based control - systems that work by operating within constraint parameters.

- Genetic algorithm controls - systems that are controlled by genetic algorithms, such as automata (Wolf and Holvoet (2003)).
- Physical laws - systems that are controlled by physics related functions.
- Adaptive organizational controls - systems that are controlled through the formation of a structure in real time.

3.3 Discussion about Autonomous Computing

De Wolf (Wolf and Holvoet (2003)) also makes the statement that interaction design is more powerful than algorithms, showing the importance of interaction in autonomous systems. This is an especially favorable result, since such systems are ones which are highly interaction based, meaning that modelling and design for interaction cannot be forgotten when designing such systems. Jennings, (Jennings (2000)), mentions however, that interaction design is difficult to develop, and especially difficult to debug. Hence both make it clear that tools are going to be essential as the young field of autonomous systems design takes off.

Chapter 4

Agent Systems

As mentioned, the first and most important tier of Autonomic systems is the use of agents as a central modeling paradigm for the interactions, and general design of the system components (Wolf and Holvoet (2003)). As discussed by Newell (Newell (1980)), the benefits of agent systems are primarily seen in the fact that there is a knowledge level, as well as a social level (Jennings (2000)) of computing. These two levels combine when multi-agent systems are used, allowing for system components to share knowledge through social interaction. As mentioned in the previous chapter, interaction techniques are often better than the use of conventional algorithms to solve problems (Wolf and Holvoet (2003)), thus giving a segway to a discussion of agent systems.

4.1 Agents and Programs

First, when discussing agency, one has to distinguish what is an “agent” from what is otherwise just another thread-based “object,” or any other program (Wooldridge and Ciancarini (2000)). According to agent researchers, agency is measured in accordance with the complexity of their internal state (Chrisman et al. (1991)), a spectrum that ranges from low degrees of complexity to high degrees. These are broken down in more detail according to the ability of the program to decide not only what task to work on, but also when to perform a task (Wooldridge and Ciancarini (2000)). The traditional java object, for example, acts only when “called” based on an initiator program, preset by the author. It does not decide whether or not to execute itself. Agents, however, are able to make this decision. Since agents are encoded using similar constructs as objects, and sometimes even using the same java language, it becomes unclear how

different agents are. However, the agent paradigm represents a shift in the sense that the agent based program a) has control over its thread of execution abilities (whatever methods it may call, it may decide at any time to do so) b) has a method of continuously/sporadically scanning the environment for new input or changes that may affect its decisions. c) has the ability to stop what it is doing at any point and do something else d) finally, agents typically rely on communication from other agents/actors in the environment, adding some degree of economic/social ability. Agents traditionally are self-interested (ie. they are greedy, making focus on either maximizing or minimizing some utility function, according to the designs). The results of the utility influence the action of the agent program. Objects, however, lack such functionality, although they may be augmented, or “agentified” with these agent behaviors.

4.2 Agents Defined

Agent Systems have many definitions in the literature, but two fitting ones are that of Franklin, (Franklin and Graesser (1996)):

“An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.”

And a similar definition of Jennings, (Jennings (2000)):

“An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives.”

Both these definitions combine to highlight the important features of agents, however, there several key points that are always considered, namely that agents are Autonomous, Reactive, Proactive, and Social systems, as seen in (Wooldridge and Ciancarini (2000)).

- Autonomous - agents are able to control when they will act, based on their own goals and perceptions of the world.
- Reactive - agents are a part of an environment that they can sense and control in some way through effectors. Reactive agents are able to respond to the environment as needed.
- Proactive - agents are directed and goal oriented, taking steps towards creating a world state where their goals succeed.

- Social - agents are able to communicate through some medium to other agents, or actors.

These are good definitions because they describe agents and agency in a way that is both flexible, yet still rule out programs that are definitely not agents, while also including many kinds of agent, including humans, ants, and thermometers (Franklin and Graesser (1996)). Nonetheless, it is enough to consider such systems as those that are continuously scanning their environment, and making changes, as needed.

4.3 Agent Benefits

As Wooldridge shows in (Wooldridge and Ciancarini (2000)), agents are useful for providing four key benefits; a natural abstraction method of designing solutions to complex problems involving interactions; they are inherently distributed systems; they can be used to evolve legacy systems; and they are open systems that are flexible and adaptable to changes. Jennings, (Jennings (2000)), calls such systems a logical evolution to traditional software engineering, and has also proposed two reasons in support of the paradigm. One is that of adequacy: being able to improve the method of building complex distributed software systems. The other is that of establishment, being able to be accepted by mainstream software system designers as a standard approach. These benefits provide the essentials for modeling, with the added ability of being able to make use of high level interactions and changing organizational relationships to tackle important problems (like that of the snowballing complexity in the distributed systems industry, mentioned previously).

4.4 Agent Classifications

At the highest level there are two ways to classify an agent system, namely if it is a social system involving multiple agents, or an asocial system of a single agent. Undoubtedly the social system is the more interesting, but first the different types of agents must be looked at from a strictly asocial perspective.

First, in order to classify agency it must be admitted that there is a spectrum of agency that ranges from a fully functional program, such as those written in C or Haskell, to those programs that are fully reactive, (Wooldridge and Ciancarini (2000)). Functional systems are those which take an input, transform it somehow and produce

an output. Purely functional systems are not able to control their execution once they have been initiated. Once the program enters a function call, it must run to the end of the call without halting, even if something changes during the execution. Purely reactive systems, on the other hand continuously sample the environment for changes, and react to them immediately in some way. These programs are always able to curb, or halt activity since they respond to each change in the environment. Agents, however, fall somewhere in between being functional enough so that they can get tasks completed, but yet reactive enough so that changes in the environment will be detected, allowing the agent to adapt smoothly. This type of system is noted by Wooldridge as being a “Pnuelian reactive system,” (Wooldridge and Ciancarini (2000)), and by Rao, (Rao and Georgeff (1995)), as having the degree of reactivity termed as “commitment”. An agent may be designed to either be blindly committed to an action plan that it has initiated, or it may be single minded so that it changes its goals based on changes in its beliefs about the environment or conflicts, or it may be open minded so that it changes its plans based on changes in either its beliefs or goals. So it is seen that the level of commitment represents a particular spectrum of agency (Rao and Georgeff (1995)). Agents must strike an appropriate balance between the purely functional end of the spectrum and the purely reactive, and the ways that this is done is determined by the commitment mechanisms.

4.4.1 Agent Internal States

In addition to the commitment spectrum, Chrisman, (Chrisman et al. (1991)), has presented three particular classes of agent internal states, namely the model reactive, expectation utilizing, and contingency anticipating states. For the model reactive agents, the agent maintains a record of the state of the world and uses this to make decisions about its actions. It does not record the future state of the world or past decisions, only the current information about the world. Expectation utilizing agents maintain preset plans of action that it uses in combination with information about expected future state of the world, as well as its senses of the current world state (which are not stored). This means that the agent operates on a set of plans for what to do, but uses current sense of the world to decide how to continue with the plans. The third class of internal state, contingency anticipating, is one in which an agent faced with a choice, based on what it currently perceives from the environment, anticipates all possible future states of the world and makes an appropriate decision for each state. This is all done before

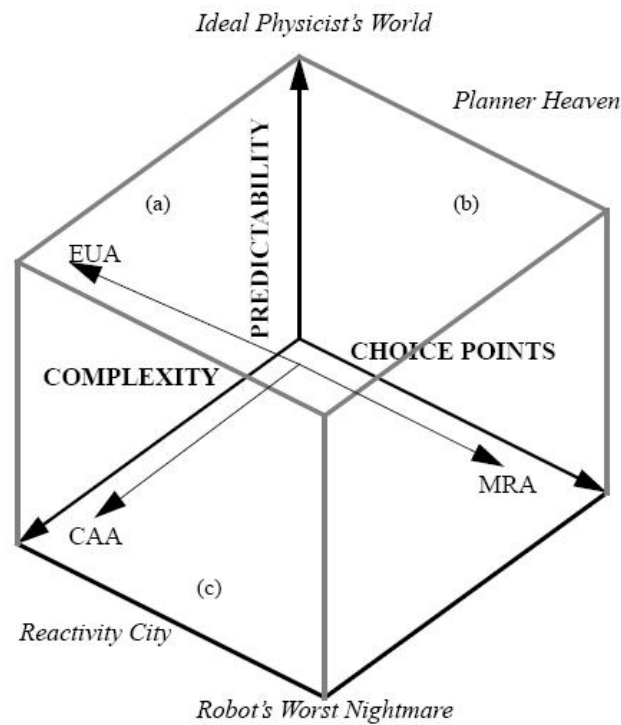


Figure 4.1: Agent environment and dimensions, taken from (Chrisman et al. (1991)).

the agent actually initiates the action choice.

Chrisman, also provides a view of the environment dimensions where these three classes of agent are most effective. These three dimensions are the predictability of the environment, the complexity of the world state, and the number of decision points to be made by the agent. Model reactive agents are most effective when the environment has many possible decisions, or choice points, and the world is reasonably non-complex. Predictability is not a factor. Expectation utilizing agents are best when the environment is highly predictable, and the possible decision points are low. Complexity is not a factor. Contingency anticipating agents are best when the world is complex, but choice points are low. Predictability, and complexity is not a factor.

Based on these agent design considerations, two types of agent classes are used, those of weak agents, and strong agents. This is based on the level of internal state for committedness, and reactivity.

4.5 Weak Agents

Weaker agents are typically those without deliberation capabilities, although they may be programmed with self interested behaviors. This type of programming, when com-

bined in a distributed environment, produces agent results, despite the fact that the level of commitment of the agents is not present. In particular, such approaches make use of three techniques, as seen in the literature. The first is the use of policy based programming, which includes role based agents. A policy is a simple rule for behavior control, termed a constraint based approach in the previous chapter. Some form of policy based control is necessary in just about all weak agent systems, however, without the presence of strong BDI architectures its agents have little reactive choice about which plans to follow, but only follow the policy (or role) blindly (an example is found in Ruiz, et al's MANNA system for fault management (Ruiz et al. (2005))). Similarly, a second common approach is seen in agents that make use of game theoretical approaches (such as those that solely use computational mechanism design, or solely utility maximization functions, as in (Rogers et al. (2005))). The third technique is seen in the use of middleware programs that act as mobile agents when in the presence of a distributed network, see (Fok et al. (2005)). Each mobile agent is executed on a middleware system and may migrate through the different hardware of the network by forwarding its internal state. Such systems are not as strong as BDI style agents because of their lack of a reasoning engine for selecting plans.

4.6 Strong Agents

The three classifications mentioned previously about Model Reactive, Expectation Utilizing, and Contingency anticipating, are states which may be combined in different ways for different types of agents, as needed. This section looks at a class of agent that does this in order to strike a balance between these three internal states, while at the same time having an open-minded level of commitment. This is the class of intentional agents, otherwise known as belief-desire-intension (BDI) agents. BDI Agents are those which have an internal state that consists of a store of world facts (beliefs), a store of goals (desires), and a store of plans to achieve the goal (see Rao, (Rao and Georgeff (1995))). Intentions, represent the current progress of actions started by the agent, following some plan that has not terminated. The possible worlds theory mentioned by Rao, shows that these points may be considered as a large tree structure of possible worlds that is eventually pruned according to the beliefs, then further pruned by the goals/desires, and finally narrowed to the chosen action, the intention.

This type of agent builds off of the work for the distributed MultiAgent Reasoning System (dMARS) for building a “procedural reasoning system”, and the ideas have

been extended by Rao and Georgeff to incorporate them into an abstract architecture (Rao and Georgeff (1995)). This BDI architecture has been seen to be among the most practical of agent systems, ranging from air-traffic management systems (Rao and Georgeff (1995)), to process management systems (Jennings et al. (1996)). What is important to note about the BDI agent is that it is flexible and can adapt to its environment in the model reactive sense, yet can still operate in complex worlds like the expectation utilizing agent. In some degree it is also contingency planning, since by having an architecture which processes through beliefs to desires to intentions, the “possible worlds” logic narrow from belief worlds to desire worlds and further to intentional worlds, essentially taking into account future plans (Rao and Georgeff (1995)). However, it is not strictly contingency planning since it does not regard all the future states and choices. Nonetheless, with the advantage of being smarter comes some costs. The first is the BDI agent has to have the required overhead costs and memory costs associated with keeping track of its internal states, and minimally has three databases (for beliefs, desires, and intentions). This may be a problem on some limited systems (such as the wireless sensor networks, in our case). The other limiting factor, as mentioned by Jennings, (Jennings (2000)), is that such agents do not directly allow for changing organizational structures that are inherent to multiagent programming.

Strong agency involves deliberation in the the agent in order to determine commitment levels, and choices of plans. This type of agent has been accepted by the community as being one of the best approaches (David Marsh (2004)). When it comes to the implementation for such agents, there have been several platforms that have been developed that take into account a BDI, or related approach. Other approaches are invoked when a weaker notion of agency is used.

4.6.1 Platforms for Strong Agency

Creating strong agents results from the work of Bratman, (Bratman et al. (1988)), and subsequently Georgeff, culminating in the development of the abstract BDI architecture (see (Rao and Georgeff (1995))). This architecture has been used in much of the literature, in some fashion. Below are several popular platforms for designing strong agent systems:

- JASON/AgentSpeak (Bordini et al. (2007)): A relatively new java-based implementation of the Agentspeak interpreter presented by Rao in seminal work about BDI agents (Rao and Georgeff (1995)).

- Jack ("Howden N and A" (2001)): This is a commercial, java-based platform for BDI implementations.
- JADEX (A. Pokahr (2003)): A popular BDI implementation that is able to run on remote systems, also java-based.
- AgentFactory (Collier (2002)): Has an implementation that is BDI like, involving beliefs and commitment rules. There is also a micro edition for resource constrained devices.
- PROSOCS (Bracciali et al. (2006)): This implementation works like BDI, but instead uses the concepts of KGP (knowledge, goals, and plans). It is based primarily on computational logics, rather than the modal logics of BDI driven architectures.

4.7 Multi-Agent Systems

Among the most important benefits of agent systems is its ability to perform tasks through interactions of autonomous components. Systems that involve multiple agents in a shared environment present a host of possibilities and challenges to provide adequate mechanisms for cooperation, negotiation, and coordination among the agents in the worlds. This turns out to be the primary reason why multi agent software engineering is a primary fit for Autonomic system design. The components are all local agents, and the system's behaviors rest on having interaction driven, distributed features.

The validity of the paradigm has been seen by Jennings, (Jennings (2000)), in that MAS systems attack a decentralized problem, with multiple loci of control, having multiple possible perspectives, or competing interests. Looking back at Chrisman's environment cube, above, it is clear that there are areas where the world is too complex for a single agent to function well. In such cases, having a team of such agents that can communicate and share knowledge of the world would make much better progress. However, this means that instead of having a centralized solution within a single agent, the solution objectives, and tasks must be distributed amongst the team, and looked at from the perspective of the entire team, rather than a single member. Such approaches are better able to manage systems that are dynamic, and able to be controlled from various different locations, and each agent is able to perform a unique task, while delegating other tasks to more specialized agents. This reduces the level of complexity

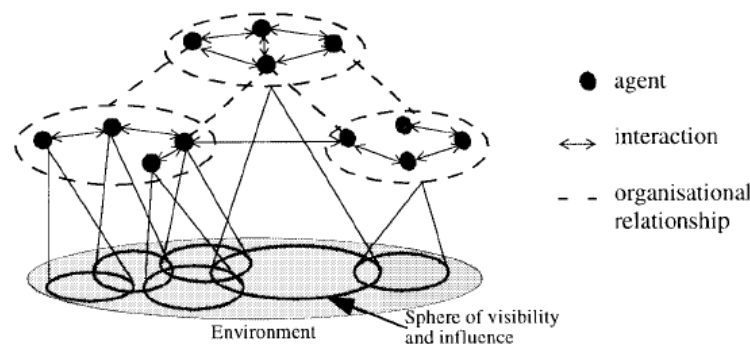


Figure 4.2: An overview of multi-agent systems, from (Jennings (2000)).

of the entire system to more manageable dimensions. Through active interactions two levels of problem solving take place, one at the local level where each agent is following its own directives (beliefs, desires), and must flexibly augur for its own success in the world, while at the same time a global level of problem solving takes place.

4.7.1 Emergence in Multi-agent Systems

Emergent behaviour in such systems is the result of the functioning of self-interested components that achieve a global state through the unique actions of each agent. The ability of the multi agent paradigm to model and design mechanisms that can control desired behaviour at this level is precisely what is mentioned by De-Wolf, (Wolf and Holvoet (2003)), as a part of the autonomic system design process. Organizational choreography in multi-agent systems becomes a challenge for designers, as the scope for designing appropriate mechanisms requires consideration of the entirety of the system, or some scalable abstraction of the system (Jennings (2000)). When this is done, however, the solution can become close to the optimum solution, as in (Rogers et al. (2005)).

As organization formation is dynamic at the local level, the system as a whole becomes dynamic in some degree. At the global level, a system made up of distributed, communicating components is flexible enough to tackle scalable problems. When agent oriented programming is used to give the individual components of the system more reactivity, this improves the general reactivity and efficiency of the whole. It may also be true that when weak agents are used the system operations benefit from this agency; and also when strong agents are used, the system becomes much more adaptable. This comparison of the system at the global and local level remains

to be fully quantified in the literature, but work on game theory, and computational mechanism design makes the claims reasonableDash et al. (2003)).

What is important to note is that agent systems design for autonomic computing is a natural fit that already incorporates elements of dynamic systems theory, and decentralized control (Wolf and Holvoet (2003)). Multi-agents, combined with mechanism designs from game theory (as seen in (Rogers et al. (2005))) are productive in creating the sorts of self managing networks that are envisioned for the future of wireless sensor networks.

Chapter 5

Wireless Sensor Networks

This chapter makes the point that the wireless sensor network is one which is naturally autonomic, in terms of its structure, and environment, and control objectives. Hence it will be clear that the WSN is also a good fit for becoming an agent system. In general, a wireless sensor network is one which consists of sensing units that are distributed in an environment, but are not connected by a physical network, but rather a wireless communication medium, such as radio. The promise of these networks is in their ability to perform long term monitoring and even control of an environment. The typical usage of such networks, involves sensing the environment, storing data, aggregating it somehow, and forwarding summaries of the data to a receiving sensor, known as a sink, or base station. The challenge in doing this lie in several areas, and is best summed up by Vinyals seven challenge areas (Vinyals et al. (2008)). These are as follows:

- Complexity - Typically the environments are complex
- Scale - WSN's may involve hundreds to thousands of sensors, and solutions must be scalable and efficient
- Physical Distribution - WSN's involve components that are distributed over potentially vast spaces, depending on the strengths of the communication medium.
- Dynamics - These systems must adapt to changes in the network itself, and changes in the functioning of other nodes is not very predictable. For instance some sensors may die, or new sensors may be added, or communication losses may occur due to low power levels, message channel conflicts and other mishaps.
- Resource Availability - Sensors operate on constrained hardware, due largely to the limits of the size of the device and the battery technology (see Marsh for

more). Although this is likely to change as Moore's Law makes technology more efficient, for the present the limits are considerable.

- Interdependence - Sensors are dependent on other sensors in the environment in order to operate more efficiently, and to maintain flexibility in the presence of dynamism.
- Situatedness - Sensors are placed in environments that are frequently changing, so decisions and actions must be made quickly.

5.1 Classifying Sensor Networks

WSN's may be grouped according to their sensors, the network, the environment, and the goal of the network, as proposed by Vinyals, et al, (Vinyals et al. (2008)). In their work they present a taxonomy where sensor networks are classifiable by power supply type, self-awareness level, dynamics, configurability, and activity orientation. The network may have varying composition, deployment, communication, dynamics, ownerships, and varying number of nodes. Additionally, the environment has a degree of dynamics, nature (or type characteristics), and some degree of observability by the nodes. Finally, the purpose of the network depends on the action of the sensors, the degree of dependency nodes have on the environment, and the effects of their actions over the environment. This taxonomy is presented graphically in figure 5.1.

5.2 WSNs as Autonomous Systems

WSN's in theory are not difficult to represent using multiagent system designs. MAS agents must interact with each other, through communication, or coordination in the presence of potentially limited resources. This is common in literature, such as Rosenschein's factory agents (Rosenchein and Zlotkin (1994)) or Kraus' data allocation servers (Kraus (2001)). The WSN is also good for agents because it is inherently dynamic, in terms of structure, deployment settings, and behavior. Nodes situated in a real world environment sample it continuously and relay information to other nodes, according to directives based on the priority of the information, or other factors. These nodes are wirelessly connected, and hence communication is broadcast based, and is typically done in a multihop routing fashion to save power, see (Karl and Willig

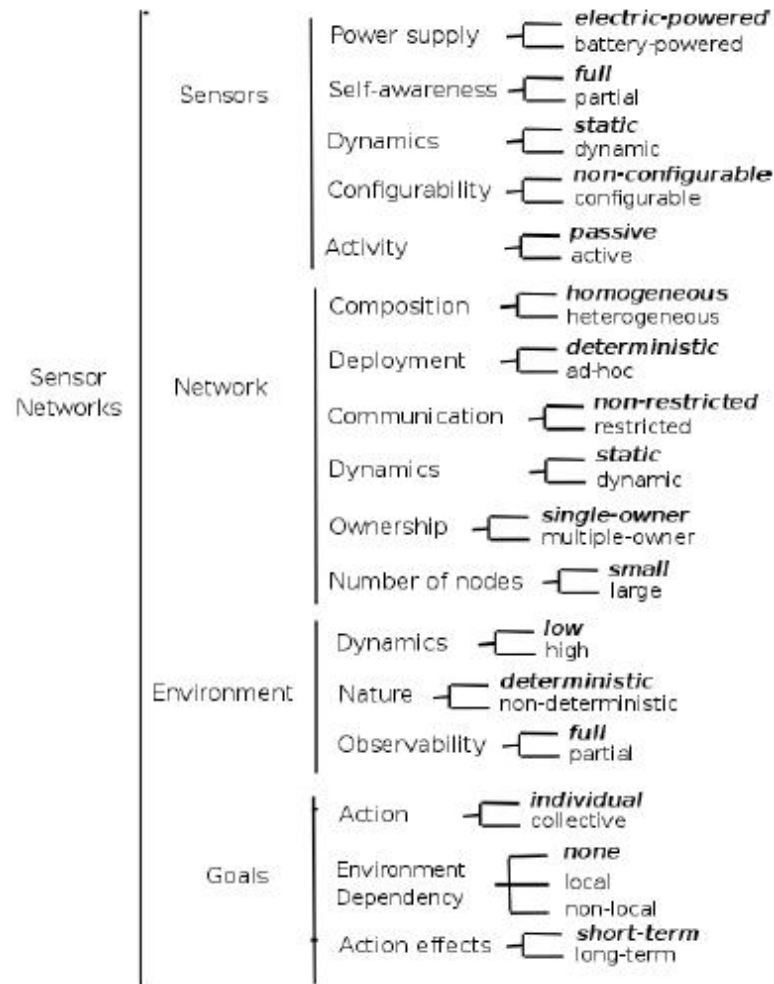


Figure 5.1: Vinyals taxonomy for WSN's, from (Vinyals et al. (2008)).

(2005)). Hence there is an interdependency among the nodes for tasks of message relay.

Additionally, WSN nodes do not have a large battery capacity, so sending, storing, and receiving messages are costly operations. Other limitations of the hardware are lack of a large memory, processing ability, and radio range limits. This means that WSN solutions must employ methods of sharing such resource costs, much like agent systems must distribute tasks. When combined with the fact that the sensors may be heterogeneous in terms of what data they are collecting, where they are sending data (perhaps they are even deployed in the same environment by two competing parties), and what their current position is in the environment, it is evident that nodes in a WSN easily coincide with the notion of self-interested agents in a society. This makes the WSN an excellent testbed for exploring MAS engineering, coordination, and negotiation protocols.

When considered in light of the environment cube mentioned previously by Chrisman, (Chrisman et al. (1991)), it is fair to place the wireless sensor network domain somewhere near the category of “robot’s worst nightmare”. The domain is very unpredictable, situated primarily where there is a complex world state, and where the number of critical choice points to perform effectively is potentially very high. For example, the decision to send long or short range messages has the effect of reducing battery lifetimes; thereby representing a highly critical decision for a node. This suggests that an agent that has combined internal states of model reactivity (for critical choices), and contingency anticipation are appropriate for such situations. However, a single agent would still be at a disadvantage due to the inability to survey the environment adequately, hence knowledge sharing from multiple agents is needed. The WSN, for all these reasons, is adequate for multi-agent solutions that take advantage of self-interested local behaviors on the part of the nodes as well as the organizational management of interactions that can promote global behaviors that fit with designs.

5.2.1 Testing Autonomous WSN Systems

Several important research areas are key topics for wireless sensor networks today, namely localization, routing, information processing, and active sensing (see (Vinyals et al. (2008))). These do not consider the, simpler, but not less significant challenge of testing and debugging strategies in wireless sensor networks. According to Georgoulas and Blow, (Georgoulas and Blow (2006)),

“Applications...developed in this platform (TinyOS) can not be reprogrammed dynamically and the only flexibility is located in changing various parameters prior to the installation...post-development reprogramming is not feasible as the huge deployment of nodes in an environment makes the manual collection, reprogramming and redeployment of them impossible.”

This statement highlights a real deployment issue with the current standard for WSN development and deployment, TinyOS, making it clear that designs and implementations must be accurate before programming. Thus, simulation becomes the primary solution, as it extends to all forms of WSN systems, and is both cheaper and faster to anticipate problems and solutions at design time; long before the network is deployed. Additionally, validation of multiagent systems is a complex affair because the interactions that take place, along with the state of the agent, and the organizations/partnerships that are formed by the system are generally unpredictable at design time, (Jennings (2000)). This means that mechanism design for the local and global behavior of the system needs to be rigorously assessed beforehand as well.

Simulation, in general, and especially for Wireless Sensor Networks (WSN's) is established as a method of evaluation for behaviour analysis (Levis et al. (2003)). In addition to the reasons already mentioned this is also due to the practical aspects of testing systems based on different layers of abstraction. It is simpler to prove results computationally than it may be to do so in formal, theoretical terms while still allowing for viewing a system in a macroscopic way. This means that emergent behaviors can be validated by actual observation, much as the popular “invisible hand” of economic domains. The crux of the matter is that the Autonomous WSN needs to address good testing methods for its agent models and solutions, and simulations represent a key tool for doing so.

Chapter 6

Agent Implementations in Wireless Sensor Networks

The current state of the literature shows that there are a number of agent based wireless sensor network implementations (Fok et al. (2005)), (Zboril and Zboril (2008)), (Rogers et al. (2005)), (Yadgar and Kraus (2006)), etc. This is growing, as the advantages of the autonomous WSN becomes more accepted. Additionally, the methods of programming a WSN have become more mainstream, with the features of the TinyOS 2.0 operating system, (2.x Working Group (2005)), and newer, Java based systems from Sun (Simon et al. (2006)) and Sentilla (Sentilla (2008a)). This section presents the agent implementations and/or simulations for WSN solutions, all of which are very recent developments. In particular it focuses on the classification of weak or strong agent WSN's to be found. What is noticed, however, is the general agreement that strong agents are more difficult due to the hardware (although they are more useful) (Tynan et al. (2005)). Hence more “weak” approaches exist in practice.

6.1 Weak Agents in WSNs

Among the most prominent of these is the work of Jennings, et al, on the GLACSWEB (Rogers et al. (2005)). Here a wireless sensor network is designed for monitoring sub-glacier movements over time. The nodes are situated in an environment that is largely out of reach for the data users and maintenance operators. The degree of change in the environment depends on the movements of the glacier over time. The nodes are equipped with four AA batteries, and are programmed with power consumption preservation as the “mechanism”. In order to save power these nodes negotiate actively,

changing partnerships and forwarding data to the basestation. The key consideration is that, not only do they need to make partners to nodes with a shorter reach to the basestation, but also take into consideration the eventual power loss of the partner node. Further when requested to become a partner these nodes need to decide whether to accept the request, based on power usage expectations, and the expectations of future requests. It is evident that such a problem is complex to design for traditional programming methods. In Jennings's work, the agent model, combined with a negotiation model, and a computational mechanism for the behavior, the nodes are able to function in a way that guarantees self interested behavior for their own power, but also consider the power of other nodes; thus making efficient partnerships that also improve the total network lifetime. Additionally, the solution was shown to be near-optimal, and adaptive to network changes. Amongst the autonomous wireless networks found in the literature, this is an undeniable success story. The paper does not discuss the inner workings of the agents used, but does demonstrate that mechanisms may be designed for simple agents.

In a similar work, Kho and Jennings, have also proposed a method for flood monitoring known as FloodNet, (Kho et al. (2007)). This uses a similar utility style approach to "agentify" the network, along with mechanism designs. The objective is to perform decentralized adaptive sampling as a power saving technique. In particular, the problem is that the processor consumes too much power, along with the power required for sensing water-depth. The solution involves using agents to vary the sampling rates of the sensors in the network based on the value of the information being gathered.

In the work of Ruiz, et al, (Ruiz et al. (2005)), agents are proposed for the MANNA agent management framework, applied to fire risk/fault management situations. Their approach makes use of the PONDER language for specifying agent policies, and Voronoi diagram calculations, for focusing on non-redundant nodes in sections of the network. The approach is based on assigning a fire-risk level to messages, and involve a clusterhead routing approach where agents forward information to a manager node (clusterhead) which then relays that information to the basestation; a common sense and send problem. The PONDER language used, allows for specifying the policy information for each agent as a finite state machine.

Finally, two similar approaches, that of Agilla, (Fok et al. (2005)), and In-Motes, (Georgoulas and Blow (2006)), show the use of middleware technology for mobile agents in wireless sensor networks. These approaches act as virtual machines for

agents that are passing through network like data messages. The advantage of the approach is seen in the ability to run multiple agents on a single node, and also to have agents that can migrate to points of interest in the network in order to do processing. This approach also works well for ad-hoc queries of the network, as custom gatherer agents may be used by a user. However, such agents are considered very weak by BDI standards, since neither the virtual machine nor the agents have BDI style-reasoners.

6.2 Strong Agents in WSNs

Jedermann, et al, (Reiner Jedermann (2006)), have applied an agent WSN to the domain of logistics applications, used primarily for monitoring goods during shipping. The sensors monitored temperature, atmosphere, and acceleration against the lifetime of the good. This approach used an agent host that was embedded in the vehicle as a gateway to the WSN and did not have the agents run on the nodes because of resource limitations. In particular, this example made use of the JADE platform.

Similarly, Stathis, et al, (Stathis et al. (2007)) have used an agent based approach for data sharing between wired and wireless networks. His approach uses agents as a gateway to the wireless network nodes as well, and hence the WSN nodes are not agent based, for the same reasons as Jedermann. The approach involved, however used the PROSOCS platform for KGP agents (Bracciali et al. (2006)). In Sandhu, et al, (J.S. Sandhu (2004)), proposed an agent approach that made use of the subsumption architecture of Brooks,(Brooks (1991)), from behavior control systems, for commercial lighting control. The approach centered around creating intelligent lights, but had a WSN as the base platform. As before the authors noted that simple agents are needed because of the hardware limitations.

Tynan, et al, (Tynan et al. (2005)), have noted that the strong agent for WSN's is an open problem, due to its hardware constraints, and deployment and testing difficulties, especially given the lack of user interfaces. With this in mind the work presented a desktop methodology for mapping agents to nodes and using agents that share the desktop basestation for debugging perceived information from their relevant sensors in the field in a distributed context. The work implemented three types of agents, for one to one mapping, one to n mapping, and n to one mapping, in order to verify the agent interaction designs before placing on the actual hardware. The authors have used the strong agent platform, AgentFactory, (Collier (2002)), for commitment rules and policies, although they note that weak agency is preferred.

6.3 Simulating Agents for WSNs

In the literature it is common practice to simulate for wireless sensor network deployments. Of all the simulations found, several are worth mentioning because of their similarity to the SAMSON simulator, as proposed. The similarities are based on those simulators which embed an agent component, and a hardware model into a common framework, including a form of environment modeling.

The first is the work of Kho and Jennings, (Kho et al. (2007)), for adaptive sampling within the FloodNet project, as mentioned previously. Here the authors present a simulation tool known as DC-WSNS, in order to evaluate the algorithms. DC-WSNS provides modules for nodes, batteries, sensing capabilities, network stack (adopted from an earlier work), and an environment model to represent cloud coverage for solar panels). The does not, however, model the wireless communication channel, or transmission details such as messaging, and propagation. Again, in this approach weak agency is used. The interface allows a top-down view of the network.

Another similar approach is that of Ruiz, et al, (Ruiz et al. (2005)), using the MANNA project to model fire risk monitoring in a WSN. The work adopts the commonly used Network Simulator 2 (NS-2), (Riley (2003)), and its adaptor for Wireless Sensor Networking. NS-2 provides a simulation environment that maintains common MAC protocols, a transmission layer, network layer, a broadcast mechanism, and an application layer (for implementing MANNA policies). Energy consumption, and data delivery rates are the main metrics gained by the simulation. As mentioned this also is a weak agent approach.

In the work of Jennings, another simulation is presented for the domain of the GLACSWEB project (Rogers et al. (2005)). Here, the authors show simulation results that depict a network topology that is dynamic, based on routing protocols that are determined by mechanism design. The simulator details are not discussed, however the results show simulation of network formation topologies and displays the network as nodes are dying off, and network restructuring takes place, allowing for maintaining the network coverage size. This approach is primarily policy driven, and uses weak agency.

In Ma, et al, (Ma et al. (2006)), the authors briefly discuss exploring the use of BDI agents in WSN's, and show the result of a simple application for power management. In this work, as the agents battery consumption level increases, the agent adjusts the radio state, and lowers sampling rates as a simple approach to save energy. The

simulation framework uses the AgentFactory Micro Edition (AFME) combined with the common J-Sim WSN simulator.

Finally, in a recent work, Zboril, (Zboril and Zboril (2008)), presents work on mobile agents for the CrossBow platform (Crossbow (2008)), designing a BDI like framework, and middleware, known as MOTE. Using a language such as ALLL (Agent Low-Level Language) and a specification of the CrossBow hardware they present a simulator known as T-Mass. T-Mass is able to simulate agent behaviours for the platform, but does not inherently focus on the environment model.

6.4 Discussion

As the above works show, agents in wireless sensor networks is scarce, Tynan et al. (2005)), primarily for two reasons. The first is that the field of applying agents to make an autonomous WSN is very young and a great deal of solid research remains to be done. The approaches shown are all at the forefront of such work, but there is much more to be done in order to reach the autonomic vision of Kephart (Kephart and Chess (2003)). The second reason is that the current WSN hardware presents constraints that are significant obstacles to multi-agent development. Having a constrained hardware device also means that it becomes difficult to program, test, and debug. This is not insurmountable, and the computing architectures will eventually allow for more freedom in designs. Testing, however will require more tools targeted directly to helping agent researchers of WSN's, and this will become more evident as the field continues to grow.

The above works also show that the state of the art involves mainly weaker notions of agency, as deliberative agents are only just becoming feasible. Furthermore, simulations appear to be adapting existing WSN simulator tools for merging with agent frameworks as a test environment. Only AgentFactory, (Ma et al. (2006)), and T-Mass, (Zboril and Zboril (2008)), offer strong agent simulation environments, similar to the one proposed in this paper.

Chapter 7

Proposed Solution

A complete solution for autonomous wireless sensor network simulation would require models of the environment, and hardware that are as accurate as possible, (Levis et al. (2003)), flexible enough for many WSN scenarios. Such a solution would consist of “model components”, namely for power consumption, radio propagation, network management, packet management, radio channel model, and environment modeling. Additional utility would also be added through the possibility of adding an interpreter based on the TinyOS platform, or SystemC, or Java, allowing for easy translation of agent code into runnable WSN code. Further the tool would be customizable, and have a clean user interface, and be readily available (perhaps as an open-source project).

7.1 Objectives

In this work, a partial solution is presented as a first step towards BDI agency in a WSN, based on the Agentspeak language, (Rao (1996)). Although it has been agreed on in the literature that the existing WSN hardware is not yet sufficient for a truly deliberative agent platform, this is expected to change quickly. Already, the TMote Sky, (Sentilla (2008b)), sensors have outpaced the hardware and software used in some of the experiments discussed previously (Beutel (2006)). For this, hardware, radio propagation, and environment models are proposed. The second key component of the work is a look at practical scenarios for placing agents in WSN's in order to gauge benefits, or costs.

In order to build this first step solution, it is necessary to identify a framework for agent modeling, derive a model of the TMote Sky hardware based on spec sheets and other tools, perform an implementation of the tool, and a display mechanism, and

finally to conduct tests and demos of the work. These primary objectives are listed below:

1. Develop an autonomous WSN simulation environment that is based on strong agents. This simulation environment should be flexible enough for conducting future work without redesigning the underlying models. Additionally it should be accurate enough for industry analysis of certain testing and deployment cases (as presented by ArsLogica, (ArsLogica (2008b))). This also entails being based on current hardware.
2. To study the benefits of agents in WSN's, in particular to note the characteristics of agency and the fit with WSN paradigms. This is useful as it allows for stakeholders to gain an appreciation for the autonomous techniques that can be applied in WSN projects.
3. To design agents for common sense and send scenarios, in particular agents that can perform low power management, routing, and request acknowledge basic uses.
4. To evaluate agent simulation platforms, using the criteria for the project, and to select a tool from among these, for integration with the simulator.

7.1.1 Assessment Focus

While there are many different ways to build the simulation framework, in this paper it is to be considered according to its agency, hardware accuracy level, flexibility for different scenarios and future works/industry works, its environment model, usefulness and difference from other agent WSN simulators, as well as whether or not the study has clarified the fit of agents in WSN's in terms of costs, benefits, or implementation concerns. Finally, the testcases should demonstrate realistic behavior.

7.2 Methodologies

This section highlights the major decisions for the design and construction of the simulator. It consists of selecting an appropriate agent platform, methodology for agent designs, and the choice of hardware modeler.

Criteria for Deciding on a Simulation Environment					
Criteria	REPAST	AgentScape	TAEM	Mason	JASON
BDI Support Available	No.	No	No	No	Yes
Inbuilt Agent Reasoning Engine	No.	No	No	No	Yes
Logical constructs supported	No.	No	No	No	Yes
Theoretical background published	Yes	Yes	Yes	Yes	Yes
Agent Constructs Supported	Yes	Yes	Yes	Yes	Yes
Agent Communications Medium Available	Yes	Yes	Yes	Yes	Yes
Distributed MAS Support Available	No.	Yes	No	No	Yes
Output Visualizations Available	Yes	Yes	Unknown	Yes	Yes
Java based/Object-oriented	Yes	Yes	Yes	Yes	Yes
Easily Extensible	Yes	No	No	No	Yes
Support Forums Available	Yes	Yes	Unknown	Yes	Yes
Adequate Documentation/Tutorials Available	Yes	Yes	Yes	Yes	Yes

Table 7.1: Several simulation platforms considered at design time.

7.2.1 Agent Platform Selection

In deciding on an appropriate agent platform, strong agency was the primary concern, along with having a flexible and logical agent programming language that was both well published, and well received by the agent community. Additionally, the platform needed to be easy to adapt for new components and new integration modules. The functionality for robust multi-agent system design and communication, along with visualization capabilities was also a key factor. After reviewing selected tools in the media JASON/Agentspeak was selected as the agent design platform. Table 7.1 shows the platforms considered when selecting the agent framework.

7.2.2 Agent Methodology Selection

The Prometheus methodology has been selected for the design of agents in SAMSON, primarily as the methodology is a good fit with the AgentSpeak notions of plans, percepts, and goals, while still being relatively easy to use. Prometheus is a “detailed and complete” process for designing multi-agent systems (Padgham and Winikoff (2002)) that is suited for those who are just starting to develop BDI agents. The methodology consists of a sequence of deliverable diagrams that are generic enough to design a number of MAS systems. In particular, there are three phases of abstraction that are then iteratively revised. The first is the system specification phase, where users

design the use case scenarios and the analysis overview of the problem domain, including percepts, actions, roles, agents, and data items. Second is the architectural design phase, which highlights the protocols, agents, and provides a system overview. Finally there is a detailed design phase where agents are expanded based on capability, role, and functionality. Within this project a design tool has been used to facilitate the diagramming (see PDT (Padgham et al. (2005))).

In (Padgham and Winikoff (2002)), the authors provide a strong justification as to why this methodology is a good fit for BDI agent design. First is that it supports BDI agents, and has been targeted to that area. The second is that it provides a detailed process that is easy to use for both beginners and practiced designers. Additionally, the guided hierarchical process is able to be crosschecked automatically for errors using a tool such as PDT. Finally, the tool has been shown in papers such as (Bordini et al. (2006)), and (Bordini et al. (2007)) when designing reasonably complex systems.

7.2.3 Simulation Framework Selection

As mentioned in the SENS paper, Sundresh et al. (2004)), there are roughly three classes of simulation frameworks for wireless sensor networks, not including those that are built specifically for a particular application. These are simulators for wireless protocol stack monitoring, discrete event simulators, and application oriented simulators. The first category is typically overkill for many applications; the second is closely linked to the operating system (like TOSSIM, Levis et al. (2003))) and are able to run actual source code for actual hardware, which is also overkill for testing application ideas such as agency; the third class involves modules, and reusable components and layers that are adaptable for each application use. The latter is the class of simulator that is used in this project.

The decision about the model environment involved looking at existing options to decide whether to build. The two main options available based on this research were J-SIM, (Sobeih et al. (2005)), an open-source Java-based wireless sensor development environment, and the SENS simulator. The tool is a cross between the wireless protocol stack approach and the application-oriented approach, and provides a power model and a sensor function model, along with network layers and sensor layers along with a host of other components. Although it provides some useful insight into the workings of a WSN simulator, it is rejected because of the extra complexity.

The SENS framework is a closer fit with the objectives of the project. The soft-

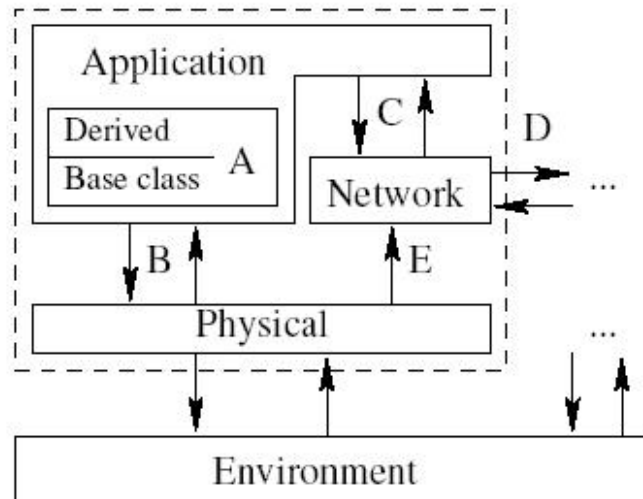


Figure 7.1: The SENS simulator design, from (Sundresh et al. (2004)).

ware architecture design is flexible, and modular, allowing for customizations to be made using an object oriented plug-in approach. Such an approach allows for users to adapt the framework, or extend it to general purpose applications. Additionally, the design is platform-independent, since new hardware profiles can be added easily. The framework consists of class objects for the environment, and nodes; nodes consist of an application class (for node programming constructs), a network representation class, and a physical (hardware) representation class. The tool also has visualization properties that use a Tileworld, (Pollack and Ringuette (1990b)), and also models radio propagation and basic sensor actions. The primary problem with the SENS framework, however, is that it is not java-based, and hence is not a direct fit with the JASON interpreter. This has led to the approach taken in the SAMSON project, which is to construct a “SENS-style simulator” in Java for the hardware simulation model. The difference, of course, is that in SAMSON the application layer is entirely driven by JASON/Agentspeak (see figure 7.1).

7.3 TMote Sky Hardware Platform

The approach taken in this project is to build a new simulation architecture, hence it is important to use detailed components of the hardware used in current projects by industry (as represented by ArsLogica), namely the TMote Sky hardware, as previously mentioned. According to the TMote Sky spec sheet from Sentilla (Sentilla (2008b)) the

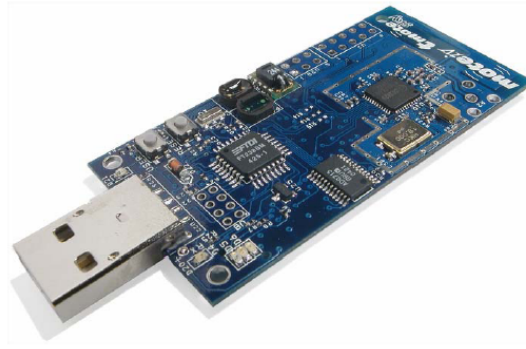


Figure 7.2: The TMote Sky sensor used as the hardware model; image from (Sentilla (2008b)).

Tmote is an “ultra low power wireless module for use in sensor networks, monitoring applications, and rapid application prototyping.” The device is among the most popular sensing units in the industry currently, and have been deployed in many successful implementations. The primary features are its industry standard sensing technology, low power functionality, performance measurements, expansion capabilities, and its improved wireless transmitter and receiver technology.

7.3.1 Key Features and Components

The device, consists of a microprocessor, radio, antenna, external flash, sensors for humidity, temperature, light, and internal voltage (with additional connectors for custom sensors). Some of the relevant features of the TMote Sky are highlighted below, as shown in the datasheet:

- 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver
- Interoperability with other IEEE 802.15.4 devices
- 8MHz Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash)
- Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller
- Integrated onboard antenna with 50m range indoors / 125m range outdoors
- Integrated Humidity, Temperature, and Light sensors
- Ultra low current consumption

7.3.2 TMote Sky vs Other Sensors

In a comparison paper by Beutel, Beutel (2006)), the TMote is among the best of industry standards for low power sensor nodes. The device is a 16bit architecture, running at 8MHZ, having 48KB of program memory on the chip, 10 KB of data memory, and a large storage memory of 1024KB. This gives it among the highest of storage (although the chip memory size is relatively small). As mentioned, the device also has 5+ on board sensors, where most other devices have only two.

In terms of its radio characteristics the TMote transmits at a high frequency (2.4GHZ) standard, with a reasonably average data rate of 250kbps. The radio is able to be controlled at various power levels from 0dBm (max) to -24dBm (min) (others can transmit at a minimum of -30dBm). Its radio has the fastest wakeup time (1ms) and can sense received signals of -94 dBm or higher (reasonably standard), with an outdoor range of 125m. It has 16 channels of frequencies available (which is more than most). See (Beutel (2006))) for more details on the TMote Sky in comparison with other state of the art hardware.

Chapter 8

SAMSON Simulator Designs

In this chapter the simulation environment for strong agent WSN's is presented in detail. The requirements are to provide strong agency, heterogeneity, scalability, reliable models, and easy to use and program interfaces. The design criteria for SAMSON is listed below:

- Agent-based (BDI) controller and agent definition language.
- Support for heterogeneous agents with different objectives and goals.
- Scalable for testing variable sized agent networks.
- One to one mapping of agents to nodes.
- Extensible model of hardware components, starting with power and radio.
- Model of the environment (should include node locations, and obstacles of various thickness).
- Java-based model for object-oriented design benefits.
- Ease of programming agent logic.
- User interface allowing for moving nodes, and obstacles.
- Use interface allowing for changing hardware parameters easily.
- Visualizing charts of various properties (such as power consumption).

8.1 SAMSON Architecture

The overview of the design is shown in the figure below. The details are discussed in the following sections:

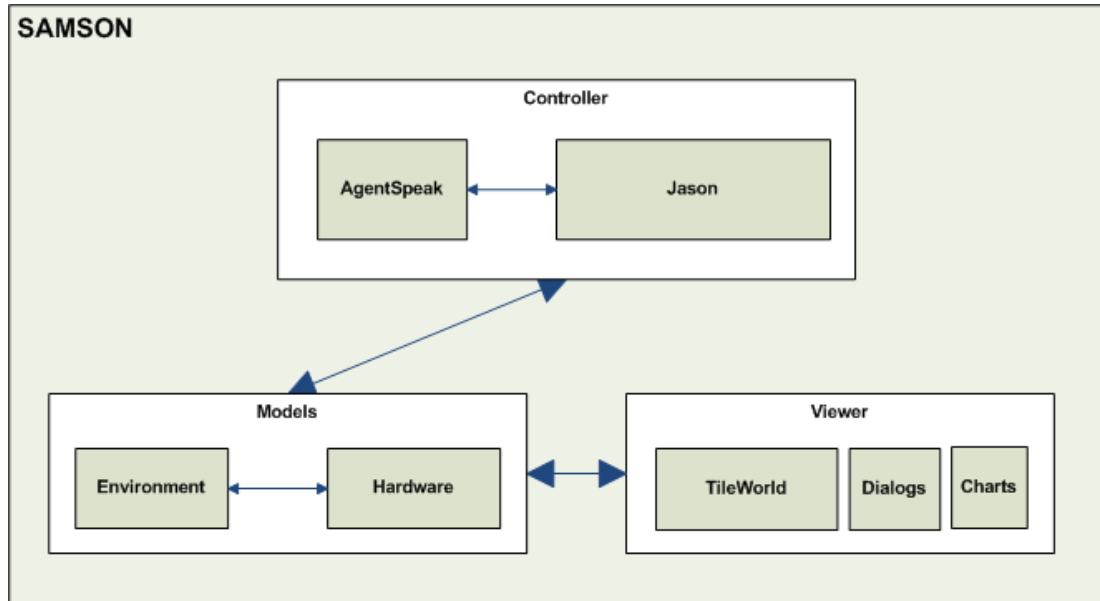


Figure 8.1: The simulation architecture, following the MVC pattern.

8.1.1 JASON

JASON is designed as an interpreter for the AgentSpeak language, as developed by (Rao (1996)), and revised by Bordini and Hubner, (Bordini et al. (2007)). The language of AgentSpeak and the abstract interpreter are based on the procedural reasoning system, as mentioned in the earlier chapters of this report. JASON agents are able to be executed in a local, multithreaded context, or on distributed system architectures via its SACI intercommunication protocol. JASON adds a java-based architecture and interface for programming simulation environments for agents. Additionally The designers of the JASON framework have provided for communication and interaction specifications according to a “speech-act” mechanism for agent communication. Further they provide basic internal actions pre-set for common programming tasks. Users of the JASON framework are able to design agents following the AgentSpeak logic and language, which is regarded as a standard for BDI reasoning, as well as interface between agents and custom environments.

Each JASON agent contains several key BDI reasoning components, a belief base,

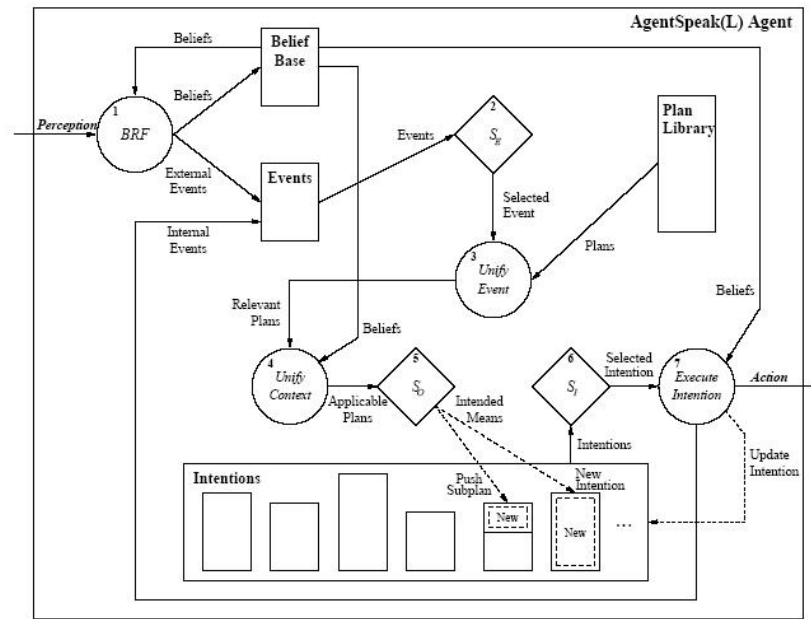


Figure 8.2: The JASON/AgentSpeak interpreter cycle for BDI, from (Bordini et al. (2007)).

an event library, a plan library, and a library of current actions (intentions). As mentioned before, the procedural reasoning system is one in which agents receive input from the environment in the form of percepts, or beliefs. These are added to a belief base (which is a stack structure) for future reasoning. Beliefs in the stack are then used to decide future actions based on current actions and existing plan conditions. Upon receiving new beliefs the interpretation cycle proceeds sequentially through stages of belief revision, event selection, plan selection, and intention execution. This provides the agent with continuous reactive responses, as well as flexibility of executing new intentions and older ones, following plans and policies specified by a user. See figure 8.2 for a look at the interpretation cycle.

This interpretation presents a “perceive-act” cycle supported by the JASON environment architecture; a means of continuously providing perceptual information (percepts) to agents, as well as communicating effects made by agents on the environment (as done through the agent’s actions/effectors). In the case of SAMSON this environment is constructed in java according to the common MVC (model-view-controller) design pattern, with interaction between the JASON interpreter and the architecture handled by the controller.

8.1.2 Agentspeak

The Agentspeak language (Rao (1996)) is designed to extend logical principles to agent programming. At its highest level the language consists of a set of initial beliefs, plans, and target events. Initial beliefs about the world are stored in a growing belief base, and represent some fact about the agent's mental state regarding the world (for example "+hungry" adds a belief "hungry(me)[source=self]" to the base, indicating that the agent has the belief from itself that "hungry(me)" is true. Plans are preset actions to be taken in the world when certain specific conditions hold true. Each plan consists of a triggering condition, a context, and a sequence of actions, goals, or belief updates. When an agent decides to use a plan this is represented as either an achievement goal (indicating that the agent should make steps to achieve a state of the world where the goal succeeds (Bordini et al. (2007))) or a test goal, for validating assumptions about its beliefs. Achievement goals (like "!getFood") and test goals (like "?hungry(me)") both result in checking plans and belief conditions. Additionally, Agentspeak programs may involve some triggering events when a belief, or a goal, is added or removed from the base. It must be mentioned that goals represent plans that the agent wants to execute, while the plan itself is a recipe for action. A simple Agentspeak program is seen below for ordering pizza, as an example; the complete language is discussed in (Bordini et al. (2007)).

8.2 Hardware Models

As mentioned, the model is a SENS-style simulator, (Sundresh et al. (2004)), with customizable components that are built on an Application layer, a node layer consisting of a network model, and a physical model. The "Single-node Architecture" from (Karl and Willig (2005)) is also considered in the design, along with the TMote Sky datasheet, to narrow the important properties and parameters for the hardware model (which is represented by the Physical and Network components of the SensorNode class in figure 8.3). In particular, this Single-node architecture (Karl and Willig (2005)) refers to having the following components of a basic sensor node:

- Controller - for data processing, and code execution.
- Memory - for data storage.
- Sensors and actuators - for interfacing with the environment, movement, sensing.

Program 1 A simple Agentspeak example.

```
//Initial belief
+hungry(me).

//Triggers for what to do when beliefs are added to the base
// for being hungry and for answering the doorbell
+hungry(me) : true
    !getFood

+doorbellRings : true
<- !handleDelivery;
    eat.

//Plans for getting food, ordering pizza, and handling delivery

+!getFood : hungry(me)
<- !orderPizza(444-4444).

+!orderPizza(Number) : true
<- call(Number); //internal actions in Java
    placeOrder; //assumes an interrupt percept when arrives
    +orderPlaced.

+!handleDelivery : true
<- transaction(Pizzaguy, Me).
```

- Communication device - for network formation through message passing over a wireless medium.
- Power Supply - for operating the hardware above, may be battery or solar based or both.

In SAMSON, the model covers Controller, Sensors and actuators, Communication device, and Power Supply. Memory management is not handled at present, although it could be incorporated into future iterations of the simulator. The class diagram for the main components is seen in figure 8.3 below. The detailed hardware descriptions of

the TMote sky are seen in the following sections, and are realized in the SensorNode class (and subclasses).

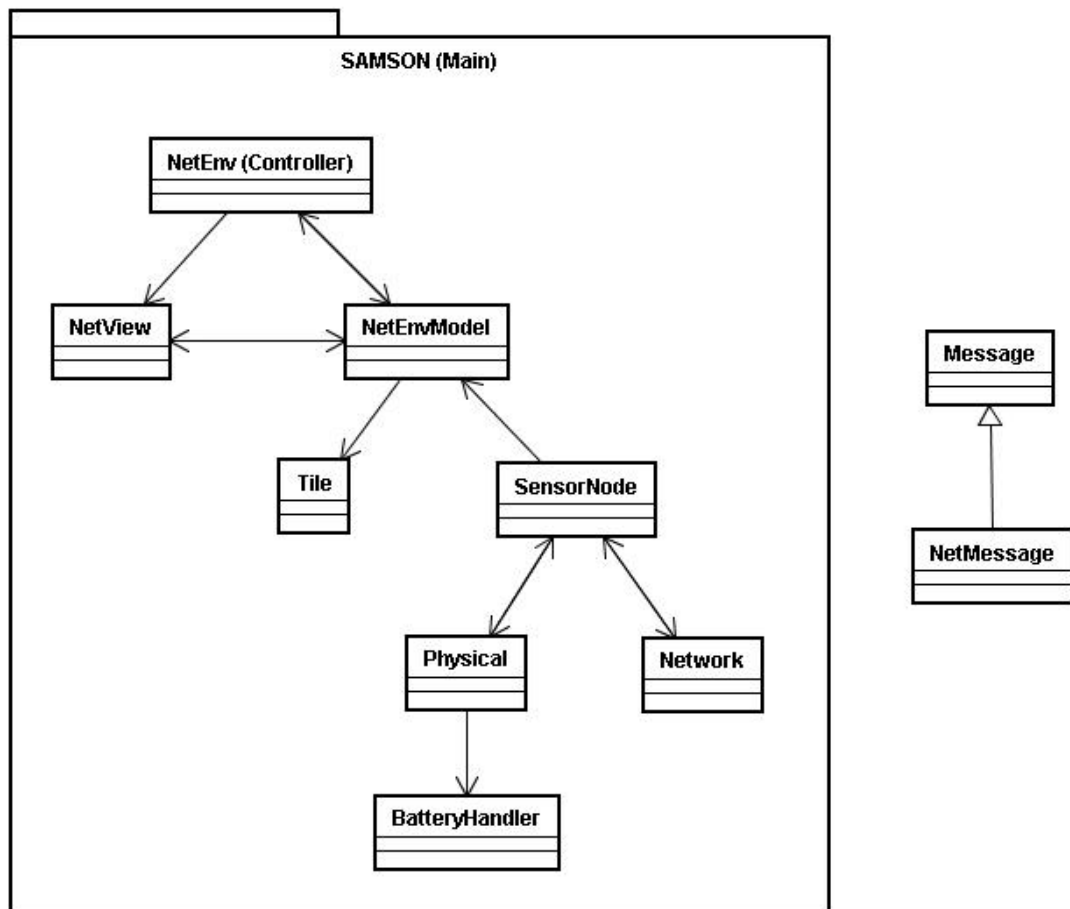


Figure 8.3: Class hierarchy for SAMSON, inspired by SENS (Sundresh et al. (2004)).

8.2.1 Micro Controller Unit (MCU)

This component is the central processing unit (CPU) of the node, and handles the coordination of everything hardware and software related. In our design the controller is abstract, and represented as the JASON program which executes the directives of the agent. Low level details and functions of the controller are not taken into consideration in this simulation, with the exception of power consumption based on MCU states. In the agent based scenario the use of the JASON engine to control the agent is the target, and hence this works perfectly as a model controller, called by the JASON agent on startup. The Controller states are seen in the table in figure 8.1: Active is when the CPU is on, all parts full power. Idle is when the CPU is on, clocks stopped. Standby

MCU Models:	Value	Units
Active Range		
Min Power	1.8	V
Max Power	3.6	V
Consumption		
Active Mode	500	Mu Amps
Idle Mode	5.1	Mu Amps
Sleep Mode	2.6	Mu Amps

Table 8.1: MCU States according to (Sentilla (2008b))

mode (Sleep) only has the interrupt timer running.

8.2.2 Sensors and actuators

While there are many sensor devices available for measuring the environment these are classified into three categories based on direction and activity, according to (Karl and Willig (2005)):

- Passive, omnidirectional sensors: These measure the environment without any required activity, simple sampling takes place. Such sensors may even be self-powered, or powered by the environment itself (eg. solar, or vibrational power). The assumption of self/environment powered sensors is also adopted. The sensors used in this work are seen in the table in figure 8.2.
- Passive, narrow-beam sensors: A directional sensor, requiring rotation to a certain direction (like a camera sensor).
- Active sensors: These probe the environment by sending out signals and gathering feedback (such as radar).

The sensors in this scenario are all of the passive, omnidirectional type. No actuators for moving sensors, etc, are considered as part of the model. In the simulator, the degree and range of the sensor is entered as a parameter of a simple environment model (see discussion of the TileWorld later in this chapter).

Sensor Model:	Value	Units	Comments
<i>Humidity Sensor</i>			
Min Range	0	%RH	Relative Humidity
Max Range	100	%RH	
<i>Temperature Sensor</i>			
Min Range	-40	C	
Max Range	123.8	C	
<i>Light Sensor PAR</i>			
Min Range	0.1	AW	300 NM in Wavelength
Max Range	0.3	AW	800 NM in Wavelength
<i>Light Sensor TSR</i>			
Min Range	0.1	AW	300 NM in Wavelength
Max Range	0.6	AW	1100 NM in Wavelength
<i>Internal Voltage</i>			
Min Range	0	V	
Max Range	3.6	V	

Table 8.2: Sensors included in the model of the TMote Sky unit (Sentilla (2008b)).

8.2.3 Radio Transceiver

Radio-frequency-based communication is used in this model. The transceiver (transmitter and receiver) is modeled primarily during send and receive actions according to the parameters of the TMote, and formulas for wave based propagation found in (Karl and Willig (2005)). Transmission layers are abstracted, so when a message is sent, it is assumed that all packets arrive safely before data is transferred to the controller. Also noted is that transmission and receiving does not take place at the same time in the model, as in reality this is a form of noise interference. The transceiver is able to go into several states which are factored into power costs, in addition to startup and shutdown costs (Karl and Willig (2005)), (Sentilla (2008b)). These states are:

- Transmit - transmitter active, antenna on (emitting energy).
- Receive - receiver active only (and receiving).
- Idle/Sleep - transmitter inactive, receiver inactive (or low powered).
- TXRXSwitch - When switching from transmit mode to receive mode.
- RXTXSwitch - When switching from receive to transmit mode.

The transmission and receiver characteristics are seen in the table in figure 8.3 in more detail. The Range of the transmitter is a value that is calculated based on:

- Transmission power (which depends on modes).
- Antenna characteristics (assumed to be those of the receiving node).
- Environment attenuation (which is set to each tile in the grid).

Channels are not considered, although it may be worthwhile to include channels for cases where clusters are used. Receiver sensitivity is the minimum signal power needed to achieve a read of the signal, and is modeled as a value based on battery, and radio range. RSSI (Received Signal Strength Indicator) of an incoming signal is also received a part of the model. Table 8.3 shows the radio factors in more detail.

8.2.4 Power Supply and Consumption

Battery stats are obtained from the TMote spec sheet, but the typical AA battery ranges from 3.0 to 1.78V roughly. According to measurements (from (Wikipedia (2008))) a

Radio Modelling:	Value	Unit	Notes
Message Transmit Time	200	Milliseconds	Modifiable from 20ms upwards
Dead Nodes cannot receive			
Power consumes on receiving	65	mWatts	
RX startup power	27.5	mWatts	Happens on call for powerup
RX shutdown power	5	mWatts	Happens on call for powerdown
TXX Switch power	65	mWatts	Happens after message sending
RXTX Switch power	70	mWatts	Happens before message sending
Transmit power	60	mWatts	During Transmission
Active Range			
Min Power	2.1	V	
Max Power	3.6	V	
Min Frequency Range	2400	MHZ	
Max Frequency Range	2483.5	MHZ	
Indoor Field Distance	50	Meters	
Outdoor Field Distance	125	Meters	
Bit Rate	250	kbps	
Peak Antenna Gain	-5	dBi	
Antenna Wavelength	5.04	inches	antenna length*4
RFWavelength	0.00	Meters	=speedOfLight/RFFrequencyHz
Losses through Circuitry	≥ 1		
Consumption based on State			
Active	365	Mu Amps	
Idle	20	Mu Amps	
Sleep/Off	1	Mu Amps	
Transmitter (Tx)			
Output Power(dBM)	Consumption		
0	17.4	M Amps	
-1	16.5	M Amps	
-3	15.2	M Amps	
-5	13.9	M Amps	
-7	12.5	M Amps	
-10	11.2	M Amps	
-15	9.9	M Amps	
-25	8.5	M Amps	
Receiver (Rx)			
Receiver Sensitivity	-94	dBm	
Consumption on Rec	19.7	M Amps	

Table 8.3: Radio based parameters that are used in SAMSON, from (Sentilla (2008b)).

Power Consumption States	Value	Units	MCU State	Radio State
Min Voltage(2.1 in specs)	1.8	V		
Max Voltage	3.6	V		
State1	21.8	M Amps	Mcu On	Radio Rx
State2	19.5	M Amps	Mcu On	Radio Tx
State3	1800	Mu Amps	Mcu On	Radio Off
State4	54.5	Mu Amps	Mcu Idle	Radio Off
State5	5.1	Mu Amps	Mcu Standby	Radio Off
State6	70	mWatts	Mcu On	RXTX Switch power
State7	63	mWatts	Mcu On	TXRX Switch power
State8	65	mWatts	Mcu On	RXReceiving
State9	txPowerMAmps	M Amps	Mcu On	TXTransmitting

Table 8.4: Overall System states, and power consumption levels taken from TMote nominal behaviors (Sentilla (2008b)).

AA battery normally has 2850 milliamperes per hour of capacity. When the power supply is electric (for instance, when the node is attached to a PC via USB) the rate detected by the internal voltage sensor is 3.6V. The battery level factors into the hardware availability and output power during usage. Main power consumption is due to the controller state, the radio, the memory read/writes, and the use of sensors (in some cases). Voltage is reduced as battery capacity falls according to the defines of the normal TMote hardware.

The power consumption metrics used in the simulator are shown in table 8.4:

8.3 Environment Model

The model of the environment is important to the SAMSON simulation, in particular this involves the placement of sensors, obstacles, and setting location based parameters for the sensor devices. The environment is grid-based, where each section of the grid contains a Tile object which stores sensor parameters, some of which affect wave based propagation, see figure 8.5. The obstacles in the grid may have attenuation factors similar to measurements of path loss coefficients as seen in (Karl and Willig (2005)). This allows for obtaining propagation based distances. *In SAMSON, it must also be mentioned that each grid cell represents units of approximately 1 meter.*

Environment	Path Loss
	Exponent
Obstacle	Gamma
Floor	2
Thin Obstacle	3
Thick Obstacle	5

Table 8.5: Attenuation coefficients for obstacles of different thicknesses, according to (Karl and Willig (2005))

8.3.1 Wave-based Propagation

As mentioned in (Sundresh et al. (2004)), (Karl and Willig (2005)), the effect of wave based propagation and noise, should not be ignored. Depending on the environment signals may be reflected, diffracted, scattered, faded, or even mixed with other signals on the same channel. These are common considerations in WSN design, and are taken into consideration in the model. In particular the model designed focuses on antenna efficiency and radiated power, but this section aims to define the parameters used to gauge such forces more accurately. See chapter four of (Karl and Willig (2005)) for more; the model maintains parameters for:

- Path loss and attenuation: As the signal transfers from point a to point b it loses power according to set parameters. The *Friis free-space equation* is used, according to definition 4.2 and 4.3 of the *log-distance path loss model* found in (Karl and Willig (2005)).

$$PL(d)[dB] = PL(d_0)[dB] + 10\gamma \log_{10}\left(\frac{d_0}{d}\right) \quad (8.1)$$

Where γ is the path loss exponent (ranging from 2, for open spaces to 5 for thick obstacles), d is the distance between the transmitter and receiver (in meters), d_0 is the known far field distance of the hardware. $PL(d_0)[dB]$ is known or computed as:

$$PL(d_0)[dB] = \frac{P_{tx}}{P_{rcvd}(d_0)} \quad (8.2)$$

$P_{rcvd}(d)$ is:

$$\frac{P_{tx} \cdot G_t \cdot G_r \cdot \lambda^2}{(4\pi)^2 \cdot d_0^2 \cdot L} \cdot \left(\frac{d_0}{d}\right) \quad (8.3)$$

Where λ is the wavelength of the monopole antenna on the TMote Sky, P_{tx} is the transmission power in decibels, G_t, G_r is the antenna gain for transmitter and receiver (in decibels), and $L \geq 1$ represents the losses through circuitry. See Figure 8.3. Attenuation, is factored into the above model through the path loss exponent, allowing for a measurement of distance propagation that reflects a general environment, (Karl and Willig (2005)).

Reflection and scattering calculations have not been factored into the current model.

8.4 View Component

The Viewing component, as shown in figure 8.4 handles the graphical interface for the end user of the tool, providing:

- TileWorld for showing the environment map. Allows for changing node locations, obstacles, etc
- Graph outputs for showing battery levels, etc.
- Options for saving maps, loading maps, and drawing new maps.
- Standard Jason Console, used to debug agents in the Jason mind inspector tool.

8.4.1 Tileworld

The Tileworld is a common approach for designing simulation environments that allow for map-views of a field for controlling placements, (Pollack and Ringuette (1990b)). The approach is adapted to the sensor network to represent a general-purpose world for creating and loading maps with different layouts, according to the tile settings. The unit of measure corresponds to 1 meter per cell. In some cases scaling factors are used so the representation may be considered to be $1/scalefactor$ meters. A snapshot of the SAMSON Grid is below.

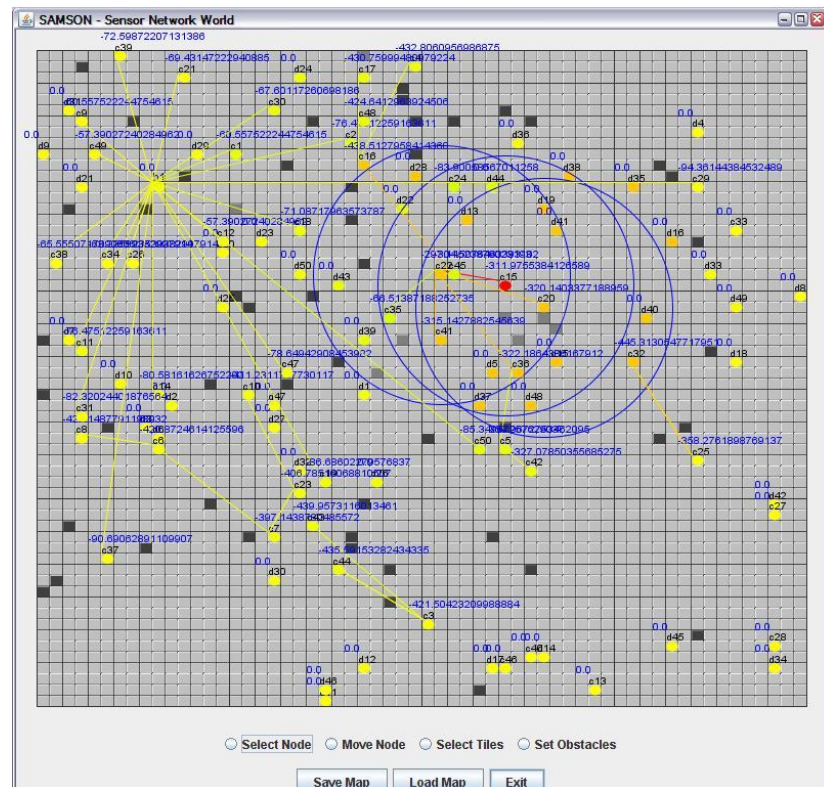


Figure 8.4: Sensor Network World view in SAMSON (shows nodes and random obstacles).

8.4.2 Dialogs

Additionally, SAMSON has settings for changing sensor node parameters and environment (Tile) parameters (see the figure for this representation). Charts are customizable using the JFreeCharts java plugin. Power consumption levels in Watts, and Voltage level over time are the main metrics implemented. These components may be seen in figure 8.5.

8.5 Controller

As mentioned, the control is handled by the JASON interpreter and is programmed in AgentSpeak. However, the interface between Agentspeak and the underlying architecture is handled by the NetEnvironment class in figure 8.3, which performs the initialization and other important functionality for executing the actions for a particular agent thread, and for updating percepts when they are required. In the model percepts are updated continually as part of the control loop of the interpreter, but also

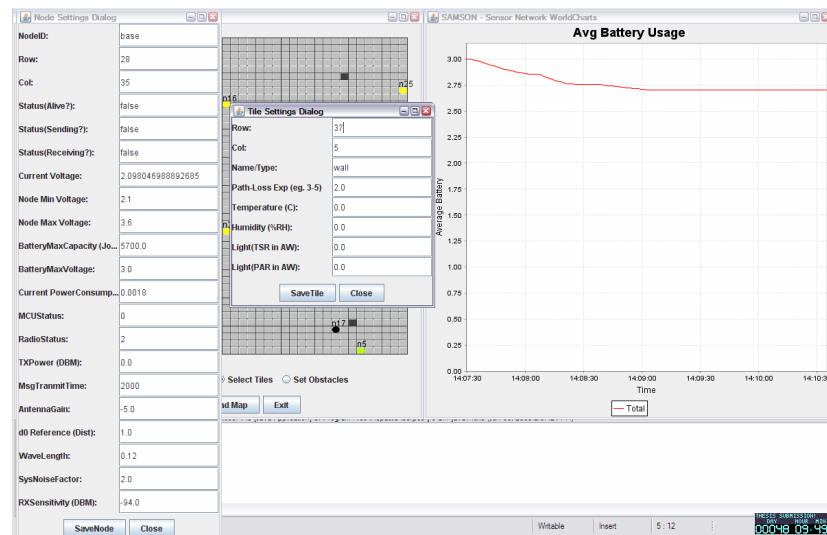


Figure 8.5: Dialogs and charts in SAMSON for Tile and Node properties, and Average power.

when a message is received by the Network component, or a sensor has retrieved data. This information is then acted upon by the agent, which is able to perform several preset, but easily customizable, actions. The main actions that are made available to agents as internal actions in JASON are seen in the figure 8.6 below. In particular, nodes may send messages, sense the environment, set hardware power and state, and conduct special actions for saving, and custom computation messages. Together, they allow an agent to control a model of a sensor, and designers of multi-agent systems to compose interaction protocols for different cases.

Action	Comment
sendTX(Msg)	Sends a message across the radio.
sense(SensorId)	Polls the sensor based on id.
setTXPower(Pwr)	Sets radio transmission power
setRadioState(State)	Sets radio status (8states)
setMCUState(State)	Sets MCU status (3states)
saveData(Data)	Saves data to file
computeRelayPartner	Special action for routing
setRelayPartner	Special action for routing
resetRelayPartner	Special action for routing

Table 8.6: Actions for SAMSON agents.

Chapter 9

Testing Scenarios

In order to test the simulator design, functionality, and usefulness, there are two main cases used in this work. The first presents a basic design for flooding in WSN's. Flooding is a classical data dissemination algorithm for wireless networks that is simple (Akkaya and Younis (2005)), and testing with the flooding protocol is useful for checking power computations and validating the basic simulator functionality. This test aims to show that agents in the network can perform such a task easily as a verification of the approach.

The second testcase provides a small case study of a project within ArsLogica called STATUS, (ArsLogica (2008b)), which is a sense and send WSN solution for monitoring servers a server farm of arbitrary size. This scenario has been selected in order to examine the usefulness of the simulator in a modelling a realistic application setting. Additionally, because the scenario uses sensors for different functions, it requires a heterogeneous agent design that requires inter-agent communication through the radio. Below we describe the two tests in more detail:

9.1 Agent Based WSN Flooding

Flooding is the simplest data-centric routing protocol, allowing for messages to be spread across a network in a diffusion-like manner. Whenever a node needs to send a message, it simply broadcasts it to all of its neighbours. When a node receives a message from one of its neighbours, it re-broadcasts the message (again to all of its neighbours). This happens until the message reaches its target recipient or a specified limit to the number of hops for that message. This is also highly inefficient method of routing data, as it performs duplicate sends, duplicate receives, and also consumes

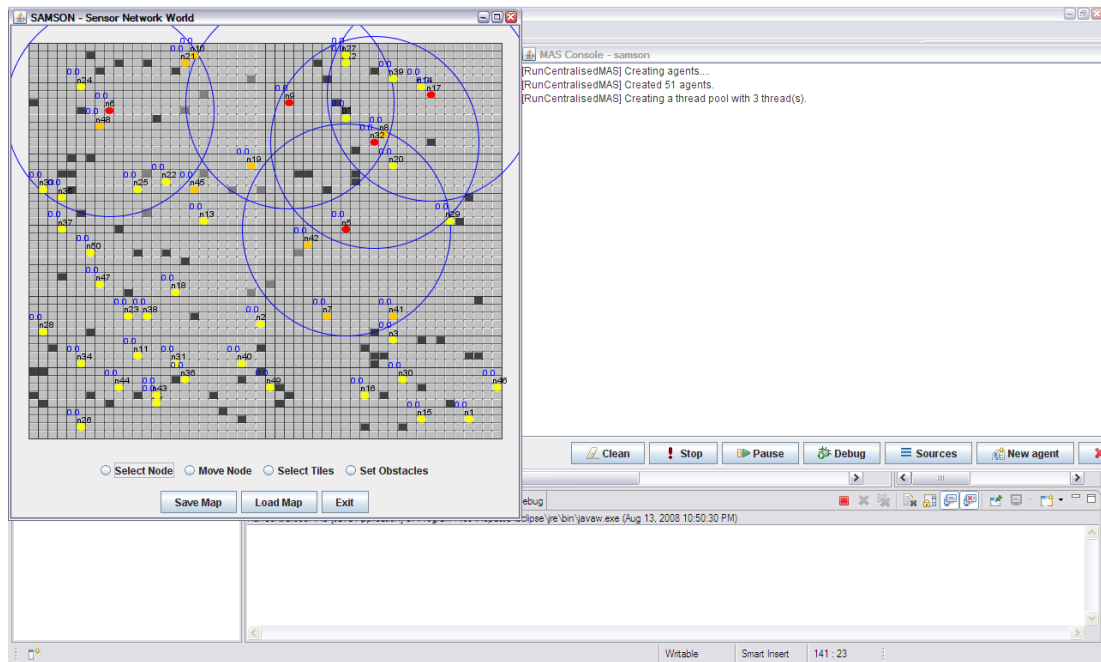


Figure 9.1: Simple flooding scenario with 50 nodes.

energy quickly, (Akkaya and Younis (2005)).

9.1.1 Designs

Design for the flooding scenario consists of a single basestation node and a specified number of active nodes in the network (for this example 50 nodes are used). When the basestation is started, it sends a single request and waits for responses. If a response is received it simply prints it, and tests its battery level. The nodes in the field, however, perform two tasks; when a message of any sort is received, it sends an acknowledgement followed by a broadcast of its current battery voltage. Both messages are sent at the lowest transceiver power level (-24dBm). The corresponding Agentspeak programs are below:

9.2 Agent Based Server Monitoring

As mentioned, a real implementation scenario has been modeled in SAMSON in order show the usefulness of the concept. What follows is a description of the actual Ars-Logica scenario followed by a the agent based version. This scenario aims to highlight how the real approach and the agent approach can reflect each other in a useful fashion. The results of this will be discussed in the following chapter.

Program 2 Agentspeak code for a basestation.

```

!start.
/* Plans */

+!start : true
<-setRadioState(3); //on
sendTX("all", "requestData!").

/*When you believe you have a msg print it*/
+msg(From,X,Y,Z,W) : true
<- .print("dataRcvd from <", From, "> is <", X, "|", Y, "|", Z);
sense(4).

+sensorVal(V) : true
<- .print("Voltage sensed is ", V, " units.").

```

9.2.1 WSN Serverfarm Management

Server farms are the backbone of nearly every information enterprise, and can number thousands of servers for large businesses. In the ArsLogica case, these servers are owned by Italian Postal system. Managing hardware assets in a large system like the Post office becomes a challenge, as servers can become too hot, or may be moved around, added or removed, etc. A server rack is a storage container that holds up to 50 servers in this case. These racks are arranged in groups of ten, with arbitrary large numbers of server-rack groups. What has been done, is to place a low transmission sensor node in each server on the rack (50 nodes per rack). Each rack also has a clusterhead sensor that sends long range messages. Finally, there is one or more basestation nodes that upload information to a wired network via a wireless gateway interface. At this point the data is then stored in a DB for use by monitoring personnel.

The operations of the nodes are as follows: Each low transmission node in a rack sends out a “presence signal” that is broadcast on a low frequency once every few seconds. This signal contains the id of the node, the temperature around the server, and the node’s current battery level. The use of a low frequency ensures that the signal remains bottled inside the server rack (and is not detectable by clusterheads of other racks). These nodes also perform a low power management technique in which they sleep between transmission times.

Program 3 Agent speak code for a sensornode used in flooding.

```

!start.

+!start : true
<- .print("Starting up");
setRadioState(3); //on
setTXPower(-24).

/*When you believe you have a msg print it
and send an acknowledgement, followed by
a voltage reading message.*/
+msg(From,X,Y,Z,W) : true
<- .my_name(Me);
.print("messageRcvd from <", From, ">");
setTXPower(-24);
sendTX("all", "acknowledge");
.print("acknowledgement from ", Me , " to ", From, "");
sense(0).

+sensorVal(V) : true
<- .print("Voltage is ", V, " units.");
sendTX("all", V).

```

The clusterhead for each rack is responsible for aggregating received data from the nodes within the rack. As a precaution validation measurements test the received signal strength of the incoming message in order to ensure the sender is not an outside node (as stray messages may still manage to escape the boundary of the rack). The clusterhead then uses multihop routing to send findings to the basestation node. The packet sent typically contains data such as temperature changes, server added, server removed, or low sensornode power levels. Additionally, the clusterhead may reform the multihop network on command from the basestation node. This involves recomputing nearest neighbour information according to a customized routing protocol.

The basestation acts primarily as a passive receiver of incoming data, and forwards information using Wi-Fi protocols directly to the computer network's gateway server, where the data is stored. The base station node is also able to initiate network reformation by broadcasting a long range reform message to the clusterheads.

9.2.2 Agent-based WSN Serverfarm Management

The agent oriented approach to the scenario described above is to locate the main agent types, highlight their goals, roles, perceptual information from the environment, relevant beliefs, and action plans. These are designed according to the Prometheus methodology, (Padgham and Winikoff (2002)), as discussed below. In Prometheus the starlike shape represents percepts from the agent's sensors; green chevrons represent actions that are available to the agent; blue envelopes represent messages flowing through the system.

It must be mentioned, however, that the designs are not an exact fit with the real scenario, due to several compromises that have been made to accommodate the simulator's limitations. The actual AgentSpeak code is included in Appendix B.

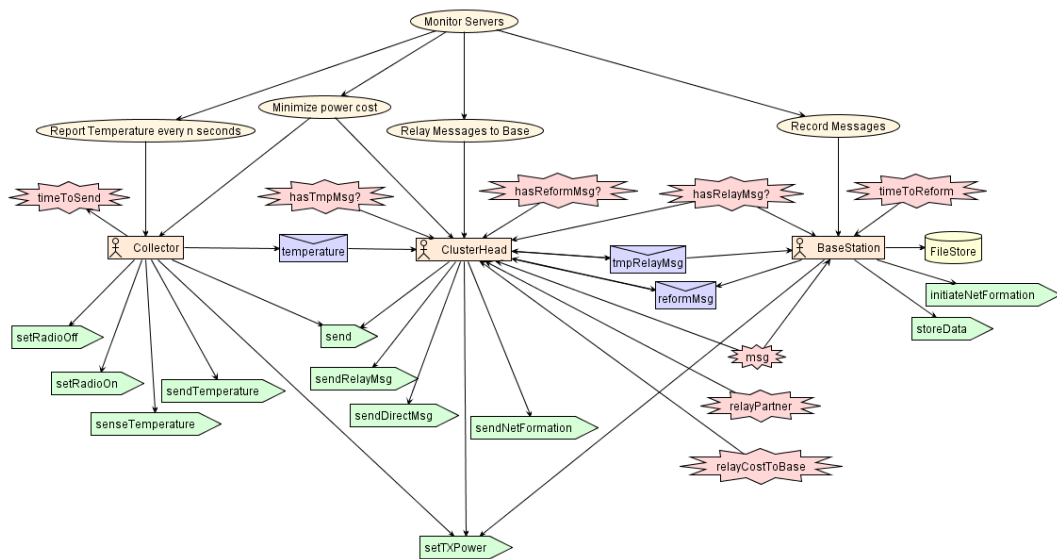


Figure 9.2: Prometheus design showing the system analysis overview - goals, agents, percepts, and actions.

9.2.2.1 System Specification

The first multi-agent system design step is to establish the goal structure of the system. The main objective is to monitor servers, which involves recording messages from the clusterhead, and also minimizing power costs through multihop routing of messages. See the figure 9.3 below for further goal details.

A second step in the process is to assess the agent types and roles within the system. There are three agent types, and five different roles. See the figures 9.5 and 9.4 below.

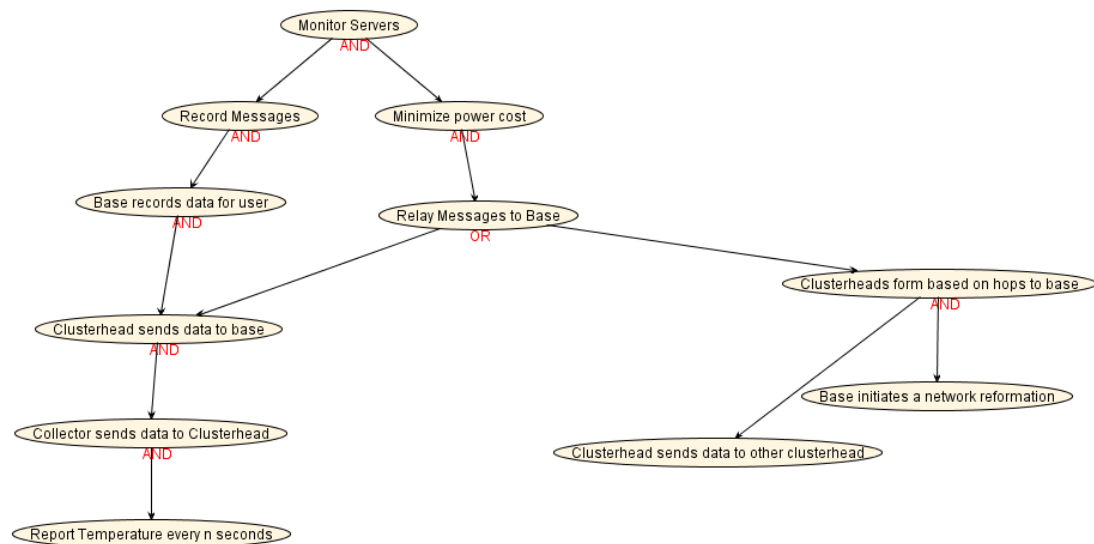


Figure 9.3: Prometheus design showing the system goals.

- Collector agents represent the nodes within each server on the rack. These have the role of DataGatherer.
- Clusterhead agents represent the long range transmission clusterheads for each rack. These have the roles of both NetworkParticipant, and DataForwarder.
- BaseStation agents are primarily passive listeners, as in the earlier scenario, but also initiate network reformation. They have the roles of NetworkOrganizer, and DataKeeper.

[

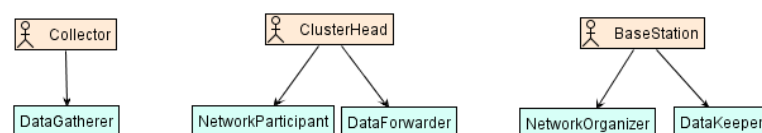


Figure 9.4: Prometheus design showing the system roles.

9.2.2.2 System Architecture

The system overview diagram in figure 9.6 shows the workings of each agent graphically. The Collector agent samples data every N seconds, whenever the “timeToSend” percept from its environment is received. The plan for this involves turning on the radio, sensing the temperature, broadcasting the message, and then turning off the radio.

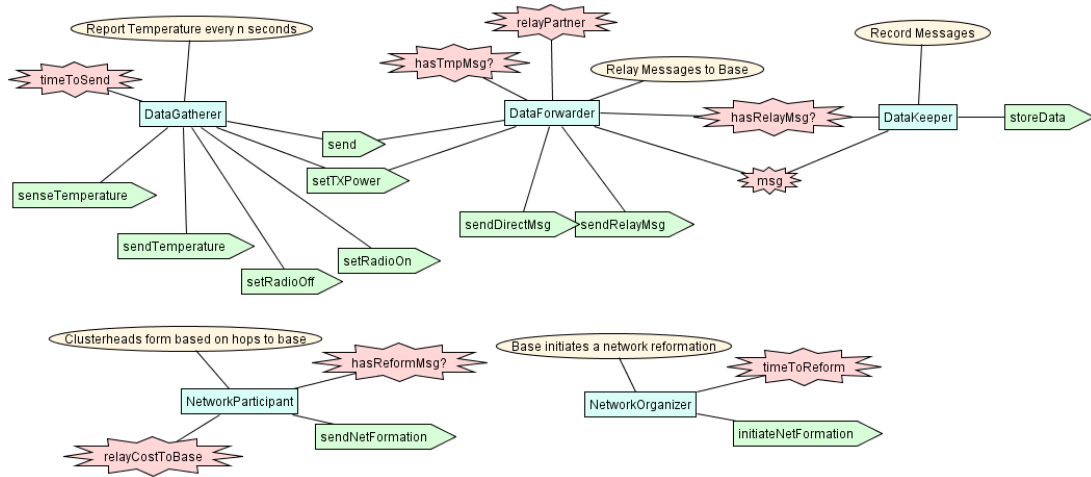


Figure 9.5: Prometheus design showing the system roles and actions.

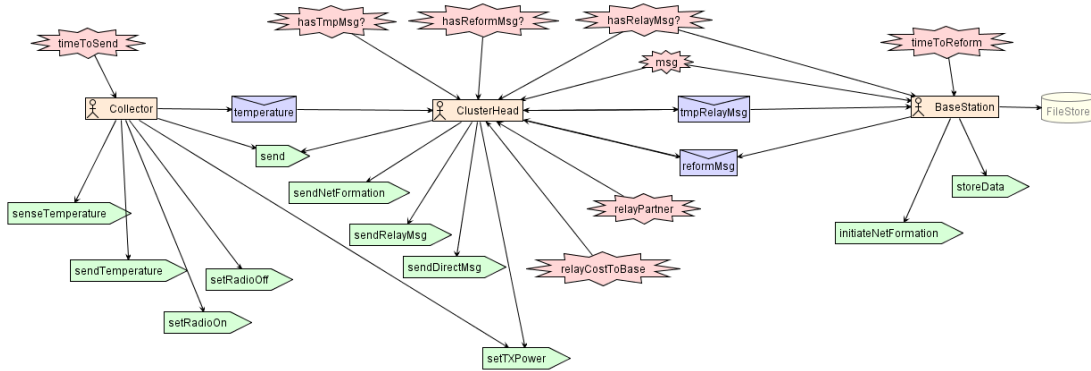


Figure 9.6: Prometheus design showing the system roles and actions.

The Clusterhead agent receives a number of percepts and three different message types for temperature samples, network reform, and “relayMessages” for the basestation. These are represented by the percepts “hasTempMsg”, “hasReformMsg”, and “hasRelayMsg”. The other types of percepts received by the agent are general messages, represented by “msg”, and relay calculation percepts during network reformation (represented by the “relayCostToBase” and “relayPartner” percepts). The agent performs actions for sending messages for network formation requests, sending relay messages to be multihopped to the basestation, and for sending direct messages to the basestation (after updating the transmission power, “setTXPower”).

The original ArsLogica multihop algorithm involves using a complex multihop routing protocol as part of the TinyOS 2.0 platform, (2.x Working Group (2005)), and this has not been simulated. Instead, the protocol for negotiating a relay partner is a simple multihop solution based on received signal strength (RSSI) comparisons (via

an internal action for “computeRelayPartner”). The objective is to choose the highest signal as a partner. The details assume that baseRSSI is N , and a relayRSSI is $(R, N1)$, where R is relayRSSI cost and N is the relay’s baseRSSI.

- if $N > R$ then relay is closer than base.
- if $N1 > N$ then relay has higher base station signal, so set relayPartner(RelayName) and broadcast(R) as the new cost for the node to reach the basestation. Finally, set $N = R$.

The BaseStation agent passively receives messages and saves them to a local file-store (this action represents the process of sending the data to the network gateway). It receives the percept “timeToReform” from its environment and sends out a reform message. Additionally it receives the percepts for messages (“hasRelayMsg” and “msg”).

Finally, it should be noted that the agent servers scenario differs by not having as many nodes in the implementation, as well as sending only temperature information. The clusterhead does not compute summary information or distinguish stray signals and events, or form networks in the same manner (via the same protocol).

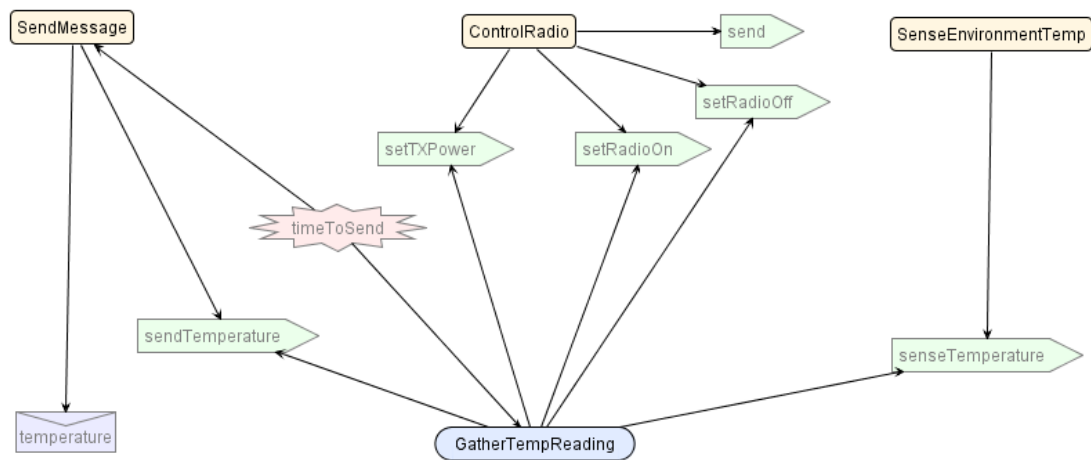


Figure 9.7: Prometheus design detail for a data collector agent.

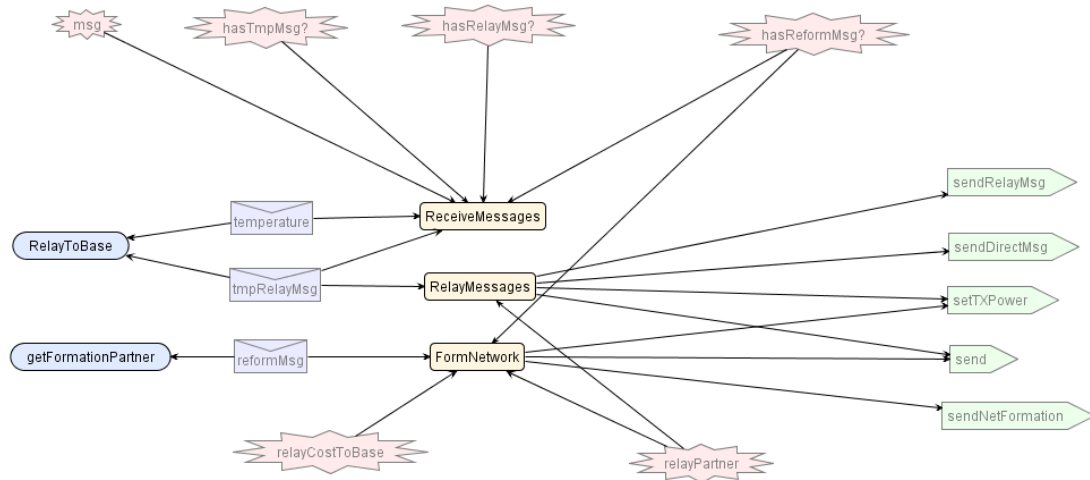


Figure 9.8: Prometheus design detail for a clusterhead agent.

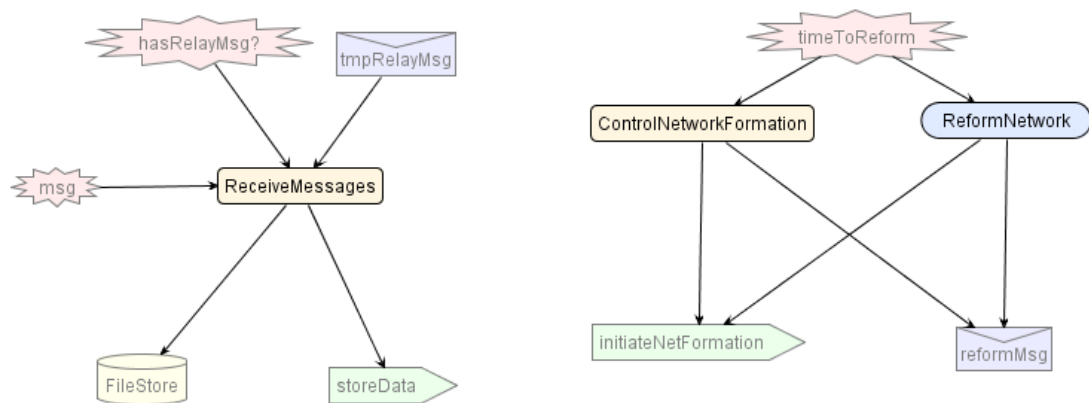


Figure 9.9: Prometheus design detail for a basestation agent.

Chapter 10

Results and Evaluation

10.1 Simulation Environment

The SAMSON experiments have been run on a laptop machine with a 1.3GHz processor and 1GB RAM, operating from within the popular Eclipse Java Development Environment. Both scenarios use a 40 x 40 grid; the flooding test has 50 node agents and 1 basestation; the servers test has 16 datacollector nodes, 4 clusterheads, and 1 basestation.

10.2 Expected Results

The Agent designs of the previous chapter have been selected because they show two features. The flooding scenario is used as a simple validation mechanism, to show that the agents within the network reflect what is normally found for sensor nodes in a similar scenario. In particular, it is expected that the flaws of the flooding approach for data dissemination will be seen, namely implosion, duplicate messages, and high losses of power (Akkaya and Younis (2005)). Further, showing this behaviour is used to validate the hardware model.

The server management scenario represents a different kind of test, in that the focus is on reflecting the behavior of the real ArsLogica server implementation. In this scenario the focus is on showing the usability and fitness of the simulator through assessing what the real nodes do, and how the agents correspond. Due to the unavailability of a simulation environment at ArsLogica, this case has no corresponding data from the real implementations, and thus only behavioral assessment is available. It is expected that the agents in this scenario will function similarly to the case description

of the previous chapter. Data collector nodes should perform low power listening, and thereby save power overall, while still forwarding data. Clusterheads should forward data to the base station via multihop routing, and reform relay partners after a request from the base station. This will result in higher amounts of message passing for clusterheads. The basestation only receives data and initiates net formation. Being able to reproduce this basic behavior will not show that agents improve the scenario, but instead that through simulation strong agency can be used for wireless sensor network interactions.

10.3 Results

10.3.1 Agent-based Flooding

The results from the flooding test are seen in figure 10.3 below. On average the number of messages sent is approximately 16 over 106 timesteps. Messages received averaged eight times that number, approximately 141 messages. This is explained by the number of duplicate messages produced by flooding in a request/acknowledge fashion, and is as expected. Power consumption fell in proportion to messages received, mostly due to radio receive messages being more costly than transmissions, as per the model.

10.3.2 Agent-based Server Management

A summary of the agent based server management scenario execution shows the operation of the basestation, clusterhead and data collectors.

- For the basestation, since it performed less tasks than the other nodes, it maintained the highest power level, and received some 300 messages, on average. 14 Network reform messages were sent. It is also noticed that although a small number of data messages are sent by collectors, the base station receives hundreds more messages. This is very likely due to it receiving negotiation messages from nearby clusterheads as well as the result of wasted messages from multihop routing (the approach is similar to flooding). Also a factor is the lack of data aggregation logic in the clusterheads.
- The clusterheads sent the highest number of messages, and lost the most power, due to the multihop forwarding costs, combined with the costs of network reformation handling.

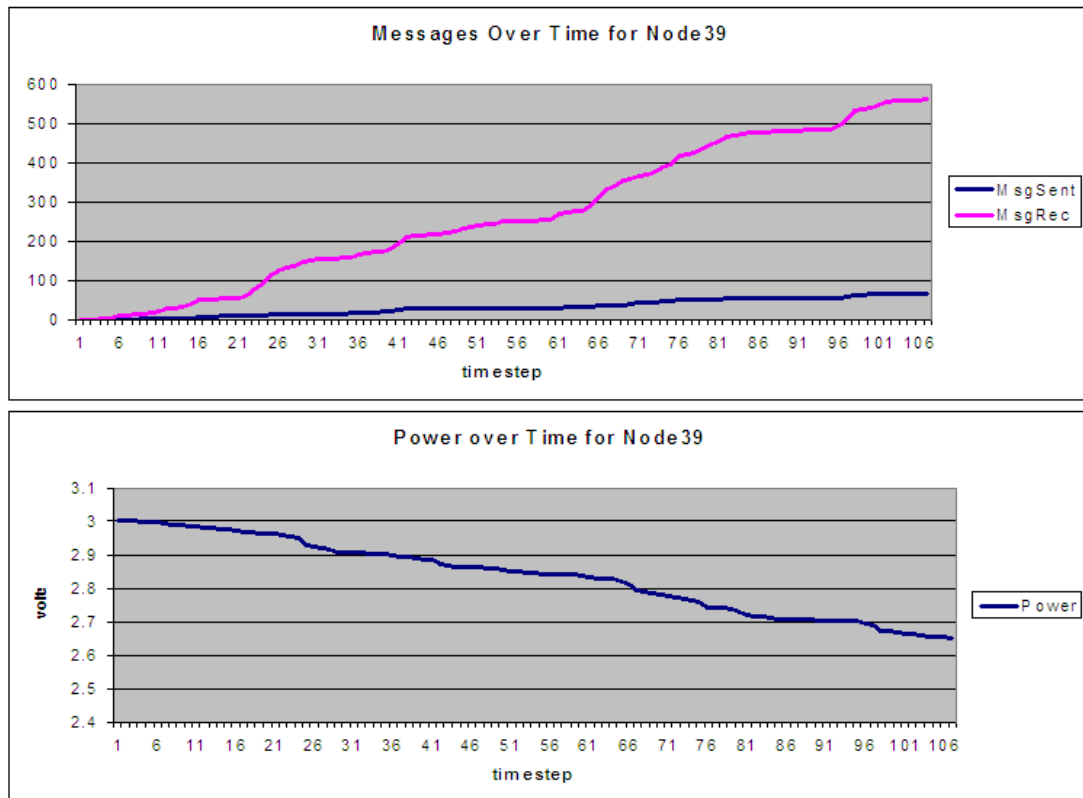


Figure 10.1: Results gained from running the Flooding testcase.

- The data collectors, performed interestingly, sending an average of 50 messages, but receiving 10 times as much. This is likely due to many receipts that would be received during the interval when the radio is turned on before transmitting a message occurs. These messages are from other data collectors nearby (that are sending at the same time) and multihop messages at the clusterhead level. Figure 10.3 shows a datacollector's power consumption over time. The lower spikes are for transmissions, and it may be seen power drops during sleep, and increases when the node wakes before a transmission. Note that whenever a receive happens it causes consumption to spike to approx 65 MWatts, due to TXRXSwitching costs and receiving the message.

10.4 Discussion and Evaluation

Evaluating SAMSON involves looking at the results as well as the simulator to see if the model and the usability meet expectations. For the results shown the simulator

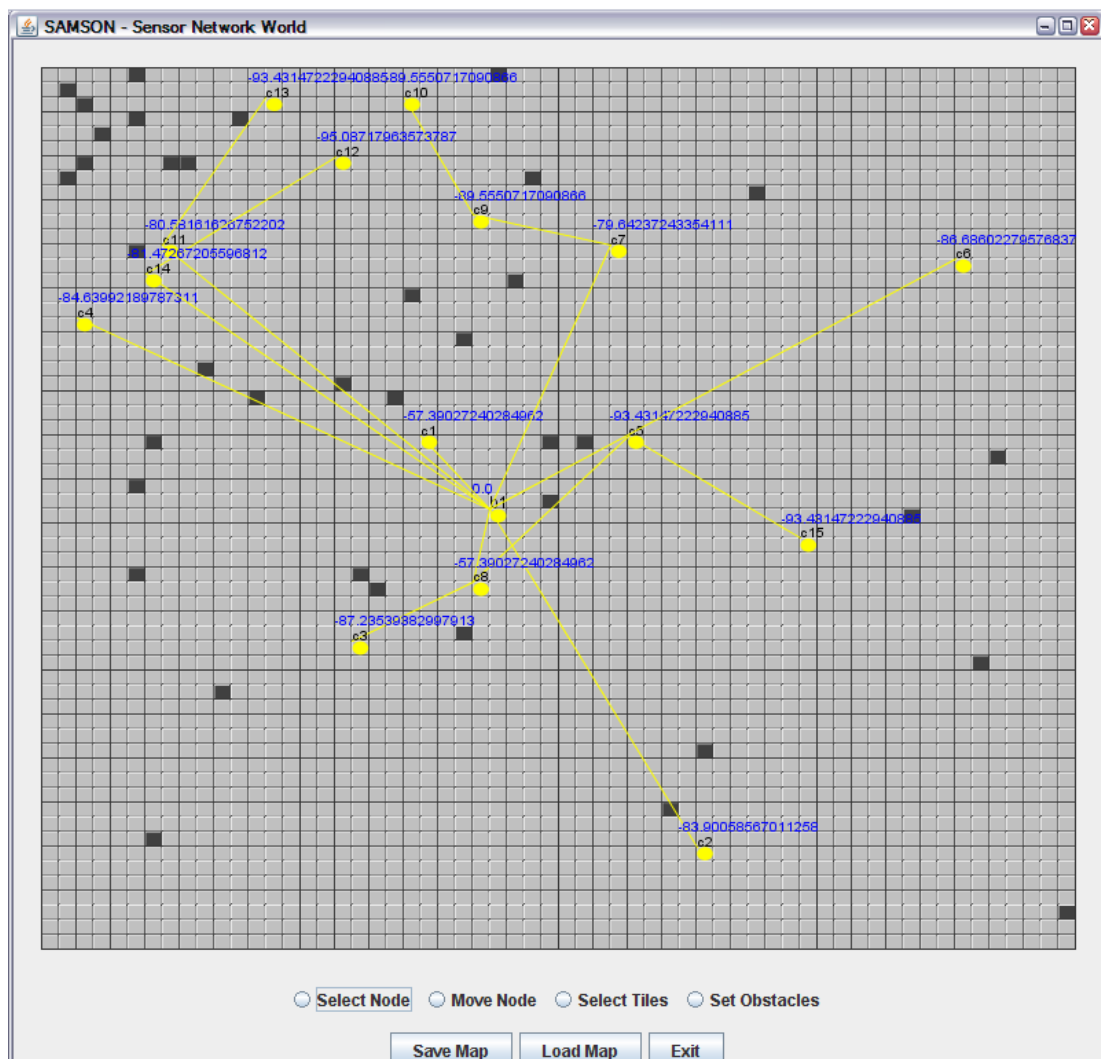


Figure 10.2: Network reformation test consisting of 15 randomly distributed cluster-heads and 1 basestation.

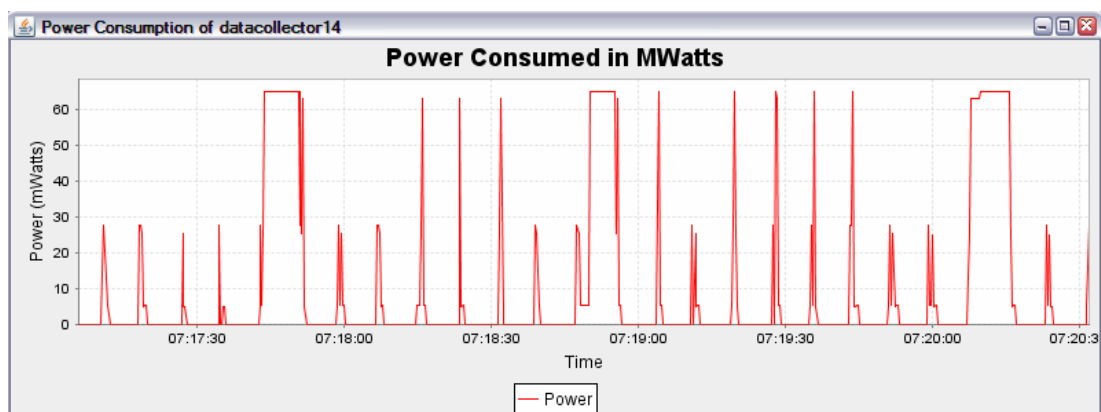


Figure 10.3: Power consumption levels of a datacollector agent.

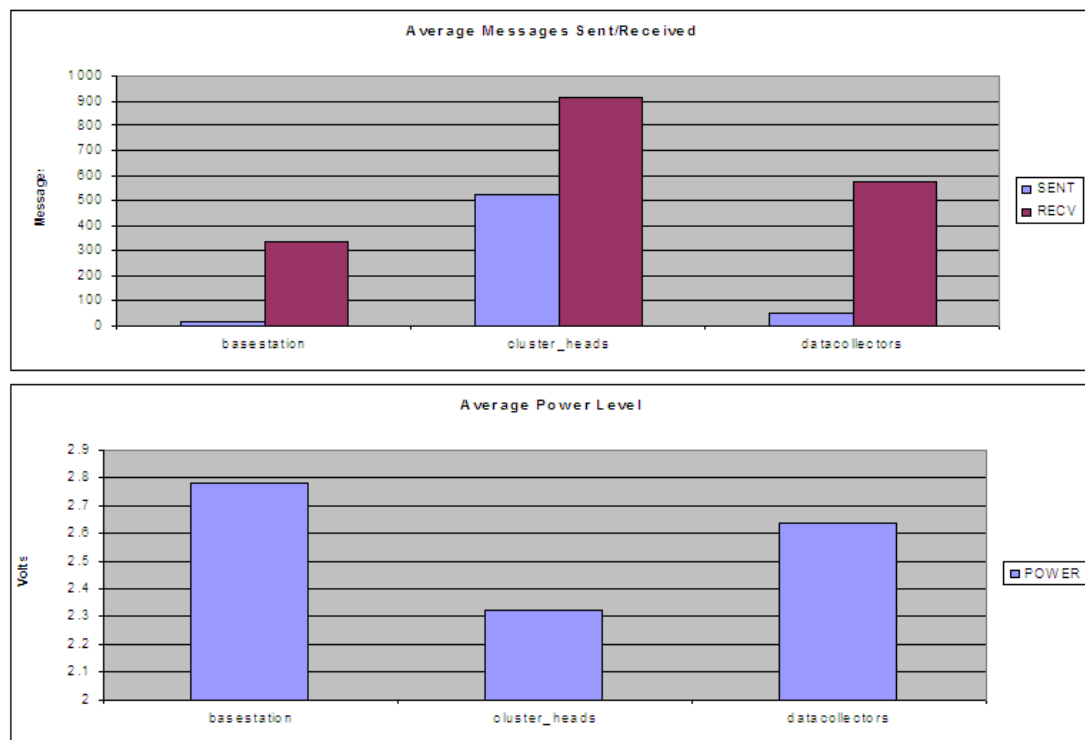


Figure 10.4: Averages of messages sent and received, and power remaining after testing.

performs partially well, providing a good sense of the flooding example, although it does not provide decent stopping conditions for the problem, hence messages tend to be continually sent. For the servers scenario, the simulation functions sufficiently to distinguish the behaviors of the three agent types; despite the discrepancy with the number of messages sent from data collectors not matching up with the number of received messages at the basestation. The multihop succeeded in transferring messages across the clusterheads to the basestation. In short, the agents managed to succeed in imitating some recognizable aspects of the system.

As for the simulator itself, the results have shown that it is adaptable to different scenarios, with the five different agents discussed. Both test scenarios were able to reflect some of the real behavior, although the degree of realism is not directly quantified with actual data. The model for power consumption and message handling works as designed, but could be improved. The environmental models for obstacles, however do not function as designed, and messages are still able to escape despite the radio attenuation factor used. Finally, the simulator would be improved significantly by providing a better interface, and providing for measurements of more interesting parameters. Additionally, there needs to be a method of managing the size of the belief-base used in

Average Data						
NODE	SENT	RECV	POWER	Delta(Sent-Rec)	%Sent	%Recvd
basestation	14	337.6134	2.780663	324	2312%	96%
clusterheads	525.2019	913.4545	2.32255	388	74%	43%
datacollectors	49.60709	576.6865	2.633863	527	1063%	91%
Total Data						
NODE	SENT	RECV	POWER	Delta(Sent-Rec)	%Sent	%Recvd
basestation	9380	226201	1863.044	216,821	2312%	96%
clusterheads	1407541	2448058	6224.433	1,040,517	74%	43%
datacollectors	531788	6182079	28235.01	5,650,291	1063%	91%

Table 10.1: Table showing average results from the Server scenario.

each agent, as this turns out to grow unboundedly in a case such as flooding. Above all, the simulations have shown that the strong agents used in the simulation model wireless sensor networks. However, the benefits gained from such agency, in terms of dynamic behavior, remains to be seen.

In general, there is a difficulty in providing the right level of agent functionality/“internal actions” to interface between Agentspeak and the underlying model, as newer actions needed to be added in order to accommodate the solutions above. Additionally, because Agentspeak code is difficult to debug, some computation (such as selecting a relay partner for a clusterhead) needed to be done in Java. Such difficulty is likely to present a problem for industry practitioners. The work behind the SAMSON simulator considered the key objectives defined in chapter 7. The result has made use of strong agency through the JASON/Agentspeak framework. A hardware model based on parameters from the TMote Sky spec sheet has been shown. Testing for the simulator involved two scenarios; one for classical flooding; the other for server management; thus showing the flexibility of the tool, and producing results that reflect the execution behaviors. The work has been shown to be different from weak agency simulators, and has been compared to the two similar approaches in the literature. Additionally, according to feedback, ArsLogica has found the work promising. However, some of the weaker aspects of the design are found in the environment model’s lack of realistic functions, and the need for more components in the hardware model (such as packet, memory, or radio channel management).

Chapter 11

Related Work

As seen in the discussion about simulating agents in wireless sensor networks, there are few approaches, and even fewer that are based on strong agency. At present strong agency in WSN's is agreed to be limited by the demands of the low-power, low-memory hardware found in most WSN's. This has led most researchers away from the approach of using intentional BDI agents. However, it is seen that hardware is improving, and will eventually allow for strong agency to be used in such networks. There are currently only two simulations of strong agent WSNs in the literature. These provide both a simulation environment that contains a wireless sensor network model, as well as a strong agent definition language/interpreter. The two approaches are that of, (Ma et al. (2006)), on adaptive sampling using AgentFactory/J-Sim, and Zboril's T-Mass WSN simulator for the Agent Low Level Language(ALLL), (Zboril and Zboril (2008)). Expectedly, these approaches differ from each other, and SAMSON.

11.1 T-Mass

The T-Mass simulator has been designed to accommodate the Crossbow sensors, (Crossbow (2008)), with a strong focus on being implementable on that architecture. The design provides an agent platform that is based on FIPA standards for agents, although it does not factor in a common communication language. The approach uses the ALLL in order to describe agent behaviours in a lisp-like format that has the advantage of being similar to actual program code for TinyOS, the common standard operating system for WSN programming. This means that the tool provides plans which involve working at the register level of the agent and hardware, which could be considered complex. Additionally, the tool does not provide a model of the environment. This differs from

SAMSON in that it also focuses on mobile agents that are able to transfer between nodes using a virtual machine style architecture, (Zboril and Zboril (2008)). Additionally, the ALLL language is a combination of calculus approaches, whereas Agentspeak is an abstract architecture based on modal logic, possible worlds theory, and a procedural reasoning system. They are similar in that they both offer plan structures, and have a level of BDI. A further distinction is that T-Mass focuses on the ATMEL hardware for the Crossbow sensors, whereas SAMSON models the newer TMote Sky device.

11.2 AgentFactory/J-Sim

The AgentFactory/J-Sim approach provides a model of WSN simulation through the use of the JavaSim (J-Sim) network framework, and the SensorSim extension for WSN capabilities (Ma et al. (2006)). For the level of agency, the tool uses the AgentFactory Micro-edition. The use of J-Sim provides a complex model of sensor functions for channel modelling, propagation, battery, CPU, and sensing. The use of J-Sim differs from SAMSON's model, which provides only the models for Power Consumption, and Radio. Additionally, the SENS simulator style is used in SAMSON, which abstracts a sensor to being built of its Physical, Network, and Environment components. J-Sim, as an open source project, has had additions of many components to model most facets of the hardware (Sobeih et al. (2005)). SAMSON aims for a simpler approach, which, as mentioned prior, depends on the application uses. However, some of the designs from J-Sim may be incorporated in future iterations of the simulator.

The AgentFactory framework for designing strong agents is one which provides a BDI like approach, that focuses on Beliefs, Commitments, and Plans. This approach is also based on the possible worlds theory that makes use of "multi-modal branching time logic" (Collier (2002)). However, the abstract interpreter of Rao, is not used, so the procedural reasoning model is different (it remains to be seen how these two compare). AgentFactory provides flexible commitment levels, plan refinement, and commitment ordering in order to effect the system state. Of the two related works this one is most similar to JASON/AgentSpeak, but differs primarily on the Agent language, and the choice to use J-Sim.

Finally, neither of the two above simulators have presented testing of a realistic scenario, such as the server management case of the previous chapters. This means that there is still much work to be done in order to compare results properly.

Chapter 12

Conclusions

This work has discussed the potential for using autonomous agents in wireless sensor networks. In particular, it has outlined the vision for such a network, and the work involved to date in using agent mechanisms in order to improve current WSN performance, flexibility, scalability, and adaptiveness. Since deploying such systems is difficult to debug, early testing is important. This work has presented a partial solution; a simulator that is both general purpose, and that also incorporates the work of strong agent research. The SAMSON simulator is among the first to target Agentspeak BDI agents, as defined by Rao (Rao (1996)), to the problem of WSN design and evaluation. The objective of the work has been to produce such a simulation environment, to evaluate agent simulation literature and tools, to design agents for common WSN scenarios, as well as a practical scenario from the ArsLogica project, and to test the BDI paradigm in general. The implementation has combined the work of Hubner and Bordini (Bordini et al. (2007)) along with a simple, yet extensible simulation design, inspired by the SENS simulator (Sundresh et al. (2004)), and a model of industry standard TMote Sky sensor devices.

The work was tested using the classical flooding problem, as well as a real scenario at ArsLogica (an Italian WSN solutions provider). It was found that the simulator design is indeed flexible enough to simulate different testcases involving heterogeneous agents with interaction; two of them have been presented. In terms of its metrics for message passing, radio propagation, and power consumption, the simulator reproduced expected behavior for the testcases. In terms of the functionality gained from having BDI agents in performing WSN tasks it is shown that the results for a typical flooding, or a sense and send application can be achieved with small amounts of Agentspeak code.

However, BDI agency promises much more through its ability to perform reactively to changes, following self-interested goal designs, and through well-designed interaction mechanisms. Although this work has not validated the improvements of having such agent behaviors, it does provide a starting point for future studies.

12.1 Possibilities for Future Work

Among the many possibilities for the simulator, one is to use negotiation protocols, such as those described by Jennings (Dash et al. (2003)) or Kraus (Kraus (2001)), in order to quantify their improvements with the hardware model. This is one of the future goals for the simulator. However, many more improvements will be necessary so as to be able to retrieve as much information as possible. The aspects that could be most useful involves channel modelling and packet management, since these have effects on interaction. This should be done along with adding to the environment model for parameters such as scattering and reflection. Additionally, there are many more testcases and scenarios that may be analyzed with the tool. It is hoped that the work sparks some interest in the agent community, particularly with the Agentspeak/JASON newsgroup.

Appendix A

Appendix A - Implementation Details

This appendix gives a look at the implementation structure found within the project. The UML class diagram is shown in addition to several choice sequence diagrams for the agent actions available in the simulator.

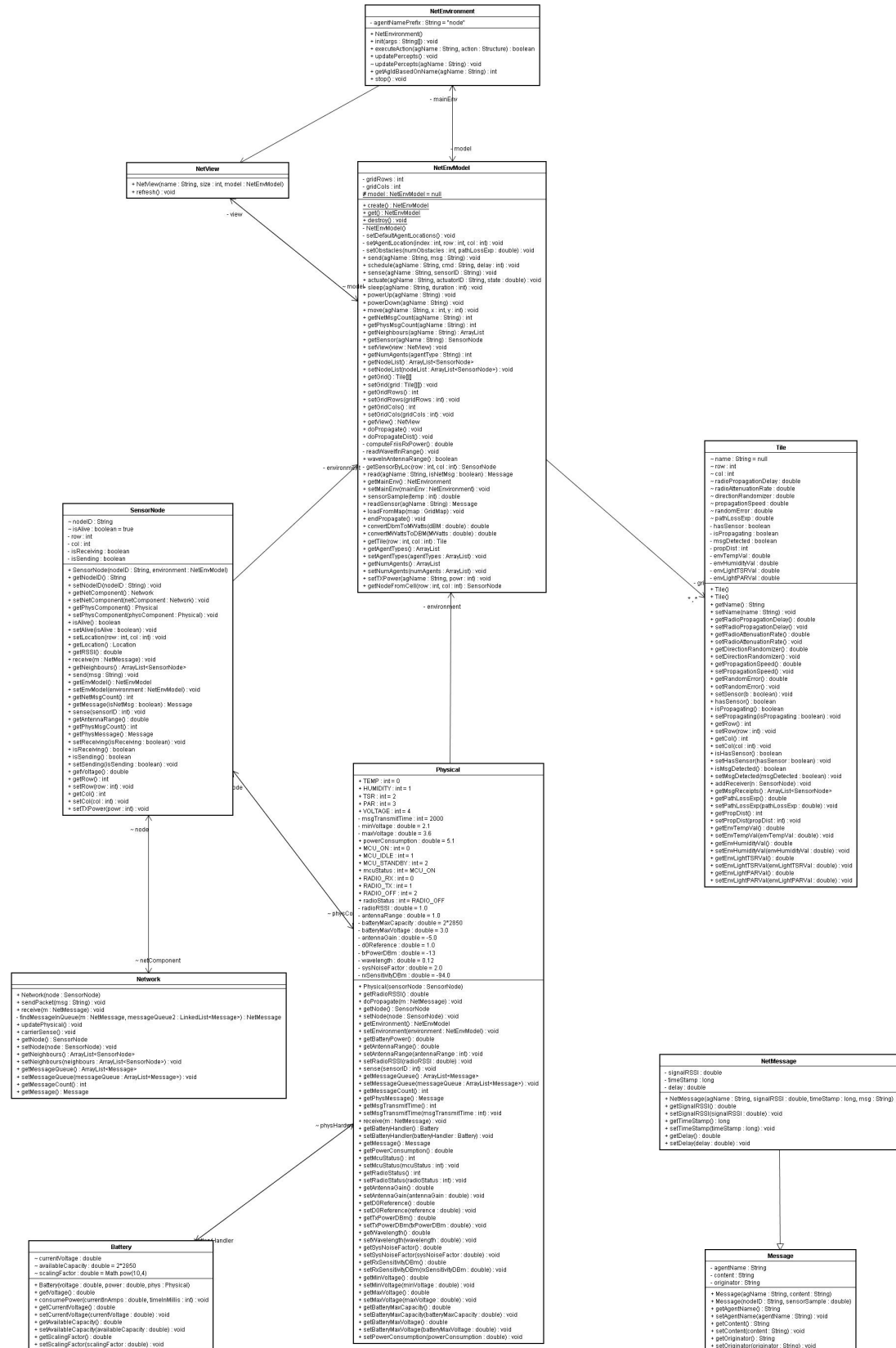
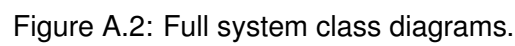


Figure A.1: Class diagram overview.



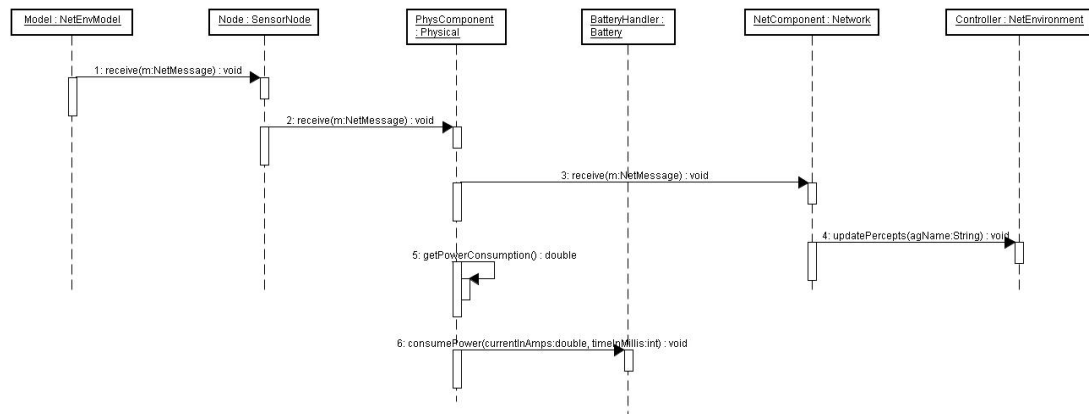


Figure A.4: Sequence diagram for receiving a message.

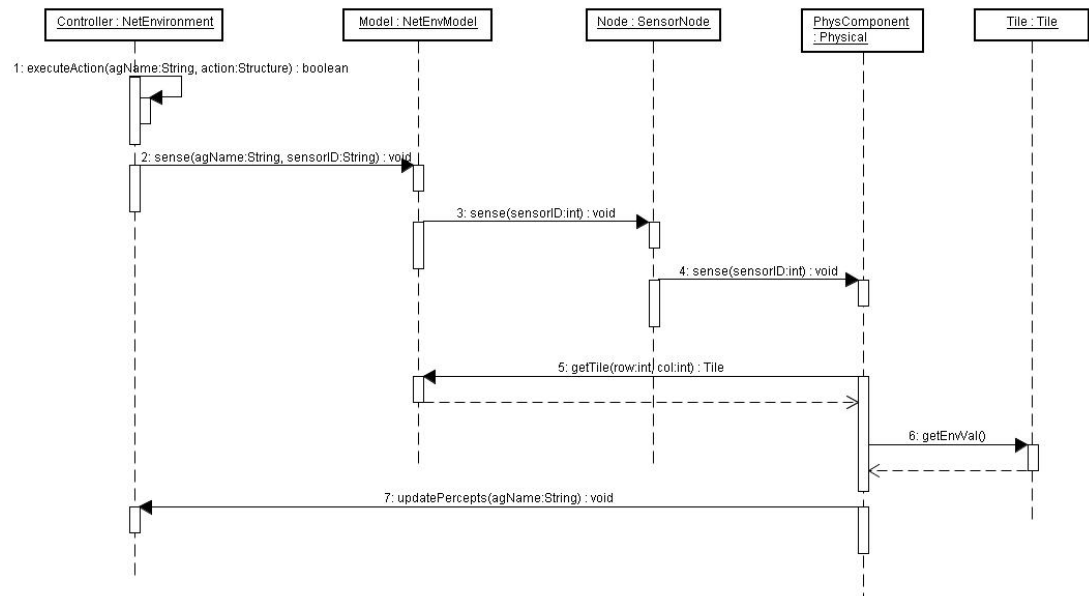


Figure A.5: Sequence diagram for sensing the environment.

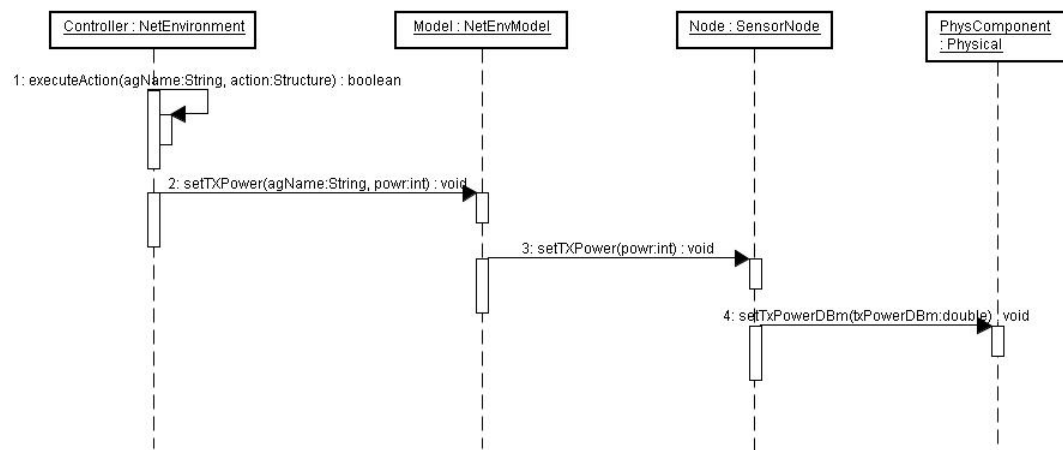


Figure A.6: Sequence diagram for changing the radio transmission power.

Appendix B

Detailed Agent Design

AgentSpeak Code from the Agent based server management is included here. This is all run using the Mas2J configuration script of JASON.

Program 4 The required configuration parameters used in the project.

```
/*
    Jason Project configuration file
*/

MAS samson {
    infrastructure: Centralised(pool,3) //use 3 threadpools

    environment: samson.NetEnvironment(
        "SAMSON - Sensor Network World",
        "basestation#1|datacollector#16|cluster_head#4",
        40,600)

    agents:
        datacollector#16;
        cluster_head#4;
        basestation#1;

    aslSourcePath: "src/asl";
}
```

Program 5 Agentspeak for a data collector.

```

!start.
/* Plans */
+!start : true
<- .at("now + 5 s", "+timeToSend").

+!gatherTempReading : true
<- !senseTemperature;
!sendTemperature.

+!senseTemperature : true
<- sense(0). //Temp sensor

+!sendTemperature : sensorVal(Temp)
<- //.print("send!");
setMCUState(0); //on
setRadioState(3); //on
setTXPower(-24);
sendTX("dc", Temp, 1,1);
setRadioState(2); //off
setMCUState(2). //standby.

/*Acquired Beliefs*/
+timeToSend : true
<- //.print("Time to send!");
!gatherTempReading;
-sensorVal(_);
.abolish(sensorVal(_));
!start.

+msg(_,_,_,_,_)[source(percept)]: true
<- -msg(_,_,_,_,_)[source(percept)].

-msg(X,Y,Z,W,K)[source(percept)] : true
<- .print("msg is deleted").

```

Program 6 Agentspeak for a clusterhead.

```

!start.

/* Plans */

+!start : true
<- .print("Start Listening for Messages.");
setRadioState(3); //on
setTXPower(-24).

/*Acquired Beliefs*/
//From base directly about reforms
+msg(From, To, Content, Pwr, Time, HMx, HCnt) : Content == "reform" &
From == basestation | From == "basestation"
<- .print("reform message from <", From, ">");
resetRelayPartner(From,Pwr);
sendTX("reform", Pwr, 1,1);setRadioState(3).

//From another clusterhead about reforms
+msg(From, To, Content, Pwr, Time, HMx, HCnt) : To == "reform"|To == reform
<- .print("reform message received from <", From, ">");
computeRelayPartner(From, Content, Pwr);setRadioState(3).

+relayPartner(P) : true
<- .print("Relay partner with ", P).

+relayCostToBase(C) : true
<- .print("Relay cost is now ", C);
sendTX("reform", C, 1,1).

+msg(From, To, Content, Pwr, Time, HMx, HCnt) : To == "dc" | To == dc
<- .print("tempRcvd from <", From, ">");
.concat(From, "->", Content, Msg);
sendTX("fwd", Msg,3,1);setRadioState(3).

+msg(From, To, Content, Pwr, Time, HMx, HCnt) : HCnt \== HMx &
To == "fwd" | To == fwd
<- .print("fwd from <", From, ">");
H = HCnt+1;
.concat(From, "->", Content, Msg);
sendTX("fwd", Msg,HMx,H);setRadioState(3).

```

Program 7 Agentspeak for a basestation.

```

!start.

/* Plans */

+!start : true
<- setRadioState(3); //on
!reformNetwork;
.at("now + 25 s", "+timeToReform").

+!reformNetwork : true
<- !initiateNetFormation.

+!initiateNetFormation : true
<- setTXPower(0);
sendTX("all", "reform", 1,1). //reform!

+!storeData(Msg) : true
<- saveData(Msg).

/*Acquired Beliefs*/
+timeToReform : true
<- !reformNetwork;
.abolish(timeToReform); //abolish stuff
!start.

+hasRelayMsg(Msg) : true
<- !storeData(Msg);
.abolish(hasRelayMsg(Msg)). //abolish stuff

+msg(From, To, Content, Pwr, Time, HMx, Hc) : To == "fwd" | To == fwd
<- .print("dataRcvd < ", Content, ">");
+hasRelayMsg(Content).

/*Dropped Beliefs*/

-hasRelayMsg(_) : true
<- .abolish(msg(_, _, _, _, _, _, _)). //abolish stuff

```

Bibliography

- 2.x Working Group, T. T. (2005). Tinyos 2.0. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 320–320, New York, NY, USA. ACM.
- A. Pokahr, L., B. W. (2003). Jadex: Implementing a bdi-infrastructure for jade agents. In *EXPI In Search of Innovation (special issue on JADE)*, volume 3(3), page 7685.
- Akkaya, K. and Younis, M. F. (2005). A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3(3):325–349.
- ArsLogica (2008a). Arslogica it laboratories. <http://www.arslogica.it/>.
- ArsLogica (2008b). Arslogica it laboratories status project. <http://www.arslogica.it/projects/status/status.html>.
- Beutel, J. (2006). Metrics for sensor network platforms. In *REALWSN 06*. ACM.
- Bordini, R. H., Hübner, J. F., and Tralamazza, D. M. (2006). Using *ason* to implement a team of gold miners. In *CLIMA VII*, pages 304–313.
- Bordini, R. H., Wooldridge, M., and Hübner, J. F. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons.
- Bracciali, A., Endriss, U., Demetriou, N., Kakas, A. C., Lu, W., and Stathis, K. (2006). Crafting the mind of prosocs agents. *Applied Artificial Intelligence*, 20(2-4):105–131.
- Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355.
- Brooks, R. A. (1991). Intelligence without representation. Number 47 in *Artificial Intelligence*, pages 139–159.

- Chrisman, L., Caruana, R., and Carriker, W. (1991). Intelligent agent design issues: Internal agent state and incomplete perception. In *In AAAI Fall Symposium Series: Sensory Aspects of Robotic Intelligence*, pages 18–25. Press/MIT Press.
- Collier, R. W. (2002). *Agent Factory: A Framework for the Engineering of Agent-Oriented Applications*. Doctor of philosophy, National University of Ireland, Faculty of Science, University College Dublin.
- Crossbow (2008). Crossbow sensors. <http://www.xbow.com/index.aspx>.
- Dash, R. K., Jennings, N. R., and Parkes, D. C. (2003). Computational-mechanism design: A call to arms. *IEEE Intelligent Systems*, 18(6):40–47.
- David Marsh, Richard Tynan, D. O. G. M. P. O. (2004). Autonomic wireless sensor networks. *Autonomic Computing Systems, Engineering Applications of Artificial Intelligence*, 17(7):741–748.
- Fok, C.-L., Roman, G.-C., and Lu, C. (2005). Mobile agent middleware for sensor networks: an application case study. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 51, Piscataway, NJ, USA. IEEE Press.
- Franklin, S. and Graesser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. pages 21–35. Springer-Verlag.
- Georgoulas, D. and Blow, K. (2006). Making motes intelligent: An agent-based approach to wireless sensor networks. *WSEAS on Communications Journal*, 5(3):515–522.
- ”Howden N, Ronnquist R, H. A. and A”, L. (2001). Jack intelligent agents-summary of an agent infrastructure. In *5th Int’l Conf. on Autonomous Agents*.
- Jennings, N. R. (2000). On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296.
- Jennings, N. R., Faratin, P., Johnson, M. J., Norman, T. J., O’Brien, P., and Wiegand, M. E. (1996). Agent-based business process management. *Int. J. Cooperative Inf. Syst.*, 5(2&3):105–130.

- J.S. Sandhu, A.M. Agogino, A. A. (2004). Wireless sensor networks for commercial lighting control: decision making with multi-agent systems. In *Working Notes of the AAAI-04 Sensor Networks Workshop*, pages 88–92.
- Karl, H. and Willig, A. (2005). *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons.
- Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.
- Kho, J., Rogers, A., and Jennings, N. R. (2007). Decentralised adaptive sampling of wireless sensor networks. In *1st Int Workshop on Agent Technology for Sensor Networks*.
- Kraus, S. (2001). *Strategic negotiation in multiagent environments*. MIT Press, Cambridge, MA, USA.
- Levis, P., Lee, N., Welsh, M., and Culler, D. (2003). Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137, New York, NY, USA. ACM.
- Ma, R., O'Hare, G. M. P., and O'Grady, M. J. (2006). Embedded intelligence: Enabling in-situ power management for wireless sensor networks. In *EuroSSC*, pages 244–247.
- Newell, A. (1980). Aaai president's message. *AI Magazine*, 1(1):1–4.
- Padgham, L., Thangarajah, J., and Winikoff, M. (2005). Tool support for agent development using the prometheus methodology. *Quality Software, 2005. (QSIC 2005). Fifth International Conference on*, pages 383–388.
- Padgham, L. and Winikoff, M. (2002). Prometheus: a methodology for developing intelligent agents. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 37–38, New York, NY, USA. ACM.
- Pollack, M. E. and Ringuette, M. (1990a). Introducing the tileworld: Experimentally evaluating agent architectures. In *In Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189.

- Pollack, M. E. and Ringuette, M. (1990b). Introducing the tileworld: Experimentally evaluating agent architectures. In *In Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189.
- Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In van Hoe, R., editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands.
- Rao, A. S. and Georgeff, M. P. (1995). BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco.
- Reiner Jedermann, Christian Behrens, D. W. W. L. (2006). Applying autonomous sensor systems in logistics—combining sensor networks, rfids and software agents. *Sensors and Actuators A: Physical, The 19th European Conference on Solid-State Transducers*, 132(1):370–375.
- Riley, G. F. (2003). The georgia tech network simulator. In *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 5–12, New York, NY, USA. ACM.
- Rogers, A., David, E., and Jennings, N. (2005). Self-organized routing for wireless microsensor networks. *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 35(3):349–359.
- Rosenschein, J. and Zlotkin, G. (1994). *Rules of Encounter*. MIT press.
- Ruiz, L., Braga, T., Silva, F., Assuncao, H., Nogueira, J., and Loureiro, A. (2005). On the design of a self-managed wireless sensor network. *Communications Magazine, IEEE*, 43(8):95–102.
- Sentilla (2008a). Sentilla.com. <http://www.sentilla.com/index.html>.
- Sentilla (2008b). Tmote sky datasheet. <http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf>.
- Simon, D., Cifuentes, C., Cleal, D., Daniels, J., and White, D. (2006). Java™ on the bare metal of wireless sensor devices: the squawk java virtual machine. In *VEE '06: Proceedings of the 2nd international conference on Virtual execution environments*, pages 78–88, New York, NY, USA. ACM.

- Sobeih, A., Chen, W.-P., Hou, J. C., Kung, L.-C., Li, N., Lim, H., Tyan, H.-Y., and Zhang, H. (2005). J-sim: A simulation environment for wireless sensor networks. In *ANSS '05: Proceedings of the 38th annual Symposium on Simulation*, pages 175–187, Washington, DC, USA. IEEE Computer Society.
- Stathis, K., Kafetzoglou, S., Papavassiliou, S., and Bromuri, S. (2007). Sensor network grids: Agent environments combined with qos in wireless sensor networks. *icas*, 0:47.
- Sundresh, S., Kim, W., and Agha, G. (2004). Sens: a sensor, environment and network simulator. *Simulation Symposium, 2004. Proceedings. 37th Annual*, pages 221–228.
- Tynan, R., Ruzzelli, A. G., and O'Hare, G. M. P. (2005). A methodology for the development of multi-agent systems on wireless sensor networks. In *SEKE*, pages 68–75.
- Vinyals, M., Rodrguez-Aguilar, J. A., and Cerquides, J. (2008). A survey on sensor networks from a multi-agent perspective. In *2th International Workshop on Agent Technology for Sensor Networks (ATSN-08)*.
- Wikipedia (2008). Battery (electricity): From wikipedia, the free encyclopedia. [http://en.wikipedia.org/wiki/Battery_\(electricity\)](http://en.wikipedia.org/wiki/Battery_(electricity)).
- Wolf, T. D. and Holvoet, T. (2003). Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control. In *In Proceedings of the First International Workshop on Autonomic Computing Principles and Architectures*, page 10.
- Wooldridge, M. and Ciancarini, P. (2000). Agent-Oriented Software Engineering: The State of the Art. In Ciancarini, P. and Wooldridge, M., editors, *First Int. Workshop on Agent-Oriented Software Engineering*, volume 1957, pages 1–28. Springer-Verlag, Berlin.
- Yadgar, O. and Kraus, S. (2006). Coverage density as a dominant property of large-scale sensor networks. In *CIA*, pages 138–152.
- Zboril, F. and Zboril, V. F. (2008). Simulation of wireless sensor networks with intelligent nodes. In *10th International Conference on Computer Modelling and Simulation*, page 6. IEEE Computer Society.