

Robust Real-Time Event Services in Wireless Sensor Networks

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Doctor of Philosophy (Computer Science)

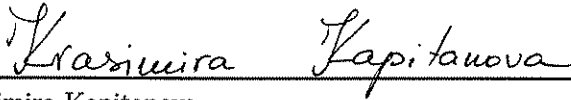
by

Krasimira Kapitanova

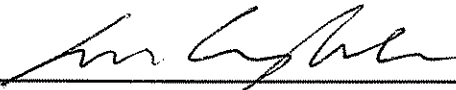
May 2012

Approval Sheet


This dissertation is submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy (Computer Science)


Krasimira Kapitanova

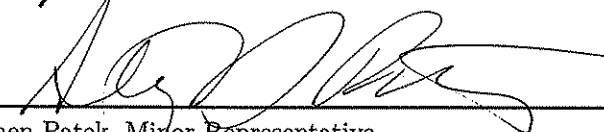
This dissertation has been read and approved by the Examining Committee:


Sang H. Son, Adviser

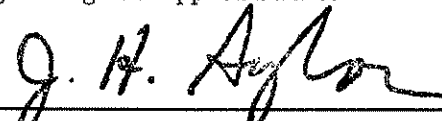

John A. Stankovic, Committee Chair


Kamin Whitehouse


Alfred Weaver


Stephen Patek, Minor Representative

Accepted for the School of Engineering and Applied Science:


James Aylor, Dean, School of Engineering and Applied Science

May 2012

Abstract

EVENT detection is one of the main components in numerous wireless sensor network (WSN) applications. Regardless of the specific application, the network should be able to detect if particular events of interest have occurred or are about to. Traditional event services allow for the definition of events including correlated events, registering for events, and upon occurrence of events, detection and notification of events. In WSNs, events are not binary, but are based on sensor fusion from many noisy sensors in complicated environments. Sensor data may be missing, wrong, or out of date. Consequently, event services must operate in real-time, support data fusion and confidence calculations, and conserve power. Event services must also fundamentally recognize location, since sensor network events are a function of where they occur. Sensor network event services must be highly decentralized in order to work on the limited capacity devices. The services must also minimize false alarms. All these features make building event services for WSNs very challenging.

This research focuses on enabling the design and development of robust real-time event services. More specifically this work investigates the following research problems:

Event specification: We have developed a formal event description language which is an enhanced Petri net and combines features from Colored, Timed and Stochastic Petri nets. This language, coMpack Event Detection and Analysis Language (MEDAL), can capture the structural, spatial, and temporal properties of a complex event detection system. MEDAL also addresses key aspects of sensor networks, such as communication, actuation, and feedback control. MEDAL's graphical support, inherited from Petri nets, makes the application models easy to understand and accessible to a wide range of users.

Event detection: The majority of current event detection approaches rely on using precise, also called “crisp”, values to specify WSN events. However, we believe that crisp values cannot adequately handle the often imprecise sensor readings. In this work we have studied how using fuzzy values could improve the accuracy, timeliness, and resource requirements of event detection. Our experiments with real-world fire data have shown that using fuzzy values results in more accurate event detection than when crisp logic is used, since fuzzy logic is more resilient to imprecise sensor readings.

Robustness to node failures: Even if an event detection system has been correctly designed and built, its continuous and reliable operation is difficult to guarantee due to hardware degradation and environmental changes. However, not all node failures have the same effect on applications' behavior. Some node failures are critical and lead to significant application degradation, while others may not affect the application at all. We have designed techniques to detect node failures that affect the application-level behavior of the system and minimize the number of maintenance dispatches without sacrificing the event detection accuracy of the application.

The techniques and approaches presented in this dissertation have the potential to:

- Broaden the range of events detected by sensor network applications.
- Provide formal specification and broad analysis capabilities for event-driven sensor network applications.
- Improve the accuracy of event detection.
- Help verify at runtime that a WSN application is able to satisfy its high-level requirements.
- Improve the accuracy of event detection applications in the presence of node failures.
- Decrease the number of necessary maintenance dispatches needed to preserve satisfactory application behavior.

Acknowledgments

I would like to take this opportunity to thank the people who have helped me through this journey.

The role played by my advisor, Sang Son, in shaping this work and my career cannot be expressed in just a few words. I cannot thank him enough for his support, advice, and encouragement. I will always be in his debt for taking me as a student at a really hard time for my graduate student career and for believing in me.

I would also like to thank Jack Stankovic and Kamin Whitehouse for all of their help and advice. I am incredibly lucky that, rather than having only one academic advisor, I also had the two of them to guide me in my research. To my fellow graduate students and coworkers - Yafeng Wu, Jingyuan Li, and Enamul Hoque - thanks a lot for all of your work and for putting up with me.

Wes Weimer has also helped me a lot through these years. I would like to thank him for all of his invaluable advice on academic issues, job applications, and various other dilemmas, as well as for his continuous support for whatever departmental endeavor Ray and I decided to pursue.

Jim Cohoon has also provided me with a lot of help and encouragement. I will always be grateful for his time and enthusiasm in helping me prepare for my Theory qual. In addition, being a TA for his class for so many semesters was an extremely valuable experience that made me realize how much I really enjoy teaching.

I would also like to thank my family, Kurt, and my friends - you mean the world to me. I am very grateful for all of your support and for providing all of those much-needed distractions from school.

Contents

Acknowledgments	iv
Contents	v
List of Tables	viii
List of Figures	ix
I Introduction	1
1 Introduction	2
1.1 Problem statement	3
1.2 Contributions	4
1.2.1 Event specification	4
1.2.2 Event detection	4
1.2.3 Robustness to node failures	5
1.3 Dissertation organization	7
2 State of the art	8
2.1 Event specification	8
2.1.1 Modeling sensor network events	8
2.1.2 Modeling data streams	10
2.2 Event detection	10
2.2.1 Stochastic methods	10
2.2.2 Fuzzy logic	11
2.3 Robustness to sensor node failures	11
2.3.1 Sensor network testing	11
2.3.2 General fault tolerance techniques	11
2.3.3 Sensor network debugging	12
2.3.4 Sensor network hardware verification	12
2.3.5 Sensor network fault identification	12
2.3.6 Classifier ensembles	13
II Event specification	14
3 coMPact Event Description and Analysis Language (MEDAL)	15
3.1 Introduction to Petri nets	18
3.2 MEDAL	20
3.2.1 Semantics	20
3.2.2 Temporal and spatial logic	22
3.3 Extending MEDAL's expressiveness	23
3.3.1 Communication	23
3.3.2 Actuation	24

3.3.3	Feedback control	27
3.4	Additional modeling concerns	29
3.4.1	Node mobility	29
3.4.2	Prioritization	30
3.5	MEDAL aided analysis for sensor network applications	31
3.5.1	Real-time analysis	31
3.5.2	Safety analysis	32
3.6	Evaluating MEDAL's modeling properties	34
3.6.1	SQL	34
3.6.2	Snlog	34
3.6.3	Macroprogramming	35
3.6.4	TLA+	35
3.6.5	Abstract regions	36
3.6.6	SNEDL	36
3.6.7	MEDAL	36
3.7	Experience with MEDAL: Beef monitoring	37
3.8	MEDAL models of existing applications	39
3.8.1	Volcano monitoring application	39
3.8.2	Rural fire detection application	41
3.8.3	Early warning flood detection application	42
3.9	Summary	44
4	Applying Formal Methods to Modeling and Analysis of Real-Time Data Streams	45
4.1	System model	46
4.1.1	Periodic query model	47
4.1.2	Query plan and query execution	48
4.1.3	Data admission controller	49
4.2	Modeling queries and control	51
4.2.1	MEDAL query plans	51
4.2.2	Data admission control	52
4.3	MEDAL analysis	53
4.3.1	Query optimization analysis	53
4.3.2	Real-time stream analysis	54
4.4	Modeling DSMS configurations with MEDAL	54
4.4.1	Single query service class	54
4.4.2	Multiple query service classes, no controller	55
4.4.3	Multiple query service classes, single controller	56
4.4.4	Multiple query service classes, multiple controllers	57
4.5	Summary	57
III	Event detection	59
5	Event Detection in Wireless Sensor Networks - Can Fuzzy Values Be Accurate?	60
5.1	Overview of fuzzy logic	62
5.1.1	Fuzzification	62
5.1.2	Decision making	63
5.1.3	Defuzzification	63
5.2	Event semantics	64
5.2.1	Spatial semantics	65
5.2.2	Temporal semantics	66
5.3	Decreasing the size of the rule-base	66
5.3.1	Separating the rule-base	67
5.3.2	Combining rules with similar outcomes	68

5.3.3	Incomplete rule-base	68
5.4	Evaluation	69
5.4.1	Experiments using real fire data	69
5.4.2	Experiments using nuisance fire data	75
5.4.3	Analysis	77
5.5	Summary	80
IV	Robustness to node failures	81
6	Run time assurance (RTA)	82
6.1	RTA Methodology	84
6.2	Implementation Framework	85
6.2.1	The MEDAL Programming Model	85
6.2.2	Automated Test Generation	86
6.2.3	Automated Code Generation	89
6.2.4	Translating MEDAL into a script	90
6.2.5	Code structure	91
6.2.6	Test Execution Support	92
6.3	Case Study	92
6.4	Evaluation	94
6.4.1	Test reduction	94
6.4.2	Robustness to Failure	96
6.4.3	Overhead	101
6.5	Summary	101
7	Simultaneous Multi-classifier Activity Recognition Technique (SMART)	102
7.1	State of the art	104
7.2	Approach	105
7.2.1	Using multiple simultaneous classifiers	106
7.2.2	Failure detection	107
7.2.3	Node failure severity analysis	109
7.2.4	Maintaining detection accuracy under failures	110
7.3	Experimental setup	111
7.4	Results	113
7.4.1	Detecting sensor node failures	113
7.4.2	Node failure severity assessment	114
7.4.3	Maintaining high activity recognition accuracy under failures	117
7.5	Discussion	118
7.6	Summary	119
V	Conclusions and future work	121
8	Conclusions and future work	122
8.1	Results and contributions	122
8.1.1	Event specification	122
8.1.2	Event detection	124
8.1.3	Robustness to node failures	124
8.2	Limitations and future work	126
8.2.1	Event specification	126
8.2.2	Event detection	126
8.2.3	Robustness to node failures	126
	Bibliography	128

List of Tables

3.1	Modeling properties of sensor network event description languages Legend:●- a language partially has the property; ✓- a language has the property.	34
5.1	An example fire detection rule-base. The size of the rule-base is determined by the number of linguistic variables and what values they can hold. In this rule-base, there are six linguistic variables, T_1 , ΔT_1 , T_2 , ΔT_2 , S , ΔS , each of which can hold a value of <i>Low</i> (L), <i>Medium</i> (M), or <i>High</i> (H).	65
5.2	Rule-base for a temperature sensor. The two linguistic input variables and the fire confidence consequent can be classified as Low (L), Medium (M), or High (H).	67
5.3	Reduced rule-base for a temperature sensor. In this rule-base, rules with similar consequents have been combined with the help of the \leq and \geq operators.	68
5.4	Fire detection rule-base for the scenario where a node decides if there is a fire based only on its own sensor readings. The Temperature, Temperature difference, Smoke, and Smoke difference variables take Low (L), Medium (M), and High (H) values.	73
5.5	Number of incorrect classifications by a Naive Bayes classifier and a J48 Tree.	79
5.6	Fire detection delay in seconds.	79
6.1	Combining static analysis techniques and knowledge about the network topology and node redundancy reduces the size of the test suite for the FD application by 35 orders of magnitude.	95
6.2	When all three reduction steps are applied, adding an extra sensor only doubles the number of necessary tests instead of causing an exponential increase.	95
6.3	Compared to HM, RTA has a larger memory footprint.	101
7.1	Sensors participating in the <i>prepare breakfast</i> , <i>prepare lunch</i> , and <i>prepare dinner</i> activities in the WSU house, <i>prepare breakfast</i> and <i>preparing dinner</i> activities in House A and House B.	112
7.2	Average MTTF for all five activities. Our approach increases the MTTF 3.2 times on average. The MTTF improvement achieved with each classifier varies with the nature of the activity. SMART chooses the most suitable classifier at any time and thus achieves the highest MTTF.	115
7.3	We evaluate how using SMART instead of classifiers trained on all nodes affects the activity recognition accuracy. The average accuracy improvement under the presence for failures the classifiers we analyze is 15%.	118

List of Figures

3.1	Probabilistic events scenario.	17
3.2	An example Petri net consisting of four places and two transitions. In the initial marking of the Petri net place p_1 and place p_2 each have a token, which enables both transitions. Depending on which transition fires first, the Petri net could end up in 2 different states.	19
3.3	MEDAL model of an explosion detection application, which uses readings from temperature, light, and acoustic sensors to determine if there is an explosion in the monitored area.	21
3.4	By introducing a dedicated <i>radio transition</i> (transition R1) to MEDAL, we can explicitly model the network communication.	24
3.5	The MEDAL model of this explosion application cannot clearly specify the requirement that the temperature, light, and acoustic nodes should remain awake for 3 minutes after the motion activity has terminated.	25
3.6	We introduce inhibitor arcs to help model actuation in MEDAL. In this explosion detection model, the temperature, light, and acoustic nodes sleep unless the token in place I is consumed, which occurs only if there is activity in the monitored area. Specifying $\beta = 3$ min for transition T2 allows the model to address the requirement that the network should remain awake 3 min after the last motion activity has been detected.	26
3.7	MEDAL model of an explosion application where the sensing frequency of the nodes is required to change based on the value of the temperature in the monitored area. Introducing a feedback loop to MEDAL (shown with double lines) allows us to model such a requirement.	28
3.8	MEDAL can be used to explicitly model mobility application requirements. This explosion application model specifies that when an explosion is detected, the mobile sink should move towards the area of the explosion. The figure also shows a more detailed view of the movement logic.	30
3.9	The MEDAL model of an application, together with time notations associated with the transitions in the model, can be used to perform real-time analysis.	32
3.10	MEDAL model of an application used to detect dangerous gases. The network can detect three types gases, A, B, and C. Exposure to the mixture of gas A and gas B for longer than an hour is considered dangerous and so is the exposure to gas C for more than 40 min.	33
3.11	MEDAL diagram for a beef monitoring sensor network application. There are four types of sensor nodes in the network: light, temperature, humidity, and air circulation. When any of the food safety conditions are violated, the respective sensor sends a message over the radio to notify the base station.	38
3.12	MEDAL model of a volcano monitoring applications. There are two types of nodes in the network: sensing nodes, containing seismic and acoustic sensors, and a base station node. For clarity, the application activities when a nearby volcanic activity has been determined are in bold.	40
3.13	MEDAL model of a rural and forest fire detection applications. There are four types of nodes in the network: sensing nodes, containing fire infrared radiation and smoke, cameras, a central server, and a computer, which is used to visualize the data from the camera.	41

3.14	MEDAL model of a flood detection application. There are three types of nodes in the network: sensing nodes, which include pressure, temperature, and rainfall sensor nodes; computation nodes, which evaluate the probability of a flood; and interface nodes, which can be used to visualize the data and send requests to the computation nodes.	43
4.1	An example query plan.	49
4.2	Data admission controller.	50
4.3	Using MEDAL to model the query plan from Figure 4.1. The query operators are modeled using transitions, while application state and data are represented with the help of places. . .	51
4.4	MEDAL model of the data admission control implementation. This design enables different queries to have different data admission ratios even when they share the same data stream source.	53
4.5	MEDAL model of a DSMS configuration, in which there is a single controller and all queries belong to the same query service class.	55
4.6	MEDAL model of a system with three different service classes, <i>class 0</i> , <i>class 1</i> , and <i>class 2</i> , and no admission controller	55
4.7	MEDAL model of a scenario with three query service classes, <i>class 0</i> , <i>class 1</i> , and <i>class 2</i> , which share a single data admission controller.	56
4.8	MEDAL model of a scenario with three service classes, <i>class 0</i> , <i>class 1</i> , and <i>class 2</i> , where each service class has its own designated controller.	57
5.1	A fuzzy logic system contains three main components: a fuzzifier, decision making, which consists of an inference scheme and a rule-base, and a defuzzifier.	62
5.2	Temperature membership function. Using this membership function, a temperature value can be classified as: <i>Freezing</i> , <i>Freezing and Cold</i> , <i>Cold</i> , <i>Cold and Warm</i> , <i>Warm</i> , <i>Warm and Hot</i> , and <i>Hot</i>	63
5.3	Membership functions for the four input linguistic variables: temperature, temperature difference, smoke obscuration, and smoke obscuration difference.	70
5.4	Fire confidence membership function.	70
5.5	Burning mattress simulation: a) crisp value detection b) fuzzy value detection.	71
5.6	Burning chair simulation: a) crisp value detection b) fuzzy value detection.	71
5.7	Burning oil simulation: a) crisp value detection b) fuzzy value detection.	71
5.8	Simulating a burning mattress: including neighbor readings in the decision. The results when only own values are used are plotted on the first y-axis. Including the neighbor values is plotted on the second y-axis.	74
5.9	Simulating a burning chair: including neighbor readings in the decision. The results when only own values are used are plotted on the first y-axis. Including the neighbor values is plotted on the second y-axis.	74
5.10	Simulating burning oil: including neighbor readings in the decision. The results when only own values are used are plotted on the first y-axis. Including the neighbor values is plotted on the second y-axis.	75
5.11	Membership functions for the Mass Concentration input linguistic variables.	76
5.12	Crisp value simulation: a) Frying margarine b) Broiling hamburgers.	76
5.13	Fuzzy value simulation: a) Frying margarine b) Broiling hamburgers.	77
5.14	Simulating a burning chair with a reduced rule-base. The results when the full rule-base is used are plotted on the first y-axis. Using the reduced rule-base is plotted on the second y-axis.	78
6.1	The main components of the RTA framework are the automatic code generator, the automatic test generator, and the test execution support.	85
6.2	The radio transition in a MEDAL model can be used by an automated code generator to determine the boundaries between the different node executables that should be generated. . .	90
6.3	The steps of the code generation process for MEDAL. The MEDAL model is translated into a script, which is then used by the code generator to produce the code for the nodes in the system. . .	90

6.4	A script specifying the MEDAL model of an application describes all arcs, places, and transitions with their types and attributes.	91
6.5	A fire detection application uses smoke and temperature readings to determine the presence of fire. The MEDAL model of this application logic concisely and unambiguously depicts the application logic.	93
6.6	RTA and HM achieve similar false negative rates.	97
6.7	RTA requires 50%-70% fewer repairs than HM.	98
6.8	On average, RTA uses 33% less messages than HM.	98
6.9	When the failures introduced in the system are location errors, the RTA system missed 75% fewer fires than the HM system on average.	99
6.10	RTA reduces the number of maintenance dispatches to only 0.3%-33.9% of HM.	100
7.1	Failure detection accuracy for a “movement” non-fail-stop node failure introduced in a smart-home deployment. Correlation-based approaches cannot achieve more than 80% failure detection accuracy even 4 days after the failure has occurred.	104
7.2	SMART has two components: offline and run time. Preliminary offline analysis of node failure severity is performed and multiple classifiers are trained. The results of this analysis together with the classifiers are used at run time to detect node failures and determine if maintenance is needed.	105
7.3	Preemptively training classifiers for the occurrence of failures by <i>holding one sensor out</i> from the training set allows us to build failure-aware systems. Classifier A is trained on all nodes. Classifiers B and C are trained by holding node 1 and node 7, respectively.	107
7.4	Multiple F-score vectors are used to detect when an application-level failure occurs in the system and to identify which node has caused the failure.	108
7.5	The severity of a sensor node failure is very strongly correlated with the level of redundancy in the system. Therefore, because of the low node redundancy in the kitchen, if the star-shaped node fails, the accuracy of detecting <i>cooking</i> activities would be severely impacted.	110
7.6	Detecting non-fail-stop node failures of the kitchen nodes in the WSU house for both NB and HMM. SMART achieves more than 85% accuracy in detecting a severe “stuck at” non-fail-stop failure and about 80% accuracy in identifying which one of the 8 kitchen sensor nodes has caused the failure. It also achieves more than 95% accuracy in detecting a failure caused by rotating the kitchen motion sensor towards the adjacent living room in the house from the WSU dataset.	113
7.7	Distribution for number of maintenance dispatches for the all kitchen activities the WSU house, House A, and House B. Compared to a baseline where maintenance has to be dispatched every time a node failure occurs, our approach repairs only the severe failures and thus decreases the number of maintenance dispatches by 55%.	114
7.8	Mean time to failure distribution for the WSU house <i>preparing breakfast</i> . We assume that a new sensor failure occurs after each time unit. The average MTTF for NB and HMM are 4.5 and 5.5 time units respectively.	116
7.9	Average activity recognition accuracy improvement for the activity <i>preparing breakfast</i> in the WSU house. Compared to the baseline NB and HMM classifiers trained with all nodes, the NB and HMM preemptively trained for node failures improves the activity detection accuracy in the presence of failures.	117
7.10	Average activity recognition accuracy improvement for all kitchen activities in the WSU house. Using ensembles of classifier instances trained for failures instead of a single instance trained on all nodes significantly improves the activity recognition accuracy under failures.	118
7.11	Relationship between the failure detection accuracy of a node and how frequently this node is being used for activities <i>prepare breakfast</i> and <i>prepare dinner</i> for House A. The majority of nodes in this dataset have low usage ratios, which results in low failure detection accuracy even for the nodes with high usage ratio when all nodes are considered by the failure detection mechanism.	119

7.12 Detecting “stuck at” failures for the important nodes participating in activities <i>prepare breakfast</i> and <i>prepare dinner</i> in House A. The failure detection accuracy for the important nodes for an activity increases when only the failures of these nodes are being considered. SMART achieves about 80% accuracy in identifying which of the important nodes has failed.	120
--	-----

Part I

Introduction

Chapter 1

Introduction

WIRELESS sensor networks (WSNs) are being introduced to a growing number of applications. Smart home sensor network applications could monitor a resident's patterns and habits to help manage home automation, energy efficiency, and home security. Health monitoring applications could assist with monitoring the daily activities of patients and reporting the occurrence of abnormal patient behavior. Military sensor network applications could be used for surveillance and detection of invading enemy forces. Regardless of the specific application, the sensor network should be able to detect or predict particular events of interest. However, the extremely large spectrum of applications results in a very diverse pool of events to be detected. These events can range from activities of daily living to emergency situations, such as fires and explosions; from a resident coming home from work to a burglar entering the house at night; from poor air quality and insufficient level of light, to sensor node and application failures. The detection of these events is based on sensor fusion from many sensor nodes deployed in constantly changing complicated environments. Sensor data may be missing, wrong, or out of date. Consequently, an event detection service must:

- be accurate and minimize false alarms;
- run in real-time;
- provide a sufficient level of robustness;
- fundamentally recognize location.

All these features make building event services for WSNs notoriously challenging.

The research presented in this dissertation focuses on developing techniques and approaches that enable robust real-time event services for sensor network applications. There are three main areas that we have

focused on: event specification, event detection, and providing robustness and fault-tolerance for event-driven sensor network applications.

1.1 Problem statement

Event specification

Providing effective approaches for event description in a sensor network remain a challenge. Currently, the majority of techniques rely on using SQL-like syntax. SQL-based approaches, however, are not very suitable for describing sensor network events since they fail to express a number of essential properties specific to sensor network events, including data dependency, collaborative decision making, spatial and temporal correlation among different types of sensors, and probabilistic events.

Event detection

The majority of current event detection approaches rely on using precise, also called *crisp*, values to specify WSN events. However, crisp values cannot adequately handle the often imprecise sensor readings. Therefore, improving the accuracy of event detection in sensor networks continues to be a challenging problem.

Providing robustness and fault-tolerance

Continuous and reliable operation of WSNs is extremely difficult to guarantee due to hardware degradation and environmental changes, which can cause operating conditions that were impossible for the original system designers to foresee. This is particularly true for applications that operate over long time durations, such as commercial building monitoring, smart home deployments, and health-monitoring applications. Wireless noise and interference may change dramatically as new wireless technologies are developed and deployed in or near a building, and sensor readings and network topology may change as the occupancy, activities, and equipment in a building evolve over time. Node failures that severely affect the behavior of sensor network applications, if not detected on time, might lead to serious consequences. Therefore, being able to 1) verify at run time that an application is able to function correctly according to its high-level specification, 2) detect the occurrence of sensor node failures, and 3) maintain sufficient event detection accuracy even in the presence of failures, are topics that need to be addressed.

1.2 Contributions

1.2.1 Event specification

We have developed a formal event description language, which is an enhanced Petri net. This language, coMpaCT Event Detection and Analysis Language (MEDAL), addresses key aspects of sensor networks, such as temporal control, spatial constraints, heterogeneity, and probability issues. MEDAL combines features from Stochastic, Timed, and Colored Petri nets. This allows it to model sensor network properties, such as collaborative decision making, temporal and spatial dependencies, and geographic control. The graphical support inherited from Petri nets makes the application models created with MEDAL easy to understand and accessible to a wide range of users. Therefore, MEDAL can be used as an interface between people who register events (e.g. domain experts and scientists who recognize the characteristics of events) and the sensor network designers (mostly computer engineers responsible for designing protocols to control sensors according to certain temporal and spatial specifications). In this sense, MEDAL provides network designers with well-defined formal requirements for building the sensor network. We have used MEDAL to model the logic of a variety of sensor network applications, including a number of existing deployments, such as a volcano monitoring application [1], rural fire detection application [2], and flood detection application [3]. Our experience has shown that MEDAL has the expressive power to model a broad array of applications running on both homogeneous and heterogeneous networks.

We have also used MEDAL to model real-time data stream applications. We show how formal data stream modeling and analysis can be used to better understand stream behavior, evaluate query costs, and improve application performance. We use MEDAL to model the data stream queries and the Quality-of-Service (QoS) management mechanisms of a data stream management system. MEDAL's ability to combine query logic and data admission control in one model allows us to design a single comprehensive model of the system. This model can be used to perform a large set of analyses to help improve the application's performance and QoS.

1.2.2 Event detection

Since *crisp* values are not very suitable for event description and detection, we have investigated whether using *fuzzy* logic might help increase the accuracy of event detection applications. In contrast to crisp logic, fuzzy logic has a number of properties that make it appropriate for describing events in WSNs:

- It can tolerate unreliable and imprecise sensor readings.

- It is much closer to our way of thinking than crisp logic - for example, we think of fire as an event described by high temperature and smoke, rather than an event characterized by temperature above 55 °C and a smoke obscuration level above 15%.
- Compared to other classification algorithms based on probability theory, fuzzy logic is much more intuitive and easier to use.

Our experiments with real-world fire data have shown that using fuzzy values results in more accurate event detection since fuzzy logic is more resilient to sensors getting temporarily “confused”.

To maintain sufficient event detection accuracy even in the presence of unreliable and imprecise sensor readings, the fuzzy logic event detection system is instrumented with spatial and temporal semantics. This allows the event detection process to employ readings from multiple sensors and/or readings over some period of time. The spatial and temporal semantics are included by augmenting the fuzzy logic system with additional spatial and temporal linguistic variables. This significantly improves the event detection accuracy of the application. However, it comes at a price - the number of rules that describe the behavior of the fuzzy logic system increases exponentially to the number of linguistic variables. A large rule-base is not desirable since sensor nodes have limited memory and constantly traversing a large rule-base might significantly slow down the event detection process. To alleviate this problem, we have designed a set of techniques that can decrease the size of a rule-base by more than 70% without reducing the event detection accuracy of the application.

1.2.3 Robustness to node failures

Our work in this area has branched into two projects: *Run time assurance* and *SMART*.

Run time assurance

The run time assurance (RTA) methodology validates at run time that a WSN application functions correctly, irrespective of any changes in the operating conditions that might have occurred since it was originally designed and deployed. The basic approach is to use program analysis and compiler techniques to facilitate automated testing of the system at run time. As part of this project we have developed techniques to automatically generate a set of input/output tests based on the MEDAL model of the WSN application. The test inputs are supplied to the WSN at run time, either periodically or by request. The WSN performs all computations, message passing, and other distributed operations required to produce output values and actions. These are compared to the expected outputs. This testing process produces an end-to-end

application-level validation of the critical application logic. We have also designed test reduction techniques which allow us to drastically decrease the size of the test suites.

The end-to-end application-level tests used for RTA have two key advantages over the tests of individual hardware components used for health monitoring. First, RTA has fewer false positives, since it does not test nodes, logic, or wireless links that are not necessary for correct system operation. Therefore, RTA produces fewer maintenance dispatches than health monitoring systems. Second, RTA has fewer false negatives. A network health monitoring system will only validate that all nodes are alive and have a route to a base station, but will not test more subtle causes of failure, such as topological changes or clock drift. In contrast, the RTA approach tests the ways that an application may fail to meet its application-level requirements because it uses end-to-end tests. Network health monitoring improves system reliability by detecting certain types of failures. However, it stops short of actually validating correct system operation. In contrast, the goal of RTA is to provide positive affirmation of correct application operation. Our results indicate that RTA performs much better than health monitoring in identifying application-level failures and almost just as well in identifying low-level failures.

SMART

The motivation behind this work is that a number of studies have found that many of the inexpensive sensor network home installations suffer from numerous failures. A significant number of these failures are non-fail-stop failures, where the sensor does not completely fail. Instead, it continues to report values, but the meaning of the values changes or becomes invalid. Several techniques have previously been used to detect non-fail-stop failures, but they are designed for homogeneous, periodic, and continuous-valued sensors. This makes them hard to generalize to the heterogeneous, binary, and event-triggered sensor suites often used in smart home applications.

Similarly to RTA, the Simultaneous Multi-classifier Activity Recognition Technique (SMART) uses application-level semantics to detect, assess, and adapt to sensor failures at run time. The basic insight underlying SMART's failure detection is that sensors monitoring the same activities in the physical world typically have correlated values. SMART uses a classifier ensemble to identify the degree to which different sensors indicate the same ongoing activity. Our evaluation shows that SMART can accurately identify non-fail-stop application-level failures. In addition, since it only repairs the nodes whose failure affects the application behavior, SMART reduces the number of maintenance dispatches. Due to its ability to adapt to application-level failures at run time, SMART increases the mean time to failure of the application and improves the activity recognition accuracy under node failures.

1.3 Dissertation organization

The rest of the dissertation is organized as follows:

- *Chapter 2* discusses the state of the art research related to our work.
- *Chapter 3* describes the coMpack Event Description and Analysis Language (MEDAL), we designed for specifying sensor network events.
- *Chapter 4* discusses how MEDAL can also be used to model and analyze real-time data stream queries and stream management mechanisms.
- *Chapter 5* introduces how fuzzy logic can be used to help improve the detection accuracy of sensor network events.
- *Chapter 6* describes the run time assurance methodology and its components. We discuss in detail the automated test generation, the test reduction techniques we have designed, and our results of comparing RTA to an existing health-monitoring technique.
- *Chapter 7* introduces the Simultaneous Multi-classifier Activity Recognition Technique (SMART), which focuses on detecting sensor node failures, maintaining sufficient event detection accuracy even in the presence of node failures, and minimizing the number of necessary maintenance dispatches.
- *Chapter 8* concludes the dissertation, brings up some of the limitations of this work, and discusses a number of possible directions for future work.

Chapter 2

State of the art

IN this chapter we discuss the state of the art research in a number of areas related to the work in this dissertation, including, modeling sensor network events and data streams, stochastic methods, fuzzy logic, sensor network testing and debugging, fault tolerance and fault identification techniques, hardware verification, classifiers, and classifier ensembles.

2.1 Event specification

2.1.1 Modeling sensor network events

The work that uses SQL-like primitives to describe sensor network events varies a little in semantics [4, 5, 6, 7]. General SQL primitives are employed in [5] and [6] to define events in sensor networks. However, with this approach, events can only be defined by predicates on sensor readings connected with “AND” and “OR” with very simple temporal and spatial constraints. Madden et al. extend the SQL primitives by incorporating streaming support, where the desired sample rate can be specified [7]. Li et al. propose defining events using a sub-event list and confidence functions in SQL [4]. Nevertheless, because of its inherent limitations, SQL is not very suitable for formally describing events in sensor networks. Some of those limitations include:

- *Data dependency and correlation among different types of sensors in heterogeneous sensor networks.*
- *Collaborative decision making:* Sensor networks are distributed, concurrent, and asynchronous, and detecting events usually requires spatial and temporal composition of ad-hoc sensor readings.

- *Modeling probabilities*: Individual sensor nodes are unreliable, which results in non-deterministic event detection. In order to tolerate such non-determinism, sensor network events need to be defined using probabilistic models. SQL has no explicit support in this regard.
- *Hierarchical model*: Most WSN applications need to form groups for data aggregation and event detection. Some events are detected at the node-level, others are global events detected by cluster-head nodes or even base stations. Events could also form hierarchies. However, SQL-like languages do not naturally present a hierarchical model for event structure.

Worboys also attempts to define events in sensor networks [8]. He proposes an object-oriented model with integrated timing and location properties to represent events. The approach was initially designed for modeling generic events, including social and economic ones. Therefore, it does not support unique characteristics of sensor networks, e.g. there is no consideration of different sensor types or interactions.

Few formal methods have been used for event specification in sensor networks. However, formalized approaches for event descriptions and compositions are widely studied in other areas. A survey on formal methods for specification and analysis shows that most of the popular approaches are based on theoretical models, such as finite state machine, timed automata, process algebra, and Petri nets [9]. Some widely used methods such as SDL [10], SPIN (Simple Promela INterpreter) [11], and Estelle [12] are based on finite state machines (FSM). Unfortunately, FSM approaches have difficulty describing hierarchical modeling. FSMs, however, can be augmented to Timed Automata which are the mathematical foundations for many specification methods, including UPPAAL [13], Kronos [14], and HyTech [15]. However, both FSM and Timed Automata inherit the limitations of finite state machines, such as state explosion.

There are a number of approaches based on process algebra and composition logic. Composition of events together using the concept of Event-Condition-Action rules is presented in [16]. In the paper, the authors propose HiPAC event algebra for database systems. A few other event specification approaches for database systems are introduced in other papers [17, 18, 19]. All of these methods are designed for the definition and composition of events in active database systems. In addition, various composition algebra approaches have been proposed to handle events in a distributed system [20, 21, 22]. However, since these approaches were mainly developed for database systems, some important characteristics of sensor networks, such as sensing activities or spatial and temporal properties, have not been addressed.

A large number of papers and books has been published on Petri nets since their introduction in 1962 [23]. Places and transition nets were first formally defined by Jantzen et al. [24], high-level Petri nets were introduced by Genrich et al. [25], and Colored Petri nets were proposed by Jensen [26] and Kristensen et

al. [27]. Later on, Timed Petri nets and Stochastic Petri nets were proposed by DiCesare et al. [28] and Tremblay et al. [29].

For the first time Petri nets were used to describe events in WSNs by Jiao et al. [30]. In this work the authors introduce a Sensor Network Event Description Language (SNEDL). As a formal method, SNEDL is based on Petri nets, which allows it to rigorously specify events in sensor networks. An important advantage of this specification rigor is that it prevents ambiguity. The event specification language that we have developed, MEDAL, combines SNEDL’s event description abilities with a more compact syntax without compromising the level or rigor of the specification. In addition, MEDAL expands the feature set and modeling capabilities of SNEDL in a number of important directions.

2.1.2 Modeling data streams

The majority of work on data streams uses SQL or SQL-like semantics to define stream queries [31, 32]. Babcock et al. use standard SQL to model stream queries and extend the expressiveness of the language to allow the specification of sliding windows [32]. However, similarly to how SQL is not suitable to model WSN applications, it fails to express a number of essential characteristics that are also typical for data stream processing applications, including collaborative decision making, data dependency and correlation among different data streams, and non-deterministic behavior.

Another method for modeling stream queries was proposed for the Aurora system [33]. Aurora uses a graphical “boxes and arrows” interface for specifying data flow through the system. Compared to a declarative query language, this interface is more intuitive and gives the user more control over the exact series of steps by which the query answer is obtained. However, similarly to SQL-based solutions, this approach lacks the ability to model probabilities, data dependencies and correlation, and collaborative decisions.

2.2 Event detection

2.2.1 Stochastic methods

There is a long history of using stochastic formalisms in different WSN applications. Bayesian classifiers and hidden Markov models have been extensively used in activity recognition [34, 35] and decision fusion [36, 37]. Dempster-Shafer evidence theory has been applied to intrusion detection [38], sensor fusion [39, 40], and assisted living applications [41]. Probabilistic context free grammars have been used to solve problems such as inferring behaviors [42] as well as movement and activity monitoring [43, 44].

2.2.2 Fuzzy logic

Fuzzy sets and logic were introduced by L. Zadeh in 1965. Numerous fields have taken advantage of their properties since then. In WSNs, fuzzy logic has been used to improve decision-making, reduce resource consumption, and increase performance. Some of the areas it has been applied to are cluster-head election [45, 46], security [47, 48], data aggregation [49], routing [50, 51], MAC protocols [52], and quality of service(QoS) [53, 54]. However, not much work has been done on using fuzzy logic for event description and detection. Liang et al. [55] propose to use fuzzy logic in combination with double sliding window detection, to improve the accuracy of event detection. However, they do not study the effect of fuzzy logic alone or how using the spatial or temporal properties of the data could influence the classification accuracy.

In D-FLER [56] fuzzy logic is used to combine personal and neighbors' observations and determine if an event has occurred. The results show that fuzzy logic improves the accuracy of event detection. The use of fuzzy values allows D-FLER to distinguish between real fire data and nuisance tests. However, the approach used in D-FLER does not incorporate any temporal semantics. In addition, all of the experiments last only 60 seconds after the fire ignition. Therefore, the authors only analyze the accuracy of D-FLER in the presence of fire, but fail to evaluate the precision of the approach, when there is no fire.

2.3 Robustness to sensor node failures

2.3.1 Sensor network testing

Although testing has always been a major part of software development, a very limited amount of work has been done in the area of testing WSN applications. This is partially due to a few characteristics of WSN applications, such as operating in concurrent, event-based, and interrupt-driven manner, which considerably complicates the development of code representation. Nguyen et al. [57] proposed application posting graphs to represent behaviors of WSN applications. Regehr [58] designed a restricted interrupt discipline to enable random testing of nesC programs. Lai et al. [59] studied inter-context control-flow and data-flow adequacy criteria in nesC programs. However, all previous work is intended for testing applications prior to deployment when the size of the test suite is not as critical. Therefore, WSN-specific approaches for decreasing the size of the number of necessary tests have not been considered until now.

2.3.2 General fault tolerance techniques

There is a great array of fault tolerance and reliability techniques developed over the last 50 years many of which have been applied to WSNs [60, 61, 62, 63]. We expect that any WSN that must operate with

high confidence will utilize many of these schemes. However, most existing approaches, such as eScan [64] and CODA [65], aim to improve the robustness of individual system components. Therefore, it is difficult to use such methodologies to validate the high-level functionality of the application. Similarly, self-healing applications [66, 67, 68], although attempting to provide continuous system operation, are not capable of demonstrating adherence of the system to key high-level functional requirements.

2.3.3 Sensor network debugging

Debugging WSN applications is a complicated process and many different approaches exist. Marionette [69] and Clairvoyant [70] are source-level debuggers allowing access to source-level symbols. MDB [71] supports the debugging of macroprograms. SNTS [72], Dustminer [73], and LiveNet [74] use overhearing to gain visibility into the network operations. Some debugging approaches are based on invariants [75], others attempt to use data mining to discover hard to find bugs [76]. EnviroLog [77] uses a *record and replay service* where it stores and replays the I/O on the sensor nodes. However, all of these debugging mechanisms are either used prior deployment or in a *post mortem* manner, where data about the application is collected and then analyzed offline. Therefore, these techniques do not provide a way to monitor and analyze the application behavior at run time.

2.3.4 Sensor network hardware verification

Several techniques have been developed for fault detection and hardware verification in sensor networks. Health - monitoring systems, such as LiveNet [74], Memento [78], and MANNA [79], employ sniffers or specific embedded code to monitor the status of a system. Sympathy [80] detects node failures based on periodic data collection and neighbor sniffing on a routing tree. These applications are effective for low-level system failures and fail-stop sensor faults, the techniques we have developed, RTA and SMART, can use these systems to detect failed nodes. However, these systems are not designed to detect failures that affect the high-level application behavior or non-fail-stop sensor faults.

2.3.5 Sensor network fault identification

Recently, new techniques have been developed to identify non-fail-stop sensor faults. Ni et al. provide a taxonomy of frequently observed sensor faults, including “*stuck at*” faults and *noise* faults, and provide a set of features that can be used to detect these faults [81]. Suelo [82] detects faults by identifying outliers in a feature space of the data streams, and can learn to improve fault detection with the help of human expert assistance. Suelo uses a general feature space that can be applied to a wide range of sensors, in contrast

to RTA and SMART which use application-level features. The reputation-based framework proposed by Ganeriwal et al. [83] also uses outliers and correlation between neighboring nodes to detect non-fail-stop sensor faults. This approach is suitable for applications where there is a constant flow of information from nodes of the same type that are located close to each other. However, building and maintaining a community of trust is harder for smart home applications, where sensors are heterogeneous, discrete, and event-driven.

2.3.6 Classifier ensembles

The idea of using multiple classifiers and dynamically selecting the most accurate one has been explored in pattern recognition and neural networks. Partridge and Yates proposed a number of techniques that exploit heuristic rules for choosing classifier groups [84]. Additional techniques were proposed by Roli and Giacinto [85, 86]. These techniques could be incorporated into SMART to further increase the accuracy of activity recognition under failures.

An ensemble of classifiers has been used to detect anomalous nodes in a body sensor network [87]. The results show that detecting anomalous behavior and replacing the anomalous values with probabilistically generated meaningful ones improves the activity recognition accuracy. However, the authors do not analyze the cause or the severity of the anomalies. They also use a classifier fusion approach, which assumes that the majority of the classifiers will not be affected by the anomaly. However, in a smart home application, the failure of a single important node can affect all classifiers trained to recognize a particular set of activities. SMART addresses this by analyzing the relative change in the behavior of all classifiers, which provides better understanding about the root of the failure, as well as the impact of this failure on the application.

Part II

Event specification

Chapter 3

coMpack Event Description and Analysis Language (MEDAL)

WIRELESS sensor networks have received a lot of attention due to a number of applications they have been used for, including environmental monitoring, healthcare, military surveillance, and smart home deployments. However, the types of events these applications can currently detect are restricted. There is a need for a description language that can incorporate the knowledge of sensing with event definitions. A formalized description language can serve as an effective interface between people who register events, e.g. application semantics experts, and sensor network designers. Such a language could also make application requirements more clear and remove the ambiguity seen in application specifications.

The majority of work that uses event description for sensor network applications employs SQL or SQL-like semantics to describe events [4, 5, 6, 7, 88]. However, Franklin [89] points out that SQL-like semantics are not always suitable for sensor networks because of their lack of collaborative decision making and other fundamental features. As mentioned earlier in Chapter 2, SQL fails to express a number of essential characteristics specific to sensor network events, including:

- data dependency and correlation among different types of sensors in heterogeneous sensor networks;
- collaborative decision making;
- modeling probabilities;
- hierarchical model.

The following two examples demonstrate some of the drawbacks of using SQL for event description in sensor networks. In the first scenario a sensor network is deployed to detect explosions. The nodes in the

network are equipped with three types of sensors: sound, light, and temperature. If an explosion event is simply a combination of positive readings from the three types of sensors, using the notation of [4], an explosion event can be represented as follows:

```
INSERT INTO EventList Explosion (Event-ID, SubEventSet, Spatial Resolution ...)
VALUES (0001, SubEventSet ...)
WHERE SubEventSet is SubEventSet = (Sound, Light, Temperature,
    Confidence Function: 0.3 x Sound + 0.3 x Light + 0.4 x Temperature >= 100 ...)
```

All three, high temperature, loud sound, and strong light, should be present at the time of explosion. Therefore, if a light sensor detects strong light but there are no accompanying abnormal readings from the temperature and sound sensors within some time interval from the light reading, the system should not classify this as an explosion. To achieve more accurate explosion detection we could define sensor reading validity intervals. For example, we might specify that sound readings are valid for 15 seconds, light readings - for 10 seconds, and temperature readings - for 30 seconds. However, the above SQL-like notation cannot be used to specify events that require similar or more complicated temporal constraints.

Figure 3.1 illustrates the second scenario - probabilistically occurring events. A set of readings from sensors $S1$ and $S2$ could indicate both event A with a 30% probability and event B with a 70% probability. Events A and B themselves could be sub-events of higher level events. In Figure 3.1, event A is such a sub-event. Since the nature of a sensor network is distributed and noisy, random processes and probabilistic models are especially meaningful. Therefore, this example is much more realistic than a simple deterministic model. Since SQL-like languages lack the ability to capture random processes of event systems, they are not very suitable to describe probabilistic events for WSN applications.

In contrast to SQL-like semantics, Petri nets are well accepted as a model to describe systems with distributed, concurrent, asynchronous, and non-deterministic nature. Petri nets exhibit a number of properties that make them particularly useful for event description in WSN applications:

1. Extended Petri nets (including Colored Petri nets, Timed Petri nets, and MEDAL) are equivalent to Turing Machines in computation power, and thus can handle non-determinism [90].
2. Geographic information is crucial to events in sensor networks, since many events are related to a specific location and, in addition, may have other spatial properties. Features inherited from Colored Petri nets allow us to incorporate spatial properties into the tokens, thus handling geographical constraints for events in sensor networks.

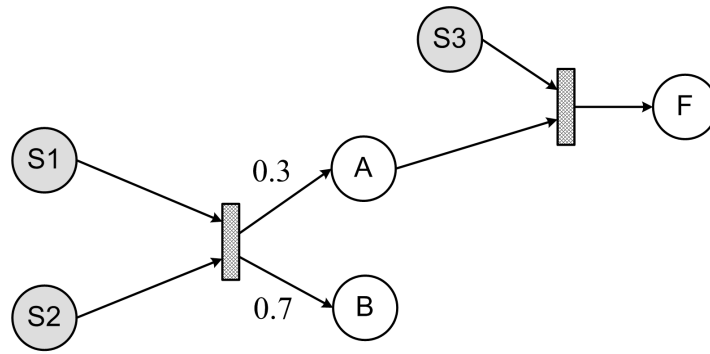


Figure 3.1: Probabilistic events scenario.

3. Features inherited from Stochastic Petri nets provide an advantage over other specification methods in terms of performance evaluation ability.
4. Petri nets naturally have graphical support which is convenient for users who are not thoroughly knowledgeable about probability theory.

We have used Petri nets as the foundation for an event description language that is specifically designed for modeling sensor network applications. This language, coMpack Event Description and Analysis Language (MEDAL) [91], addresses key aspects of sensor networks, such as temporal control, spatial constraints, heterogeneity, and probability issues. MEDAL combines features from Stochastic, Timed, and Colored Petri nets, which allows it to model sensor network properties, such as collaborative decision making, temporal and spatial dependencies, and geographic control. We have also extended MEDAL to specify additional key sensor network properties, such as communication, actuation, and feedback control.

The main contributions of the work presented in this chapter are:

1. Introducing a new event description language MEDAL, based on Petri net theory.
2. Extending MEDAL to model sensor network properties, such as communication, actuation, and feedback control.
3. Introducing the use of MEDAL for real-time and safety analysis of WSN applications.
4. A comparison between MEDAL and other approaches that have been used to describe sensor network events.
5. A proof of concept - we describe our experience using MEDAL to model a beef monitoring sensor network application.

6. Using MEDAL to model three existing event-based WSN applications: volcano monitoring [1], rural fire detection [2], and flood detection [3].

3.1 Introduction to Petri nets

A Petri net is a bipartite directed graph with two types of nodes: *places* and *transitions*. Places represent conditions or system state and are usually depicted using circles. Transitions are used to represent events that may occur and are modeled using bars or rectangles. The places and transitions in a Petri net are connected using directed lines, called *arcs*. An arc is either from a place to a transition or from a transition to a place. It never connects nodes of the same type, i.e. a place to a place or a transition to a transition.

A *marking* assigns to each place in the Petri net a non-negative number k . For a place marked with a value k , we say that there are k *tokens* in that place. Graphically a place that contains k tokens is displayed with k black dots in it.

A Petri net can be formally defined as a 5-tuple [92], $PN = \{P, T, A, W, M_0\}$, where:

1. $\mathbf{P} = \{p_1, p_2, \dots, p_n\}$ is the finite set of places.
2. $\mathbf{T} = \{t_1, t_2, \dots, t_m\}$ is the finite set of transitions.
3. $\mathbf{A} \subseteq (P \times T) \cup (T \times P)$ is the set of arcs connecting places to transitions and transitions to places.
4. $\mathbf{W}: F \rightarrow \{1, 2, 3, \dots\}$ is the weight function. In some types of Petri nets all weights are assigned to 1.
5. $\mathbf{M}_0: P \rightarrow \{0, 1, 2, \dots\}$ is the initial marking of the Petri net.

A node in a Petri net can either be a place or a transition. Formally, $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

Each place has a number of input and output transitions. The input set of transitions for a place p_i can be denoted as $\bullet p_i = \{t_j | a(t_j, p_i) \in A\}$. Similarly, the set of output transitions for a place p_i is represented by $p_i \bullet = \{t_j | a(p_i, t_j) \in A\}$. A place with no input transitions, i.e. $\bullet p_i = \emptyset$, is called a source place. A place with no output transitions, i.e. $p_i \bullet = \emptyset$ is called a sink place.

A transition has a certain number of input and output places, which represent the pre-conditions and the post-condition of the event modeled by this transition. Given a Petri net, the input set of places for a transition t_i can be denoted as $\bullet t_i = \{p_j | a(p_j, t_i) \in A\}$. Similarly, the set of output places for a transition t_i is denoted as $t_i \bullet = \{p_j | a(t_i, p_j) \in A\}$. At any time, a transition is either *enabled* or *disabled*. A transition t_i is considered enabled if each place in its input set $\bullet t_i$ has at least one token. An enabled transition can fire. When a transition t_i fires, it consumes a token from each place in $\bullet t_i$ and deposits a token into each place in

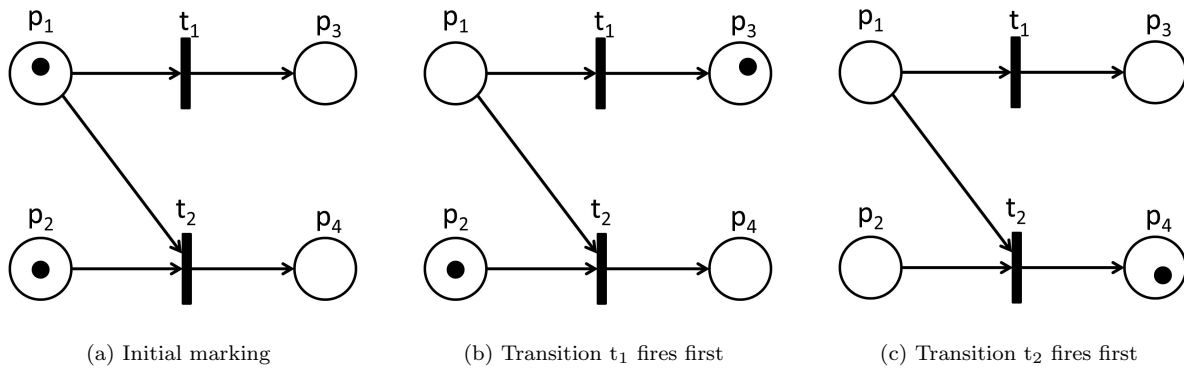


Figure 3.2: An example Petri net consisting of four places and two transitions. In the initial marking of the Petri net place p_1 and place p_2 each have a token, which enables both transitions. Depending on which transition fires first, the Petri net could end up in 2 different states.

$t_i \bullet$. This movement of tokens, which changes the marking of the Petri net, is used to simulate the dynamic behavior of a system.

Figure 3.2a shows an example Petri net that contains four places, p_1 , p_2 , p_3 , and p_4 , and two transitions, t_1 and t_2 . The marking of this Petri net is such that both transitions are enabled and can be fired. However, the firing of one of the transitions will disable the other, resulting in two different firing sequences. Transitions t_1 and t_2 are called *conflicting transitions* since they are enabled by common input places and firing one transition will result in disabling the other. The choice of which transition will fire among a set of conflicting transitions is non-deterministic.

Figure 3.2b shows the state of the Petri net after transition t_1 has fired. In this scenario, the firing of transition t_1 consumes the token in place p_1 and places a token in place p_3 . At this point, since there are no more tokens in place p_1 , transition t_2 cannot fire. Figure 3.2c shows the state of the Petri net after transition t_2 has fired. In this scenario, the tokens in places p_1 and p_2 are consumed by transition t_2 and a new token is deposited in place p_4 . In both Figure 3.2b and Figure 3.2c, there are no more transitions that can fire. A Petri net, where no transition can fire, is said to be not alive.

3.2 MEDAL

3.2.1 Semantics

The MEDAL description of an event detection sensor application is a 7-tuple structure: $F = (P, T, A, \lambda, \beta, H, L)$ where:

1. **P** is the set of all places. $P = S \cup E$, where S represents the places for sensing, and E represents the places for higher level events. We explain S and E in detail later in the section when we discuss sensor event abstraction. In a MEDAL diagram, similarly to Petri nets, places are depicted as circles (place E in Figure 3.3). To distinguish between sensor events and higher level events, the places for sensing are represented by interrupted circles (places T , L , and A in Figure 3.3).
2. **T** is the set of all transitions. Transitions model various types of actions and are represented by vertical bars (transitions $T1$, $T2$, $T3$, and $T4$ in Figure 3.3).
3. **A** is the set of arcs. $A = I \cup O$, where I is the set of input arcs entering a transition (pre-arcs), and O is the set of output arcs leaving a transition (post-arcs).
4. λ is the probability/weight function for the arcs and $\lambda : A \rightarrow [0, 1]$. If f is a post-arc from transition T to place B , then $\lambda(f) = p$ means that after T is fired, the probability that the resulting token enters place B is p . For a pre-arc, if $\lambda(f) = p$ and a token has a capacity c then its capacity after passing through arc f will be $c \times p$. With λ , MEDAL adopts features from Timed and Stochastic Petri nets and can model probabilistic problems.
5. β is the time guard function, $\beta : T \rightarrow \cup^*(r1, r2)$, where $r1 \leq r2 \in \mathbb{R}$. For a transition it means that the transition can only fire during the union closure of the given ranges. $\beta(T) = (a1, a2) \cup (a3, a4)$ indicates that transition T can only fire during interval $(a1, a2)$ or $(a3, a4)$, where $a1 \leq a2$, $a3 \leq a4$. β can also be specified using event incidents. $\beta : T \rightarrow^* (E1, E2)$ means that transition T can only fire between event $E1$ and event $E2$. For a post-arc of a transition, β shows how long it takes for this event to happen.

 β also acts as a persistency guard, i.e. we can use it to specify the amount of time a place can hold a token to participate in a transition before this token disappears. To achieve this we simply need to define β as $\beta(p) = (t1, t2)$ where $t1 \leq t2$ and $t2 - t1$ is the validity period of the token.
6. **H** is the threshold function for places and is defined as $H : P \rightarrow \mathbb{R}$. For example $H(p) = c$ means that a token can enter a specific place p only if its capacity is equal or higher than c . In a sensor network

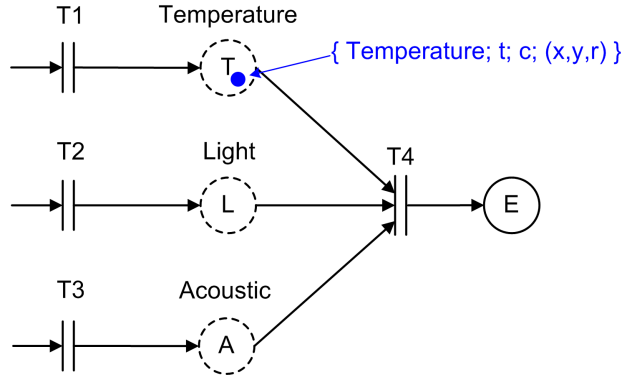


Figure 3.3: MEDAL model of an explosion detection application, which uses readings from temperature, light, and acoustic sensors to determine if there is an explosion in the monitored area.

with continuous values for sensor readings, we can simply set $H(p) = 0$ for all $p \in P$, and the token capacity can be used to store the actual value of the corresponding sensor readings. The threshold function is of particular use for cases that need a binary state (yes or no).

7. **L** is the spatial guard function for transitions, $L : T \rightarrow \mathbb{R}^+$. For example, assume there are three arcs a , b , and c entering transition T . $L(T) = r$ means that T can only fire if the locations of the tokens carried by a , b , and c are within radius r of each other. After T fires, the location associated with the new token will be determined based on the locations of the tokens from a , b , and c . This function is only one particular spatial guard function to guarantee that the sensed data is within the event radius. We can also specify spatial guard functions to support more complex spatial constructs. We discuss the spatial guard function L in more detail later when we talk about spatial logic.

Figure 3.3 shows the MEDAL model of an application designed to detect explosions. An explosion event E is characterized by specific temperature, light, and sound values. When all three values, temperature, light, and sound, exceed some predefined thresholds, transition T_4 fires and the application reports the detection of an explosion. Earlier we mentioned that SQL cannot be used to specify validity periods for the sensor readings in an application. With MEDAL we can easily define validity periods for the readings from the temperature, light, and acoustic sensors in Figure 3.3 by specifying $\beta(T) = 15s$, $\beta(L) = 10s$, and $\beta(A) = 30s$.

Tokens

The flow of sensor data and the occurrence of events in a MEDAL model are represented using tokens. Tokens in MEDAL are abstract elements defined as:

$$Token = \{Type\ tp; \ Capacity\ c; \ Time\ t; \ Location\ l\}$$

Type is used to indicate what data or event a token represents. *Capacity* corresponds to the value of the token, which indicates the confidence that an event has occurred. The capacity of a token could also hold the actual values of the sensor readings. *Time* indicates when the token was created. *Location* represents where the event occurred in the sensor network. This token definition allows us to encapsulate key aspects of the sensor readings into tokens. The information stored in the tokens can be accessed and used while the tokens proceed through a MEDAL model. For example, if a token with a time stamp t , capacity c , and location attribute (x, y, r) reaches the temperature sensor place, we can say that a temperature sensor at location (x, y) with sensing range r has registered a temperature reading with value c at time t , as shown in Figure 3.3.

Sensor events

Sensors in MEDAL are abstracted by sensor events. Sensor events are denoted as places that only take tokens from the environment. In Figure 3.3, places T , A , and L are sensor events. The number of sensor events is directly related to the types of sensors in the network. For example, if there are three types of sensors in the network, temperature, light, and acoustic sensors, then there are three types of sensor events in the MEDAL model of the application: Temperature, Light, and Acoustic. Higher level events are constructed using sensor events.

3.2.2 Temporal and spatial logic

Sensor network events are a function of when and where they occur. MEDAL addresses the need of sensor network applications for spatial and temporal semantics by incorporating both spatial and temporal logic.

Temporal logic

Temporal logic refers to the temporal guard function β . It helps specify the temporal concepts *when* and *how long* in MEDAL Petri nets. β guards all the transitions to ensure they fire only during the correct temporal intervals. Introducing β in MEDAL has practical importance because, in physical environments, some events can occur only during a particular temporal interval. For example, events that depend on sunshine can only take place during the day. In addition, β can help specify conditions such as *a transition will fire only if the input tokens have been generated within some predefined time interval*. In Figure 3.3, for example, if the generation times of the tokens entering transition T_4 are more than 30 minutes away from each other, it is more likely that the sensor values were unrelated or due to inaccurate sensor readings rather than that they were actually caused by an explosion. In cases like this, where the sensor readings fail to meet the temporal requirements of the application, the network should not report the occurrence of an event even if

the necessary number and types of tokens are present. This increases the accuracy level of the event detection application, since it prevents the system from reporting a number of false positive event detections.

Spatial logic

The geographic semantics of an application are enforced by the spatial function L . As a guard function for a transition T , L ensures that the tokens carried by T 's incoming arcs satisfy the spatial locality conditions. If $L(T) = R$, the effective radius of the higher-level event recognized by T should be equal to or smaller than R . In other words, the locations of the tokens entering T should all be within a circle of radius R in order to consider them indicative of a particular event. For example, for the application model in Figure 3.3, if the tokens entering transition T_4 are generated at a distance, such that we cannot draw a circle with radius R around the sensors that generated the readings, it is likely that the reading are not related to event E . In this way, similarly to β , L could help decrease the number of false positive event detections.

3.3 Extending MEDAL's expressiveness

MEDAL provides basic support for expressing spatial, temporal, and probabilistic semantics of sensor network applications. However, based on our experience designing MEDAL models for a number of sensor network applications, we have observed that this expressiveness is not sufficient. For example, MEDAL could not explicitly model communication, which is a central component in sensor network applications. MEDAL also lacked the support to model actuation. In addition, with the constantly increasing complexity of sensor network applications, the use of feedback control logic in WSN design is becoming more and more widespread [93, 94]. MEDAL, however, did not have the capability to model feedback control. To address these shortcomings, we have extended MEDAL's syntax to include dedicated radio transitions, inhibitor arcs, and loops. These extensions allow MEDAL to model vital WSN characteristics, such as communication, actuation, feedback control, and prioritization. These extensions are described in more detail in the rest of the section.

3.3.1 Communication

Consider a scenario where a wireless sensor network is deployed in an area to monitor and detect the occurrence of explosions. The network is heterogeneous and there are four types of nodes: temperature nodes, light nodes, acoustic nodes, and a base station. A sensor node notifies the cluster-head if it detects values that are above the predefined safety thresholds. The base station is responsible for determining if an explosion has occurred based on the data it receives from the sensor nodes. This application is very similar

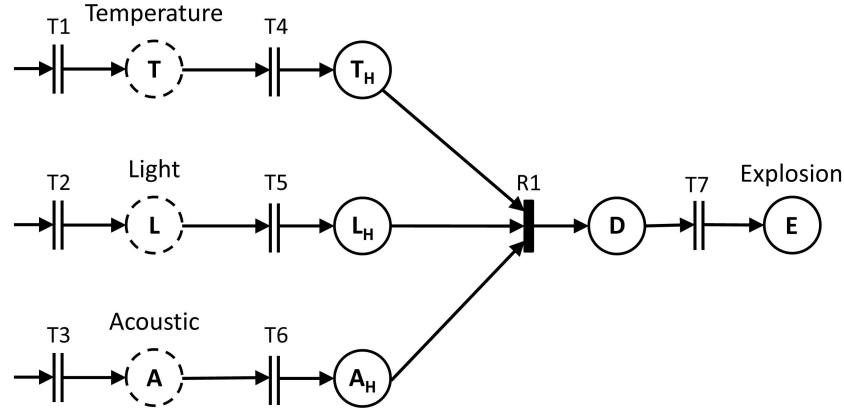


Figure 3.4: By introducing a dedicated *radio transition* (transition R1) to MEDAL, we can explicitly model the network communication.

to the explosion application described in Section 3. We are interested in detecting the same event, explosion, and the application uses the same types of sensors. The only difference is that the sensing and the decision processes occur on different nodes. However, since the MEDAL model in Figure 3.3 does not contain this important detail, it does not fully represent the application logic for the new scenario. Further, if we are to automatically generate the code for this application, we need to generate four different executables - one for each type of node. The MEDAL model in Figure 3.3, however, does not give any information about how the application execution is split among the different nodes.

We address this limitation by introducing a special *radio transition* that explicitly models the communication in a sensor network. The radio transition is represented by a solid rectangle - transition *R1* in Figure 3.4. Incoming and outgoing arcs for this transition correspond to messages that are being sent and received, respectively. The radio transition is an abstract element used to represent communication and is not associated with any particular node in the network. Therefore, it is often enough to have a single radio transition in a model. However, multiple radio transitions could be used to improve the readability of the model in the case of more complex scenarios than the one show in Figure 3.4.

3.3.2 Actuation

Actuation is an important component in many physical systems and sensor networks are not an exception. We can envision applications with requirements, such as:

- the system should remain in a particular state S unless a predefined condition C is satisfied;
- a set of steps $A - Z$ is executed only upon the occurrence of a specific event E .

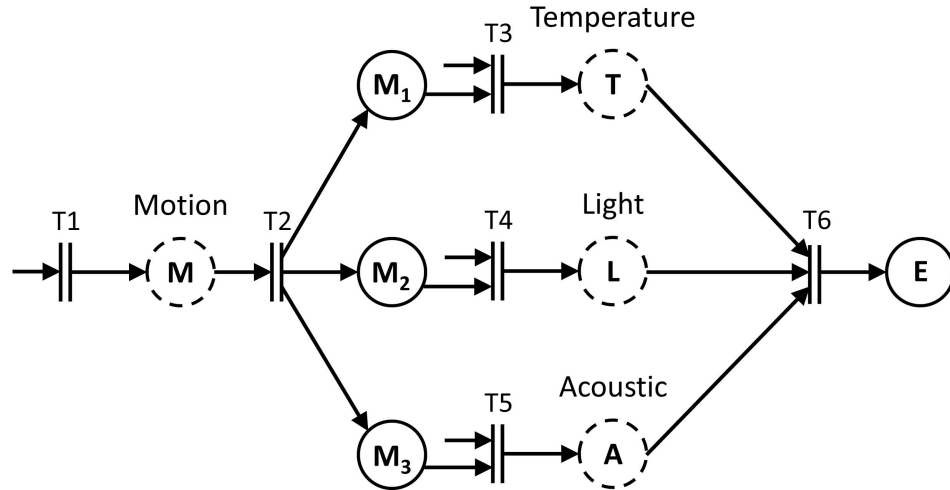


Figure 3.5: The MEDAL model of this explosion application cannot clearly specify the requirement that the temperature, light, and acoustic nodes should remain awake for 3 minutes after the motion activity has terminated.

MEDAL, however, does not provide a convenient straightforward way to model actuation. Consider the following updated explosion detection scenario. A security sensor network is deployed in a field used for explosion experiments. The area is dangerous and people are allowed to enter the explosion field only if some maintenance is needed. The sensor network is deployed to help minimize the risk for the people performing the maintenance. It is used to notify the maintenance team if an explosion occurs close to them so they can evacuate the area. Based on this scenario, the design team develops the following three requirements:

- The system should be operational when there are people in the field.
- In order to extend the lifetime of the system, the sensor nodes should be in sleep mode if there is nobody inside the dangerous perimeter.
- To improve safety, the network has to remain awake for 3 minutes after the motion activity has terminated.

This can be achieved by deploying motion sensors along the periphery of the explosion area. If the motion sensors detect activity, they wake up the rest of the network which then resumes monitoring the area for explosions.

Figure 3.5 shows a MEDAL model for this application. The actuation is represented using places M_1 , M_2 , and M_3 . If the motion sensors are not activated, no tokens can reach places M_1 , M_2 , and M_3 , which means that transitions T5, T6, and T7 cannot be activated. Therefore, the temperature, light, and acoustic nodes will remain in dormant state. One significant shortcoming of this model is that it cannot reflect the requirement that the network should stay awake for additional 3 minutes after the motion sensors have

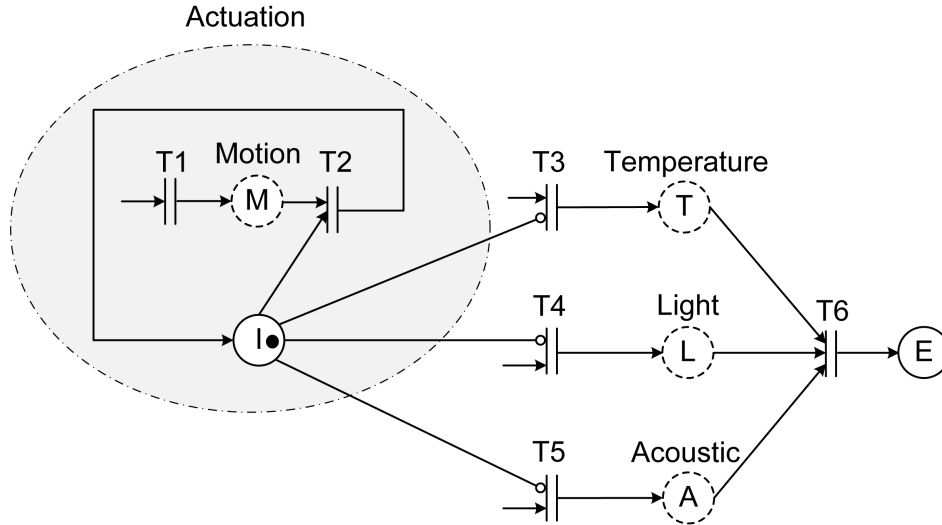


Figure 3.6: We introduce inhibitor arcs to help model actuation in MEDAL. In this explosion detection model, the temperature, light, and acoustic nodes sleep unless the token in place *I* is consumed, which occurs only if there is activity in the monitored area. Specifying $\beta = 3 \text{ min}$ for transition *T2* allows the model to address the requirement that the network should remain awake 3 min after the last motion activity has been detected.

stopped reporting activity. According to the model in Figure 3.5, as soon as the last motion token is consumed by transition *T1*, the network nodes stop sensing and can go back to sleep. The requirement that the system should remain awake for 3 minutes cannot be modeled even with the help of the time guard function β because there is not a suitable transition(s) that we can associate β with to specify that the network should remain active.

We overcome this insufficiency of MEDAL’s expressiveness by introducing the use of inhibitor arcs. An inhibitor arc is represented by an arc which terminates with a small circle. It always connects a place and a transition so that the arc is output-arc for the place and input-arc for the transition. The presence of an inhibitor arc connecting an input place to a transition changes the transition’s enabling conditions. A transition is regarded as enabled if each input place, connected to the transition by a normal arc (an arc terminated with an arrow), contains the necessary number of tokens, and no tokens are present on each input place connected to the transition by an inhibitor arc [95].

Figure 3.6 presents the MEDAL model for this application. The actuation subnet is shown in the shaded area. Place *I* is used as the “inhibitor” place and the token in it prevents transitions *T3*, *T4*, and *T5* from firing due to the inhibitor arcs connecting place *I* and transitions *T3*, *T4*, and *T5*. When the motion sensors detects activity, which produces tokens in place *M*, transition *T2* consumes the token in place *I*. At that point, transitions *T3*, *T4*, and *T5* are enabled to fire. The arc connecting transition *T2* back to place *I* creates a loop in the model. This loop guarantees that inhibition can be resumed once the motion sensors

stop firing. We discuss loops in more detail in Section 5.3.

The model in Figure 3.6 has an important advantage over the model in Figure 3.5. With the help of the time guard function β we can specify that transition $T2$ can fire 3 minutes after it has consumed its necessary input tokens. In this way we can satisfy the last requirement of the application, i.e. that the network stays awake 3 minutes after the last motion sensor report has been received.

3.3.3 Feedback control

Sensor networks are physical systems which run in constantly changing challenging environments. At the time of design of a WSN, the sensor network designer does not have a complete *a priori* knowledge of all environmental properties and future changes. A common way to address this issue is to make a set of assumptions about the environment where the network will be deployed. However, since we cannot fully predict all environment changes, the assumptions made during the design process often turn out to be incorrect. When this occurs, the most desirable response from the system would be to adapt itself to the changes that have taken place. The ability to adapt is especially valuable for sensor networks since they are often deployed in areas that are hard to access and sending maintenance is not always a feasible or a safe option.

Many WSN applications employ feedback control to adjust to the continually changing environment and thus improve the performance of the system [93, 94]. Modeling applications that can learn and adapt to the environment requires that MEDAL is able to express such learning behavior. The application logic modeled by MEDAL is a one-directional flow of tokens from sensors to events of interest. This logic flow is not sufficient to represent iterative processes such as learning. To address this limitation, we introduce the use of feedback loops. Feedback loops allow the language to express iterative behavior, such as learning and system-tuning.

Consider once again the explosion monitoring scenario. In all previous variations of this example, we have assumed that the nodes are sensing the environment at a constant rate. However, this might not be the most energy-efficient approach. What we could do instead is tie the sampling period to one of the parameters the application is monitoring. For example, we could establish a positive correlation between the temperature increase rate and the sampling rate. This approach would keep the sensing rate low and thus conserve energy when the temperature is low. It will also allow the system to promptly adjust the sensing rate if the temperature starts increasing significantly.

Introducing feedback loops to MEDAL allows us to specify the explosion scenario described above. Figure 3.7 shows the MEDAL model of the application. For clarity, the feedback control loop is displayed using

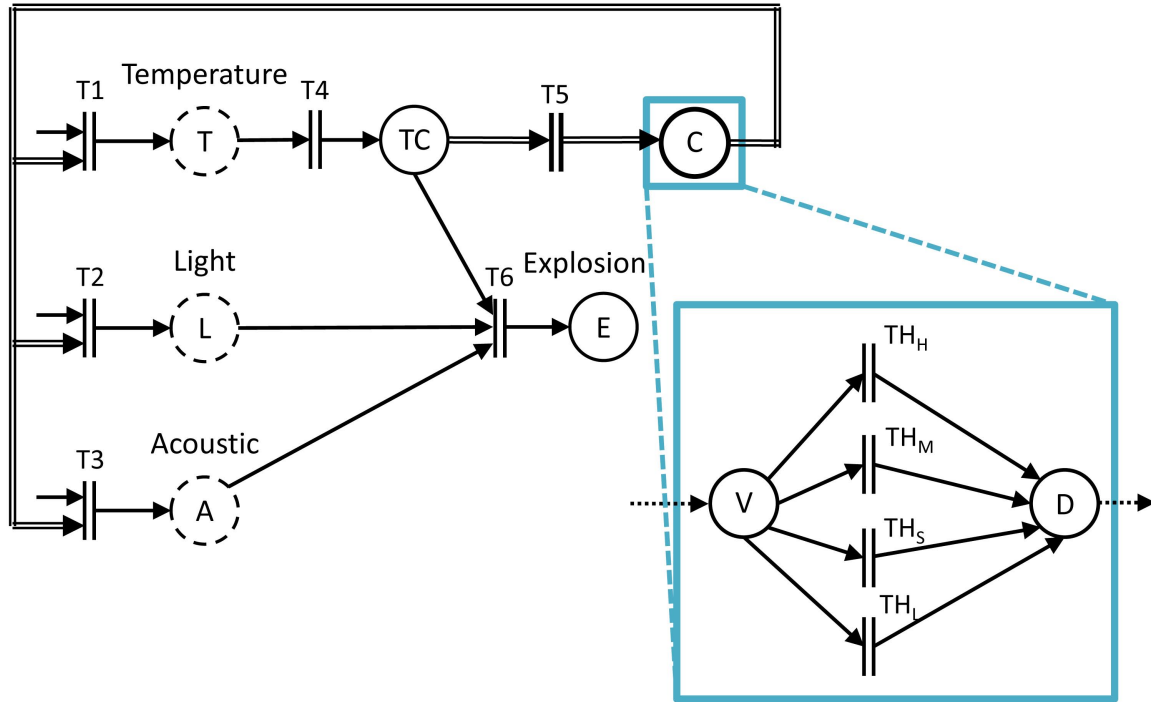


Figure 3.7: MEDAL model of an explosion application where the sensing frequency of the nodes is required to change based on the value of the temperature in the monitored area. Introducing a feedback loop to MEDAL (shown with double lines) allows us to model such a requirement.

double-line arcs. The temperature sensor is connected to the controller C through place TC (temperature control). This controller determines what the sampling frequencies should be based on the measured temperature values. When the new sampling rate is determined, it is looped back to the sensor nodes.

Place C is a very high-level representation of a controller. If a more detailed model of the controller's logic is required by the application, this model can be used to replace place C in the application model. Figure 3.7 shows an example of a more detailed controller logic. On the right we can see the expansion of place C into a set of places and transitions. Place V contains information about how quickly the values reported by the temperature sensor are changing. The controller follows the following algorithm:

- If the temperature is increasing rapidly, transition TH_H is fired. Transition TH_H generates a token which indicates that the sampling rate of the sensor nodes should be significantly increased.
- If the temperature is increasing at a medium rate, transition TH_M is fired. This transition generates a token which indicates that the sampling rate of the sensor nodes should be slightly increased.
- If the temperature is not changing or is slightly increasing or decreasing, transition TH_S is fired. This transition generates a token which indicates that the sampling rate of the sensor nodes should remain the same.

- If the temperature is decreasing, transition TH_L is fired. This transition generates a token which indicates that the sampling rate of the sensor nodes should be decreased.

In this way, the controller determines its decision based on the rate at which the temperature values are changing. More complex controllers can be modeled in a similar way.

3.4 Additional modeling concerns

3.4.1 Node mobility

The physical properties of the underlying network are often orthogonal to the application logic. An advantage of MEDAL is that it allows the application designers to abstract away the application logic from the dynamics of the network topology. For example, if the application will be executed on a network where some of the nodes are mobile, the MEDAL model of the application might not need to reflect the node mobility. Consider again the explosion detection application on Figure 3.3. The network topology might be such that the sink collecting all information from the sensors is mobile. This, however, does not change the high-level application logic - if the base station receives messages indicating high values of temperature, light, and sound, and those values satisfy the application's temporal and spatial constraints, the base station should conclude that an explosion must have occurred. The location of the explosion can be determined based on the location where the messages were generated rather than the current location of the sink.

MEDAL can also be used to model applications for which the node mobility is a vital part of the application logic. Consider, for example, an application where the designers need to specify the following logic: *When the mobile sink receives information from sensors indicating that there is an explosion, it should move in the direction of these sensors, so it can collect data about the explosion quicker.* Figure 3.8 shows the MEDAL model for this application scenario. In this model there are four types of nodes: temperature nodes, light nodes, acoustic nodes, and sink nodes. When the temperature, light, or acoustic nodes detect values above their predefined thresholds, they send a message to one of the mobile sinks. If a mobile sink receives messages from all three types of sensing nodes and these messages satisfy the spatial requirements of the application, transition $T7$ is fired. This firing indicates that 1) an explosion has occurred and 2) the sink should move towards the area of the explosion. The feedback loop that goes through place M (movement), which is a high-level representation of the logic performing the move, models the sink movement.

Figure 3.8 also shows a more detailed model of the movement logic. Transition T_{Dist} evaluates the distance between the sink and the detected explosion. If the distance is within the requirements of the application, i.e. the sink is sufficiently close to the explosion, transition T_C fires and the system reaches place

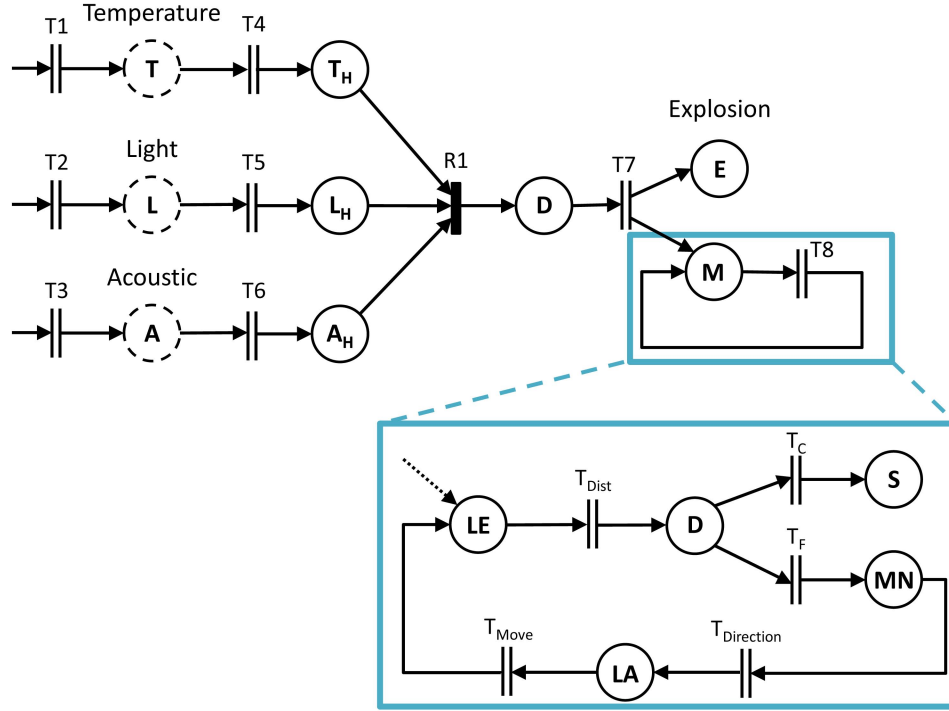


Figure 3.8: MEDAL can be used to explicitly model mobility application requirements. This explosion application model specifies that when an explosion is detected, the mobile sink should move towards the area of the explosion. The figure also shows a more detailed view of the movement logic.

S, which indicates that the sink should stop moving. If, however, the sink is too far from the area where the explosion was detected, transition T_F fires and the sink reaches place MN (movement necessary). Transition $T_{Direction}$ evaluates the direction in which the sink should move to get closer to the explosion, and transition T_{Move} sends the instruction for the move. The loop in the movement logic, which goes back to place LE, allows the system to reevaluate the new distance between the sink and the explosion location after some specified time during which the sink has been moving towards the desired area.

3.4.2 Prioritization

Currently most sensor networks run one application, where the application focuses on the detection of one particular type of event, such as a fire, a volcano eruption, or an elderly person falling in their home. However, we expect to see two trends in the near future: 1) the complexity of sensor network applications will increase and 2) multiple applications will be collocated on the same sensor network. These developments, although beneficial, will bring along the question of inter- and intra-application prioritization. Consider the following scenario where two applications are collocated on a sensor network deployed in an office building. The first application is for fire detection. The second application monitors the everyday patterns of the people in the building and uses this information to save heating energy. The network is resource constrained and

although the two applications can coexist in general, there might be situation where one application needs to monopolize the resources. For example, if the fire detection application signals the presence of fire, all network resources should be directed towards that application in an effort to find safe egress routes, rather than towards saving energy.

Petri nets with inhibitor arcs are shown to be equivalent to Petri nets with priorities [96]. Therefore, the inhibitor arcs, which we added to MEDAL to model actuation, can be used to specify prioritization. An inhibitor arc can be used to represent the prioritization condition between two transitions. For example, in Figure 3.6, transitions $T3$, $T4$, and $T5$ cannot fire if transition $T2$ is enabled. Therefore, transition $T2$ has a higher priority than transitions $T3$, $T4$, and $T5$, which belong to the same priority group. In addition to intra-application priorities, inhibitor arcs can be used in a similar way to model the priority relationships between applications.

3.5 MEDAL aided analysis for sensor network applications

3.5.1 Real-time analysis

An advantage of MEDAL is that it can also be used for real-time analysis of sensor network applications. For example, a safety requirement for the explosion detection application could be that explosions are detected within 5 seconds of their occurrence. If the application takes longer to detect an explosion and send an alert, there might not be enough time to evacuate the people in the dangerous area. The MEDAL model of the explosion application could be used to help determine if the application is able to meet this requirement. Figure 3.9 shows the application model from Section 5.1. During the design of the application, the sampling periods for the temperature, light, and acoustic sensors have been set to 3, 2, and 4 seconds, respectively. In addition, based on knowledge about the nodes that are going to be used and the environment, the designers have determined that, due to interference, transmitting a message over the radio could take up to 2 seconds in the worst case. Since the rest of the transitions in the model represent simple computations, the execution times associated with them could be neglected for this particular real-time application analysis.

Figure 3.9 shows the time information associated with each transition. The time notations, which are part of the MEDAL specification, allow us to perform the required real-time analysis of the system. Analyzing this model reveals that, in the worst case, an explosion might be detected 6 seconds after it has occurred - 4 seconds for the sound to be detected plus 2 more seconds for delivering the message over the radio. This, however, means that the application may not be able to meet its real-time requirements. Therefore, either the sampling period of the acoustic sensor has to be decreased to at most 3 seconds, or the network topology

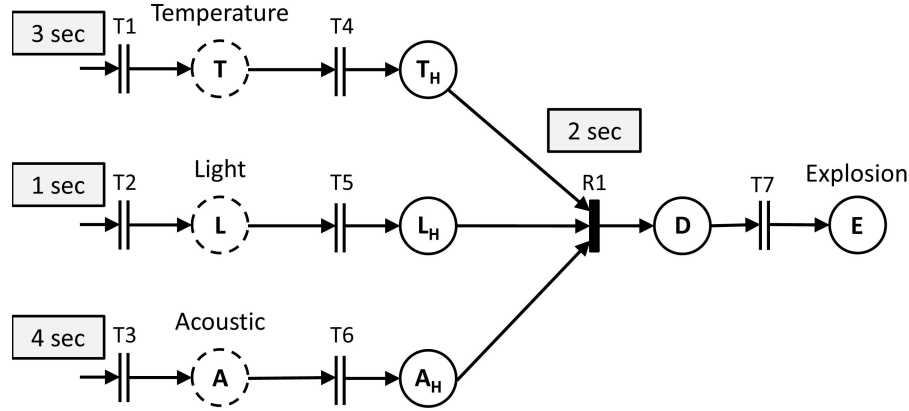


Figure 3.9: The MEDAL model of an application, together with time notations associated with the transitions in the model, can be used to perform real-time analysis.

should be altered so that successfully sending a message over the radio takes at most 1 second in the worst case. Although this is a simplistic example, similar analysis could be applied in much more complex systems.

3.5.2 Safety analysis

In scenarios where a sensor network is used to monitor for the occurrence of harmful events, MEDAL can help in both securing the people and decreasing the damage. Figure 3.10 shows the MEDAL model of a sensor network application used to monitor the emission of dangerous gasses in a factory. Gases A, B, and C could be by-products of the manufacturing process. Gases A and B alone are not harmful, but if they are mixed for more than an hour the resulting gas is dangerous. On the other hand, the presence of gas C alone for more than 40 minutes is also dangerous. Places A, B, and C are reached when gases A, B, and C are present respectively. Transition T_{10} is fired when both gas A and gas B are present. Similarly, transition T_{11} fires when gas C has been detected.

The marking of the model on Figure 3.10 indicates that the sensor network has detected the presence of only gas A. In this case, by analyzing the different paths of the model we can conclude that the safe time period to be in the factory is:

$$\min [(\beta(A_{T8,B}) + \beta(\text{communication}) + \beta(A_{T10,E})), \\ (\beta(A_{T9,C}) + \beta(\text{communication}) + \beta(A_{T11,E}))].$$

If gas B is consequently detected, the safe period to be in the building will be at most $\beta(A_{T10,E})$, which is an hour. If gas C is detected, the safe period to remain in the building will be $\beta(A_{T11,E})$, which is at most

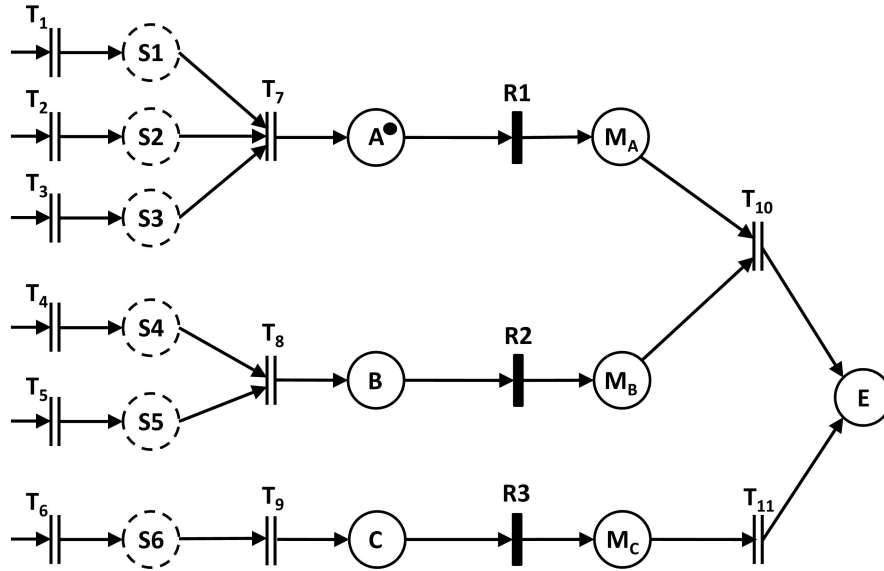


Figure 3.10: MEDAL model of an application used to detect dangerous gases. The network can detect three types gases, A, B, and C. Exposure to the mixture of gas A and gas B for longer than an hour is considered dangerous and so is the exposure to gas C for more than 40 min.

40 min. Therefore, if the system is in the state shown in Figure 3.10, the safe period to be in the factory is between 40 min, in the worst case, and infinity, if gas B and gas C are never produced.

Had the marking of the model been such that there was also a token in place B, the safe period to remain in the factory would have been:

$$\begin{aligned} & \min [(\beta(\text{communication}) + \beta(A_{T_{10},E})), (\beta(A_{T_9,C}) + \beta(\text{communication}) + \beta(A_{T_{11},E}))] \\ & = \min[(1 \text{ hour}), (\beta(A_{T_9,C}) + \beta(\text{communication}) + \beta(A_{T_{11},E}))] \end{aligned}$$

Since both gas A and gas B are present, people can remain in the building for at most an hour. If gas C is to be detected, the safe period to stay in the factory would be less than 40 min. Therefore, with a marking of the model indicating the detection of both gases A and B, the safe period to remain in the factory is between 40 min, in the worst case, and an hour otherwise.

Generally speaking, this kind of analysis relies on the analysis ability of Petri nets to compute the delays on different paths to a particular target event given a specific marking. Before the system is even deployed, various potential markings can be tested to see what the response of the application model will be under different configurations.

	SQL	Snlog	Macroprogramming	TLA+	Abstract regions	SNEDL	MEDAL
Expressiveness							
Probabilistic events						✓	✓
Hierarchical model	•			✓		✓	✓
Spatial semantics		•	✓		✓	✓	✓
Temporal semantics			✓	✓		✓	✓
Communication		✓	✓	•	✓		✓
Actuation		✓	✓	✓			✓
Feedback control			✓				✓
Analysis properties							
Real-time analysis			•			•	✓
Safety analysis				•		✓	✓
Failure analysis			•	✓		✓	✓
Graphical support							
						✓	✓

Table 3.1: Modeling properties of sensor network event description languages

Legend:•- a language partially has the property; ✓- a language has the property.

3.6 Evaluating MEDAL’s modeling properties

To evaluate MEDAL, we have compared its expressiveness to that of a number of other specification languages used in WSN applications. We evaluate the languages in three main categories: expressiveness, analysis properties, and graphical support. The results of the comparison are shown in Table 3.1.

3.6.1 SQL

SQL and SQL-like semantics have been used by the majority of work on event description in sensor networks. Not having the ability to specify essential sensor network properties, such as collaborative decision making, communication, and non-deterministic events, renders SQL not very suitable for describing sensor network events. In Table 3.1 we mark SQL’s ability to model hierarchical systems as *partial* since using complex queries does allow the specification of a hierarchical system. However, hierarchical data could be hard to both model and traverse when using SQL. Further, SQL does not support any of the analysis properties in Table 3.1.

3.6.2 Snlog

Snlog [97], a dialect of Datalog [98], is a declarative data-centric language for describing data management and communication in a WSN. Snlog has been used to specify a variety of WSN protocols, services, and applications, such as routing, data collection, link estimation, and localization. Snlog provides sensor network

designers with the opportunity to develop short sensor network programs using a high-level language. In addition to data collection and processing, the language provides mechanisms to specify communication, both node to node and broadcasting, and actuation. However, despite its advantages, Snlog exhibits drawbacks similar to those of SQL: it does not support the specification of probabilistic events, hierarchical models, temporal or spatial semantics, or feedback control. Also, it does not provide any graphical support or analysis capabilities.

3.6.3 Macroprogramming

Macroprogramming is another abstraction mechanism used for programming sensor networks. Macroprogramming allows network designers to program the global behavior of the application using a high-level specification, rather than writing low-level code for the individual nodes. This approach treats the sensor network as one whole unit, rather than producing specific software for each individual node.

There is a significant body of work done on macroprogramming WSNs, including ATaG [99], Kairos [100], MacroLab [101], and Regiment [102], and the developed macroprogramming languages and frameworks differ in many ways. However, macroprogramming approaches, in general, do not have the ability to model probabilistic events or represent hierarchical models. In addition, when compared to MEDAL, they have restricted analysis capabilities. A few solutions, such as MicroLab, can be used to perform real-time analysis. ATaG's network and application graphs could help with failure analysis. However, none of the existing macroprogramming solutions provides substantial application analysis capabilities. Further, the macroprogramming approaches lack graphical support.

3.6.4 TLA+

TLA+ is a specification language introduced by Leslie Lamport in 1998 [103, 104]. It is used for the specification of distributed, asynchronous systems and real-time concurrent processes. The language joins linear temporal logic and classical set theory providing specification and verification of software and hardware systems. TLA+ has been used for specification of WSN systems in a number of papers [105, 106]. As shown in Table 3.1, TLA+ has no support for expressing probabilities. In addition, it cannot be used to model feedback control and is cumbersome when modeling communication. Further, it does not provide convenient abstractions to perform safety analysis or real-time analysis and it has no graphical support.

3.6.5 Abstract regions

Abstract regions is a suite of general-purpose communication primitives introduced by Welsh et al. [107]. The primitives provide high-level interfaces for communication, data sharing, and performing aggregation on shared variables. The regions may be defined in terms of radio connectivity, geographic location, or other node properties. Abstract regions can support a wide range of sensor network applications, such as tracking, routing, and event detection using directed diffusion. They can also form the basis for other, higher-level communication models. Among the implemented abstract region primitives are:

- N-radio hop: nodes within N radio hops;
- N-radio hop with geographic filter: nodes within N radio hops and distance d ;
- k-nearest neighbor: k nearest nodes within N radio hops;
- k-best neighbor: k nodes within N radio hops with the highest link quality;
- approximate planar mesh: a mesh with a small number crossing edges;
- spanning tree: a spanning tree rooted at a single node, used for aggregating values over the entire network.

Although the abstract regions provide modeling support for spatial semantics and communication, they cannot be used to model probabilistic events, hierarchical structures, temporal semantics, actuation, or feedback control. Further, abstract regions do not provide any analysis capabilities or graphical support.

3.6.6 SNEDL

SNEDL [30] is the language closest to MEDAL in terms of syntax. However, SNEDL lacks the ability to model fundamental sensor network properties, including communication, actuation, and feedback control. Being an enhanced Petri net itself, SNEDL has almost identical analysis capabilities to MEDAL. However, it can perform only *partial* real-time analysis, since it cannot model communication, and communication greatly affects the time-related properties of WSN applications.

3.6.7 MEDAL

The extensive analysis capabilities of Petri nets are a great asset to MEDAL. There are multiple analyses that can be performed on the model of an application and Table 3.1 lists some of them. For example, the MEDAL model of an application can be used to aid *safety analysis* by answering questions such as: 1) Is it possible and under what conditions will event A occur? 2) How much time do we have to safely evacuate people out of a hazardous area? 3) What is the worst that can happen if events A , B , and C occur together? Similarly, the MEDAL model of an application can be used for *high-level failure analysis*. If the system fails

to detect an event, we can use the MEDAL model to generate tests to help determine what the possible causes for this failure could be. For example, if an explosion detection application fails to detect an explosion, analyzing the MEDAL model can help us identify some of the possible high-level failures, such as: 1) the temperature, light, or acoustic sensor nodes might have run out of power; 2) a message might have been lost; 3) the radio of the cluster-head node could be broken; 4) the thresholds might have been set too high. We can employ this information to generate the appropriate tests and use them to verify which of the failures could have occurred.

3.7 Experience with MEDAL: Beef monitoring

In order to illustrate the power of MEDAL, we present a specific sensor network application with a detailed MEDAL description. Suppose that a sensor network has been installed in a beef storing facility to monitor the environmental conditions. Since beef and meat in general are considered potentially hazardous food by the FDA, there are very strict regulations about the conditions in which they should be stored. The wireless network has to monitor the temperature, relative humidity, and air circulation rates. It is also preferable that beef is stored in the dark, for light accelerates the oxidation of fat with the liberation of free fatty acids and the production of rancidity [108]. According to regulations for storing beef the temperature must be maintained within the range -1°C ; to 5°C , the mean air speed over the product should be above 0.5 m/s, and the relative humidity must be maintained below 95% or, if the product is stored for longer than 72 hours, below 90% [108]. As these conditions (low temperature and darkness) are not comfortable for humans, it is desirable that the monitoring and control of beef be automated.

There are five types of nodes in the network:

1. light sensors to detect sunlight;
2. temperature sensors;
3. humidity sensors;
4. air circulation sensors;
5. a base station node.

The MEDAL model of the beef monitoring application is shown in Figure 3.11. There are four sensor places: S_L , S_T , S_H , and S_{AC} model the light, temperature, humidity, and air circulation sensors, respectively. There are also places that specify different local events and system state: L (light), HL (high light), T

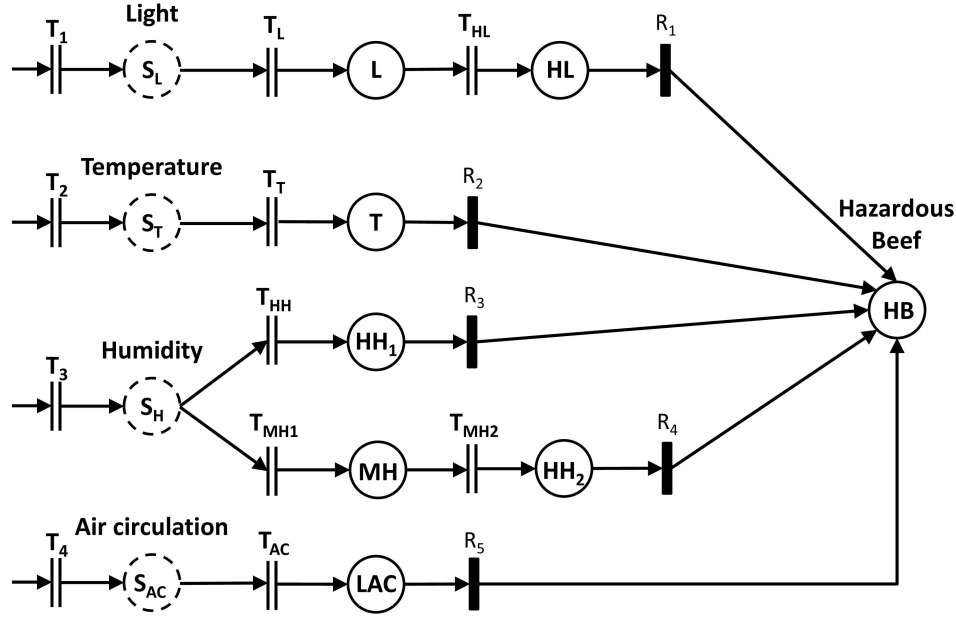


Figure 3.11: MEDAL diagram for a beef monitoring sensor network application. There are four types of sensor nodes in the network: light, temperature, humidity, and air circulation. When any of the food safety conditions are violated, the respective sensor sends a message over the radio to notify the base station.

(temperature), HH₁ and HH₂ (high humidity), HM (medium humidity), LAC (low air circulation), and HB (hazardous beef).

The MEDAL model specifies the application behavior if the environmental conditions do not meet the safety requirements:

- To model the condition that the meat should be exposed to light for only a limited period of time, transition T_L is fired when light is detected. If light is present for more than a specified length of time, transition T_{HL} (high light) is fired, which will cause the node to send a message over the radio, transition R_1 , to the base station. In this model, to make the application logic easier to follow, we have used separate radio transitions for all sensor nodes.
- The MEDAL model also specifies the condition that the temperature should be maintained between -1°C to 5°C . If the temperature is not within the specified range, transition T_T is fired and the temperature node sends a radio message to inform the base station.
- If the humidity rises above 90%, transition T_{MH1} is fired, and if the humidity remains above 90% for more than 72 hours, transition T_{MH2} fires and the humidity sensor sends a message to the base station. However, if the humidity rises above 95%, transition T_{HH} is fired, the system is in a state of high humidity (place HH₁), and the base station is informed.

- If the air circulation falls below 0.5 m/s, transition T_{AC} is fired. The system is then in a state of low air circulation, place LAC, so the air circulation node sends a message to the base station.

3.8 MEDAL models of existing applications

We empirically evaluate the expressiveness of MEDAL by specifying the models of sensor network applications that other researchers have built. In the rest of this section we present the MEDAL models for a volcano monitoring application [1], a rural fire detection application [2], and an early warning flood detection application [3].

3.8.1 Volcano monitoring application

A volcano monitoring sensor network was deployed on Volcan Reventador in 2005 [1]. The network consisted of 16 nodes, where each node was equipped with a seismic and an acoustic sensor. The application sensed data at a high rate, but the radio bandwidth of the network did not allow the application to forward all of the sensed data to the base station. Instead, the sensor nodes stored the sampled data in their local flash memory. Each node ran an event detector on the locally sampled data.

The event detector has to compute two exponentially weighted moving averages: a short-term average and a long-term average. When the ratio between the short-term average and the long-term average exceeds a predefined threshold, the detector is fired. The threshold allows nodes to distinguish between weaker signals, likely caused by distant quakes, and signals from nearby volcanic activity.

When the local event detector fires, i.e. when the node determines that it detects a nearby volcanic activity, the node sends a message to the base station. If the base station receives a sufficient number of reports within a certain time window, the base station initiates data collection from the whole network. Nodes can only store 20 minutes of eruption data locally. However, downloading 60 seconds of data from all 16 nodes in the network to the base station takes about one hour. Therefore, sampling and reporting events is paused while the nodes are uploading their locally stored data to the base station.

Figure 3.12 shows the MEDAL model for this volcano monitoring application. There are two types of nodes in the network: sensor nodes and a base station. Each sensor node is equipped with a seismic and an acoustic sensor, modeled by place S and place A, respectively. At each time step, the application calculates the two moving averages, short-term average (STA) and long-term average (LTA), for the seismic and the acoustic readings using the locally stored sensor data. Transition T_{CMP1} and transition T_{CMP2} are used to compare the difference in the averages from the seismic and the acoustic readings, respectively. If the

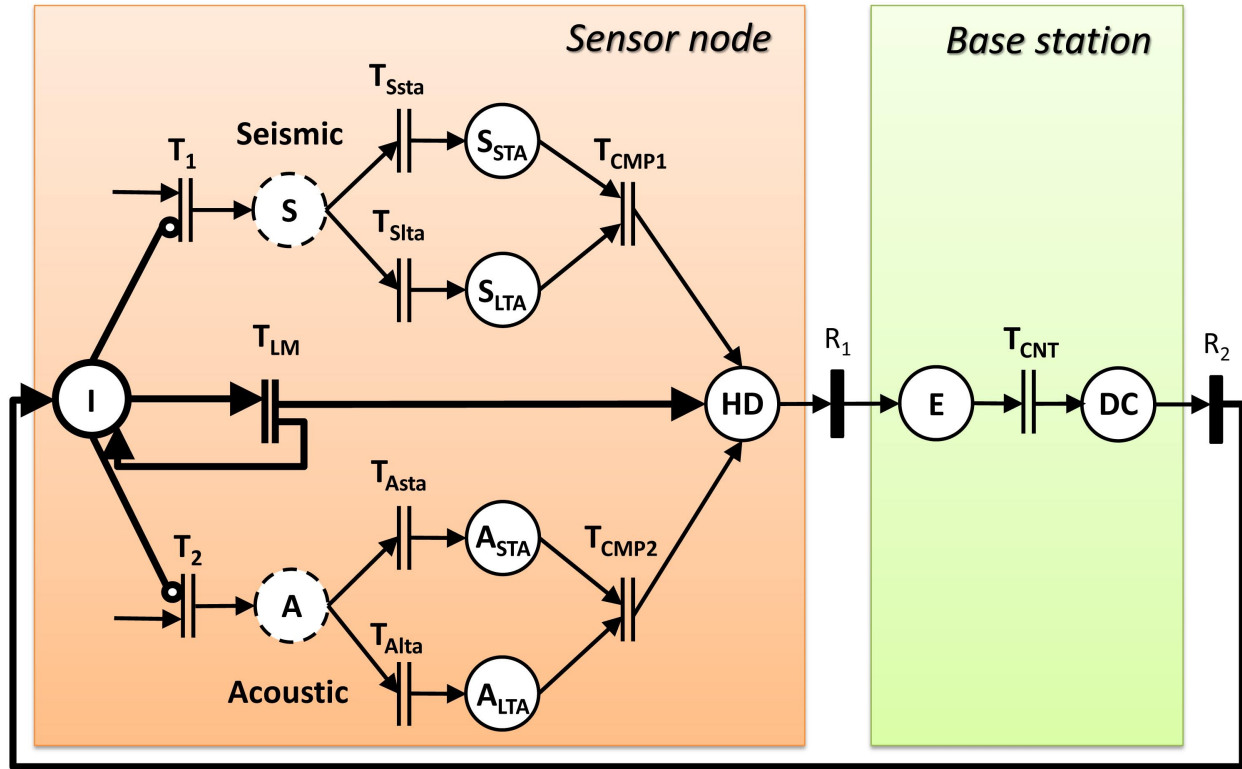


Figure 3.12: MEDAL model of a volcano monitoring applications. There are two types of nodes in the network: sensing nodes, containing seismic and acoustic sensors, and a base station node. For clarity, the application activities when a nearby volcanic activity has been determined are in bold.

difference is above the predefined detector threshold, the system reaches place HD (high difference) and the base station is notified.

Upon the reception of a sufficient number of radio messages, indicating a nearby volcano activity, the base station initiates data collection (place DC). To achieve this, the base station sends a radio message to the rest of the nodes in the system. Sending such a message, deposits a token in place I. This token achieves two things. First, since place I is connected through inhibitor arcs to transitions T_1 and T_2 , having a token in place I indicates that the sensing for the seismic and acoustic sensors is paused. Second, a token in place I enables transition T_{LM} , which is responsible for sending all readings stored in local memory to the base station over the radio. The cycle between transition T_{LM} and place I, guarantees that while there are readings that have not been sent yet, the sensing will be suspended and the transmission of information over the radio will continue.

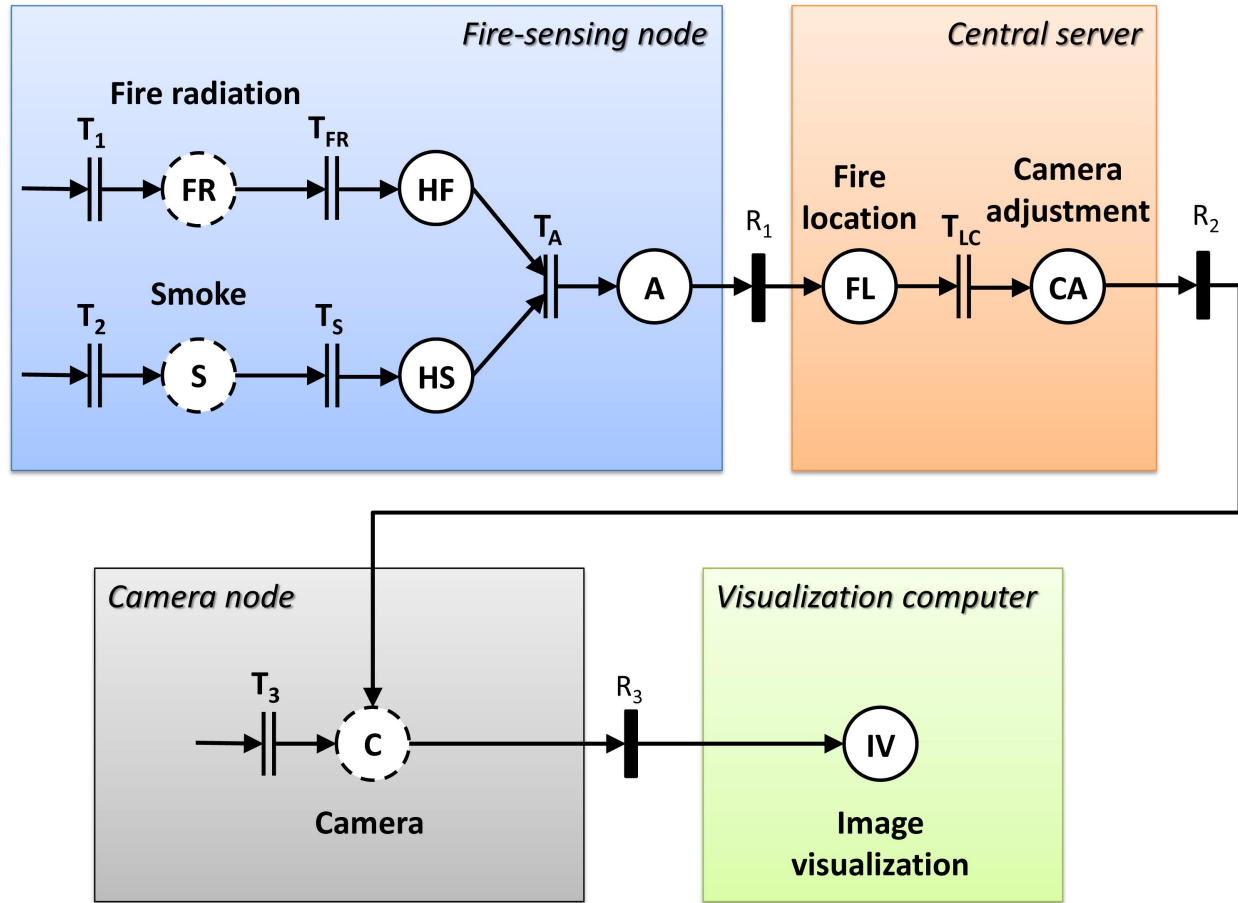


Figure 3.13: MEDAL model of a rural and forest fire detection applications. There are four types of nodes in the network: sensing nodes, containing fire infrared radiation and smoke, cameras, a central server, and a computer, which is used to visualize the data from the camera.

3.8.2 Rural fire detection application

The next WSN application we have modeled with MEDAL is a rural and forest fire detection application [2]. The network was deployed in Madrid, Spain and spanned a 2km diameter circle. The network contains sensor nodes, responsible for sensing the environment, wireless cameras, a central server, and computer used for visualizing the camera data. The sensor nodes are equipped with a sensor to monitor the fire infrared radiation and a smoke sensor. When a node detects fire radiation and smoke above the predefined application thresholds, it would send a fire alarm message to the central server.

The central server has a database that associates each sensor node with a particular location and with a set of the closest cameras deployed in that area. Upon identifying the cameras that are closest to the node which has sent the alarm, the central server sends a message to these cameras instructing them to move their objective towards that sensor. The images from all cameras located near the affected zone are sent to a

computer placed in the firefighter control room. In this way the system offers information about the location and behavior of the fire to the firefighter squad.

Figure 3.13 shows the MEDAL model of this fire detection application. Each of the four nodes in the network, sensor nodes, cameras, the central server, and the visualization computer, is shown in a separate box. Each fire-sensing node has a fire radiation sensor, place FR , and a smoke sensor, place S . If the node detects high levels of fire infrared radiation or high levels of smoke, transition T_{FR} or transition T_S is fired, respectively. When the application senses abnormal levels for both physical parameters, the node reaches an alarm state, place A . If a node detects the presence of fire, it has to send a message to the central server. The radio communication between the sensor nodes and the central server is modeled with radio transition R_1 .

When the central server is informed of a fire alarm, it determines the location of the fire using the coordinates of the node that has sent the alarm. The central server then locates the cameras closest to that location, transition T_{LC} , and prepares frames to be sent to these cameras. A frame contains instructions on how each camera should readjust itself so it can provide the best view of the affected area. The central server sends those frames to the cameras over the radio (radio transition R_2). Once the cameras have been adjusted, they send their information to a computer station, which is used to visualize the camera data.

3.8.3 Early warning flood detection application

Basha et al. have developed an application for river flood prediction [3]. The system contains three types of nodes:

- *Sensing nodes* The sensing nodes measure the variables needed to detect and predict floods. There are three different sensor nodes used in the network: temperature, pressure, and rainfall. The nodes analyze the data and compute statistics over each hour. They also send the sensed information over the radio to the closest computation node.
- *Computation nodes* Computation nodes have more processing capacity than the sensing nodes. When it receives readings from the sensor nodes in its area, a computation node runs the data through the flood detection model, computes the uncertainty of the model prediction, and requests additional data from the sensing nodes if it needs to reduce the uncertainty. If a computation node detects a flood, it sends a message to an interface node.
- *Interface nodes* The interface nodes provide user interface to the network. They are used to access data and predictions regarding the event of interest. The interface nodes communicate with the computation nodes to provide any external requests for data and receive all of the existing network data.

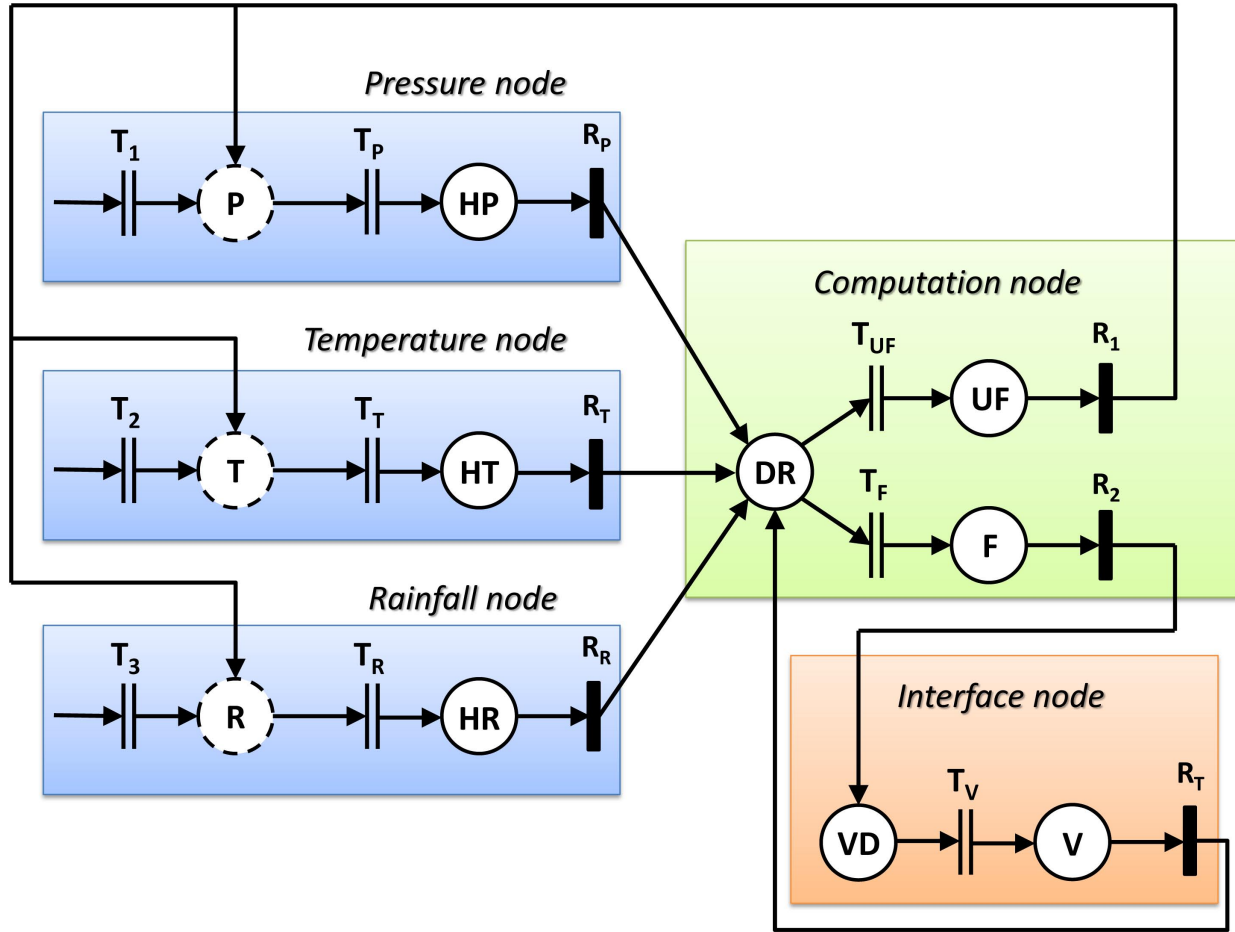


Figure 3.14: MEDAL model of a flood detection application. There are three types of nodes in the network: sensing nodes, which include pressure, temperature, and rainfall sensor nodes; computation nodes, which evaluate the probability of a flood; and interface nodes, which can be used to visualize the data and send requests to the computation nodes.

Figure 3.14 shows the MEDAL model of the flood detection application. The pressure, temperature, and rainfall nodes send their readings to a computation node when these readings are higher than the predefined respective application thresholds. When a computation node receives a reading from a sensing node, the system is in place DR . If the computation node is not sufficiently certain in the presence of a flood, transition T_{UF} fires and the node requests additional data from the sensor nodes. If, however, the computation node has sufficient information to determine if there is a fire in the system, transition T_F fires and the system reaches place F . If a flood is detected, the communication node notifies an interface node. The interface node is responsible for processing the data and visualizing it (place V). An interface node could also send a message to the computation nodes if it needs any additional data.

3.9 Summary

In this chapter we have described MEDAL, a formal event description language which is an enhanced Petri net. This language can capture the structural, spatial, and temporal characteristics of a complex sensor network event detection application. We have also introduced a number of syntax extensions to MEDAL, which allow the language to model additional fundamental sensor network properties, including communication, actuation, and feedback control. The MEDAL model of an application can also be used to analyze real-time and safety properties of the application.

As a proof of concept, we describe our experience using MEDAL to specify a beef monitoring application. Further, we have used MEDAL to model a number of existing sensor network applications, including a volcano monitoring application, a rural fire detection application, and an early warning flood detection application. Our experience shows that MEDAL can model a wide variety of event-driven applications.

Chapter 4

Applying Formal Methods to Modeling and Analysis of Real-Time Data Streams

SITUATION awareness (SA) has been recognized as a critical foundation for successful decision-making across a broad range of complex real-time applications, including aviation and air traffic control [109], emergency response [110], and military command and control operations [111]. SA is especially important for applications where the information flow can be high and poor decisions may lead to serious consequences (e.g., piloting a plane, or treating critically injured patients). These applications need to operate on continuous unbounded streams of data to understand how incoming information and events could impact the system, both now and in the near future. The streaming data may come from various sources, such as sensor readings, router traffic traces, or telephone records. Therefore, the capability to manage data streams becomes an essential application requirement.

These applications also have inherent real-time requirements, and queries on the streaming data should be finished within their respective deadlines. Consider a surveillance system as an example. The system is expected to detect if a target enters the monitored area and alert the controlling party (e.g., human operators). If the detection does not occur within a certain deadline, the target may be missed. However, due to the dynamic nature of the input data streams, the stream queries may have unpredictable execution cost. First, the arrival rate of the data streams can be volatile, which leads to variable input volumes to the queries. Second, the content of the data streams may vary with time, which causes the *selectivity* of the

query operators to change over time. The variable stream behavior and the irregular workload pattern make maintaining the desired level of QoS a challenging task.

Formal data stream analysis can help tremendously in achieving the desired levels of QoS for real-time data stream applications. Such analysis can provide query designers with better understanding of the behavior of the real-time data streams they are working with. It could also be used to more accurately predict changes in the data arrival patterns and the query workloads. In addition, analysis of the data admission controller, responsible for determining the amount of input stream data to the queries, could aid the design of better controllers that can adapt faster to workload fluctuations. To perform such analysis, we need a specification language that will allow us to formally model real-time data streams, stream queries, as well as data admission control mechanisms.

In this chapter we describe how MEDAL is used to model and analyze real-time data stream applications. First, each query plan can be specified with a MEDAL model. Second, MEDAL is also capable of modeling the data admission controller. MEDAL's ability to model both the queries and the data admission control enables us to combine the two main components of data stream applications, query logic and control, into a single comprehensive system model. Third, using the MEDAL query models to analyze different system properties, such as the behavior of the data streams, the cost and selectivity of different query operators, and the real-time properties of the queries, can significantly improve the QoS of the applications. Further, MEDAL-aided analysis could also help admission control designers build better and more accurate prediction-based controllers.

The main contributions of the work presented in this chapter are:

1. Applying MEDAL to model real-time data stream queries and stream management mechanisms.
2. Integrating query logic and data control models into a comprehensive data stream application model.
3. Introducing how MEDAL can be used for analysis of real-time data streams, such as query plan optimization.
4. Demonstrating MEDAL's ability to model a diverse set of data stream management configurations.

4.1 System model

A data stream is defined as a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamps) sequence of data items [112]. A data stream management system (DSMS) is a system especially constructed to process persistent queries on dynamic data streams. DSMSs are different from traditional

database management systems (DBMS) in that DBMSs expect the data to be persistent in the system and the queries to be dynamic, whereas DSMSs expect dynamic unbounded data streams and persistent queries. Emerging applications, such as sensor networks, emergency response systems, and intelligent traffic management, have brought research related to data streams in focus. These applications inherently generate data streams and DSMSs are well suited for managing the produced data.

4.1.1 Periodic query model

So far, DSMS research has mainly been focused on using a continuous query model [113, 33, 114, 115]. In a continuous query model, long-running continuous queries are present in the system and new data tuples trigger the query instances. These incoming data tuples are then processed and the corresponding query results are updated. The continuous model performs well when the system workload is stable and the system has sufficient resources to finish all triggered query instances. However, since the number of query instances and the system workload depend directly on the input, which could be extremely volatile, this continuous query model is not appropriate for real-time applications that need predictable responses. Another drawback of this model is that, since the query execution is driven by the data rate of the system, the application does not have control over the frequency and the deadlines of the queries. For applications where some queries have higher priority and are more important than others, it might be desirable to have the ability to execute the important queries more often. This, however, is hard to accomplish in a continuous query model.

To address this, we employ a periodic query model (PQuery) for data stream queries with timing constraints [31]. In this periodic query model, every query has an associated period. Upon initialization, a query instance takes a snapshot of the data streams at its inputs. The query input does not change throughout the course of the query instance execution even when there are new data tuples arriving over the data streams. Instead, the newly arrived data tuples are processed by the next query instance. In this way, the query execution is not interrupted or aborted by new incoming data. When an application receives the results of a periodic query instance, it is with the understanding that these results reflect the state of the system when the query instance was initiated. In the periodic query model, the application can specify the query periods and deadlines. These parameters can be used to calculate the number of queries in the system at any given time, which allows us to estimate the query workloads much easier than with the continuous query model.

4.1.2 Query plan and query execution

A DSMS contains long-running and persistent queries. When a query arrives in the system, it is registered and is triggered periodically based on its specified period. All queries are converted to query plans (containing operators, queues, and synopses) statically before execution. Queues in a query plan store the incoming data streams and the intermediate results between the operators. A synopsis is associated with a specific operator in a query plan, and stores the accessory data structures needed for the evaluation of the operator. For example, a JOIN operator may have a synopsis that contains a HASH JOIN index for each of its inputs. When the JOIN operator is executed, these hash indices are probed to generate the JOIN results.

Consider the following example scenario where a traffic monitoring system has been deployed to constantly analyze the traffic in a particular city and determine the most suitable times for delivering supplies to the grocery stores. To achieve the desired situation awareness, the system analyzes data streams from speed and traffic sensors. We perceive events as the fundamental building blocks of a situation. Therefore, achieving situation awareness requires that we identify a particular set of events and perform our situation assessment and analysis based on their occurrence. One type of event that our traffic monitoring application is interested in is trucks that travel during light traffic in specific lanes, such as non-HOV lanes. For its consequent traffic analysis, the application calculates the average speed of these trucks. The SQL data stream and query specifications are given as follows:

Stream : *Speed* (*int lane*, *float value*, *char*[8] *type*);

Stream : *Traffic* (*int lane*, *char*[10] *type*);

Relation : *Lanes* (*int ID*, *char*[10] *type*, *char*[20] *road*);

Query : *SELECT avg (Speed.value)*

FROM Speed [range 10 minutes], Lanes,

Traffic [range 10 minutes]

WHERE Speed.lane = Lanes.ID AND

Lanes.ID = Traffic.lane AND

Speed.type = Truck AND

Traffic.type = Light

Period 10 seconds

Deadline 5 seconds

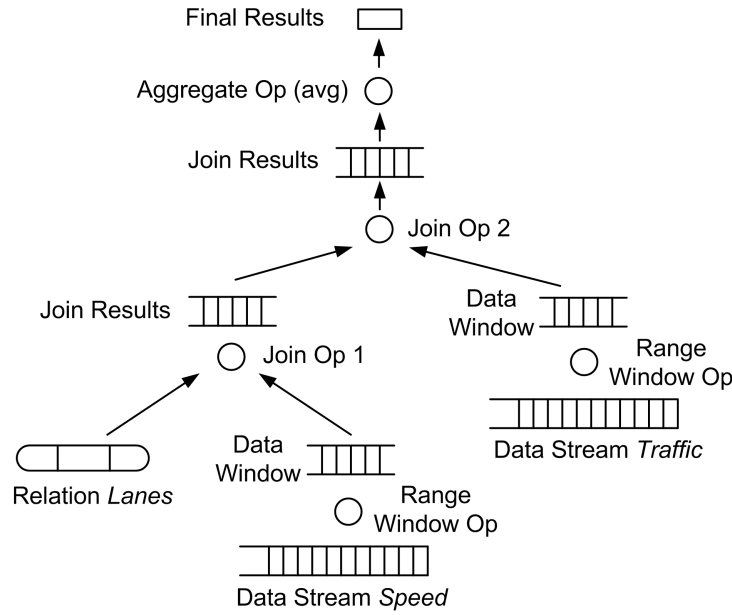


Figure 4.1: An example query plan.

The query above operates on data streams generated by speed and traffic sensors and calculates the average speed of trucks in particular lanes during light traffic over a 10-minute window. The query needs to be executed every 10 seconds and the deadline is 5 seconds after the release time of every periodic query instance. The query plan generated based on this query is shown in Figure 4.1. It contains three types of query operators (RANGE WINDOW operator, JOIN operator, and AGGREGATE operator) and two types of queues - one for storing the output of the RANGE WINDOW operators and one for storing the output of the JOIN operators.

After the query plan is generated, the operators are sent to the scheduler to be executed. Depending on the query model (e.g., continuous or periodic), a suitable scheduling algorithm, such as round-robin or earliest deadline first, could be chosen so that the system requirements are met.

4.1.3 Data admission controller

In many real-time applications, partial results are more desirable than queries missing their deadlines. Therefore, the system might trade off data completeness for better query miss ratios at run time. We employ an overload protection mechanism called *data admission controller*, which trades data completeness for better query miss ratios. The basic approach is to reduce the incoming data volume when the system becomes overloaded. The load shedding process is performed before the data stream tuples are processed by the queries. Operators perform data admission using *random* dropping. Though not covered in this chapter, *semantic*

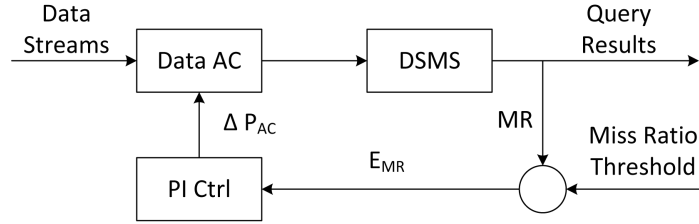


Figure 4.2: Data admission controller.

dropping can be used to improve system performance if query semantics are considered. The system also allows the data sources to mark the important data tuples to make sure they get processed. The importance flag is marked by setting the highest bit of the data tuple timestamp. The data tuples with flags are admitted to the system regardless of the current data admission ratios. However, operators maintain the target data admission percentage by dropping more unmarked data tuples.

The data admission process is controlled with a proportional-integral (PI) controller as it is simple to use and provides acceptable response time to workload fluctuations. A proportional-integral-derivative (PID) controller is not suitable in this situation because of the dramatic changes that might occur in the workloads of query systems from one sampling period to another. Adding a derivative control signal amplifies the random fluctuations in the system workloads [116].

The data admission control architecture is shown in Figure 4.2. The query miss ratio (MR) is sampled periodically and compared against the miss ratio threshold specified by the application. The result is used by the PI controller to calculate the data admission control signal ΔP_{AC} . The control signal is derived with the equation:

$$\Delta P_{AC} = P_{MR} \times (MR_{ST} - MR_t) + I_{MR} \times (MR_{LT} - MR_t),$$

where MR_{ST} and MR_{LT} are the short-term and long-term query miss ratios sampled in the last sampling period. MR_t is the maximum miss ratio allowed by the application. P_{MR} and I_{MR} are controller parameters that specify the significance of the short-term and the long-term query miss ratios when calculating the data admission control signal. The controller parameters determine the behavior of the controller. The process of tuning these parameters, i.e. *controller tuning*, is not the focus of our work. Readers are referred to [117] and [118] for details on the analysis and tuning of controllers.

In order to provide service differentiation, the system uses multiple data admission controllers. Each service class is associated with a designated data admission controller whose parameters have been specifically tuned for that particular service class. The use of multiple controllers is further discussed later when we model a number of different DSMS configurations.

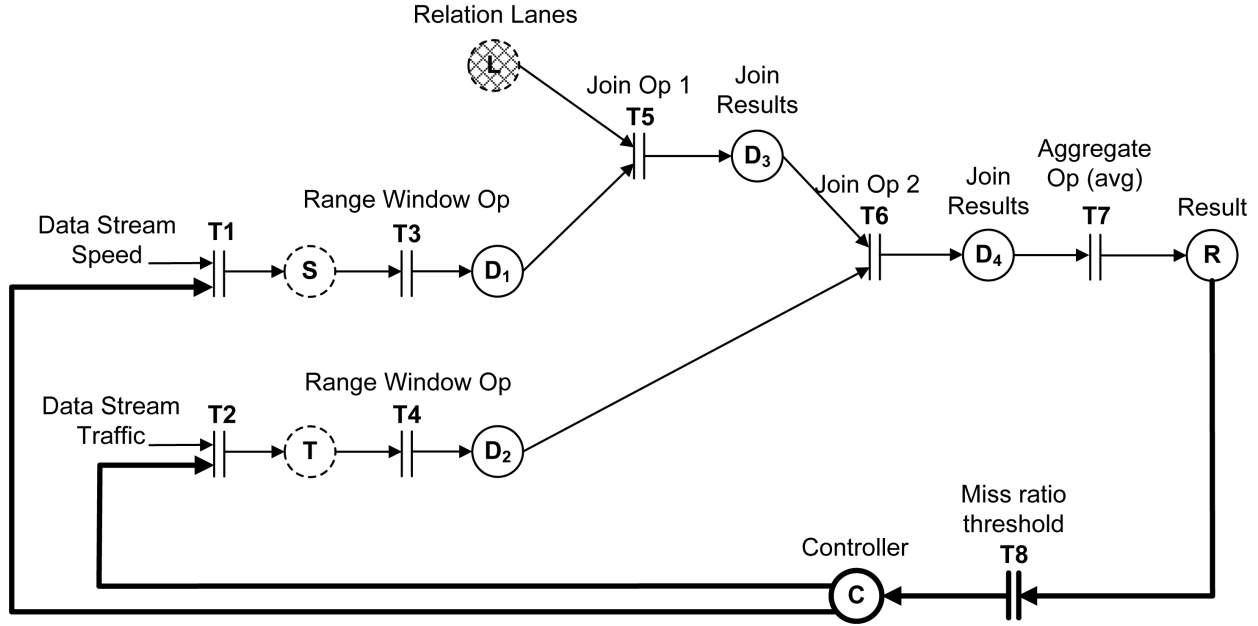


Figure 4.3: Using MEDAL to model the query plan from Figure 4.1. The query operators are modeled using transitions, while application state and data are represented with the help of places.

4.2 Modeling queries and control

An advantage of MEDAL is that it can be used to specify a comprehensive application model by combining the admission controller and the query plan models into a single MEDAL model. This is very beneficial for two main reasons. First, it allows us to model the direct relationship between the data admission controller and the query inputs. Second, integrating the query plan models and the data admission control logic gives us a better understanding of the correlation between the query results and the stream input volume.

4.2.1 MEDAL query plans

MEDAL can seamlessly be applied to modeling real-time query plans. The same query plan from Figure 4.1 is specified in Figure 4.3 using MEDAL. The different types of query operators are modeled with the help of transitions (transitions $T1 - T7$). Application state and data are represented using places. Places with interrupted border are used to model data input (places S , T , and L). We introduce the use of shaded places (place L) in order to distinguish between input relations, which tend to be static for the most part, and input data streams. Therefore, the set of places P in MEDAL can now be expressed as $P = S \cup R \cup E$, where S represents the input streams, R represents the input relations, and E represents the higher level events and intermediate results.

4.2.2 Data admission control

There is an array of real-time data stream applications that use feedback control to adjust to the constantly changing environment and improve system performance [94, 93]. To model feedback control, MEDAL employs a control theoretic approach which adopts the controller synthesis paradigm from control theory. Given a model of the system's dynamics and a specification of the desired closed-loop behavior, the objective is to synthesize a controller to achieve the specified behavior. This approach relies on the clear distinction between the system and the controller and requires that the information flow between them is explicitly modeled. We meet this requirement by designing two separate MEDAL models - one for the query plan and one for the data admission controller. The two models are then composed into a single application MEDAL model which allows us to analyze the interactions between the system and the data admission controller.

The feedback control loop in Figure 4.3 is shown in bold. Transition $T8$ compares the query deadline miss ratio to a predefined miss ratio threshold. If the observed miss ratio is higher, controller C calculates the new data admission ratio and propagates it to the system. Place C is a very high-level representation of a controller. A more detailed model of the controller's logic can be designed with MEDAL and used to replace place C , similarly to the MEDAL model in Figure 3.7.

MEDAL can be used to model different levels of abstraction. For example, analogously to using place C to model the entire controller logic, we can abstract away the query plan from Figure 4.3. The query plan could be modeled with a single place, similarly to how Figure 4.2 models a DSMS. This is especially helpful during the controller design phase, since it allows us to treat the query as a simple linear system without considering how the internal parts of this system interact. Using this simplified model of the query plan, the appropriate values for the P_{MR} and I_{MR} controller parameters are determined through system identification [119]. Once the controller is designed, we can expand the query plan model and use it to additionally adjust the controller to better reflect the query properties. The same abstraction approach can also be applied when there are multiple registered queries. The whole query set is abstracted away using a single place, which can consequently be expanded once the controller is designed.

Another requirement introduced by the control theoretic approach is that changes in the system are observable [120]. MEDAL achieves *observability* by explicitly identifying and modeling the parameters that need to be monitored. In Figure 4.3, the query results are delivered to controller C through transition $T8$. As mentioned above, this transition determines the difference between the query deadline miss ratio and the threshold specified by the application. Therefore, transition $T8$ allows us to explicitly model the query miss ratio and thus achieve the required *observability*.

The data admission control needs to be implemented at the query level so that different queries can have

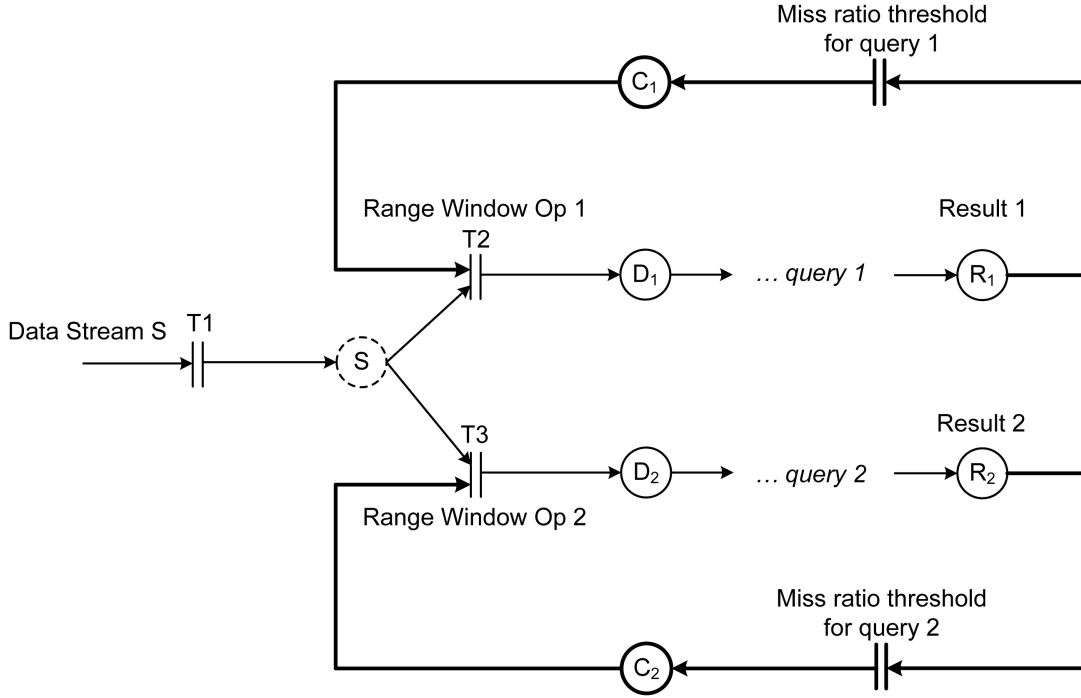


Figure 4.4: MEDAL model of the data admission control implementation. This design enables different queries to have different data admission ratios even when they share the same data stream source.

different data admission ratios even if they share the same data stream source. The two queries in Figure 4.4 share the same data stream input S . The incoming data tuples of data stream S are first processed by the *stream source* operator, represented by transition $T1$. After the stream source operator, the data stream tuples are in the system. The data admission process for the queries is carried out at the *range window* operators. In the example shown in Figure 4.4, *range window* operators Op 1 and Op 2, represented by transitions $T2$ and $T3$, perform the data admission for query 1 and query 2 respectively. With this design, different queries can use different data admission ratios.

4.3 MEDAL analysis

4.3.1 Query optimization analysis

The MEDAL model of a data stream system could be used to analyze the query plans, the controller logic, as well as the interactions between them. This analysis could help designers identify possible query plan optimizations and design more suitable data admission mechanisms. Consider, for example, the following scenario: For the query model in Figure 4.3 we have estimated the cost of each of the query operators. In addition, based on historical workload information, we have also estimated the average selectivity of the

operators. The data admission controller has been designed to decrease the data admission rate by $x\%$, where x is a function of the difference between the miss ratio threshold and the current deadline miss ratio. We can employ the MEDAL model of this application to determine if the controller logic has the desired effect on the deadline miss ratio. Using the available historical data as input to the MEDAL model, we can estimate the new deadline miss ratio, and thus evaluate the controller's efficiency. This is a very simple high-level example, but it showcases one of the possible MEDAL system analyses.

4.3.2 Real-time stream analysis

MEDAL can also be used for real-time analysis of data stream network applications. The data stream real-time analysis is similar to the real-time analysis of WSN applications introduced in the previous chapter. A MEDAL query plan can be annotated with the duration time of each of the operators in the query. If there is historical workload information that represents the typical behavior of the data streams, this data can be used to accurately estimate how much time each of the query operators needs to process the input data. This information can be used by system designers to evaluate the time requirements of queries and thus determine if the queries are able to meet their deadlines given.

4.4 Modeling DSMS configurations with MEDAL

In this section we show how MEDAL is used to model different data stream management configurations. We create MEDAL models for the following scenarios:

1. All queries belong to the same query service class.
2. There are three query service classes in the system and no data admission control is applied.
3. There are three query service classes in the system and they share a single data admission controller.
4. There are three query service classes in the system and each service class has its own designated data admission controller.

4.4.1 Single query service class

Figure 4.5 shows the MEDAL model of a system where all queries belong to the same service class. In this scenario there is one data admission controller, place C , which controls the volume of data that enters the system. If the control loop is removed from the MEDAL model, the resulting model is that of the default data stream management system configuration with one service class and no admission control.

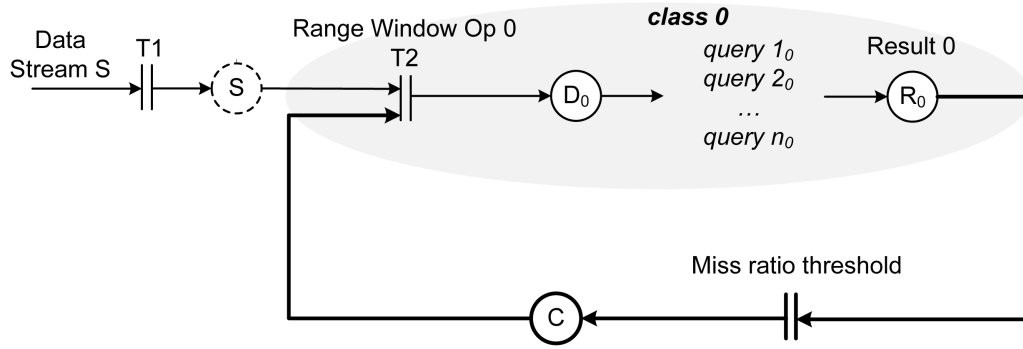


Figure 4.5: MEDAL model of a DSMS configuration, in which there is a single controller and all queries belong to the same query service class.

4.4.2 Multiple query service classes, no controller

Differentiated services are required by many applications. In case of overload, the system has to guarantee that the most important set of queries get processed. Figure 4.6 shows a scenario where the queries in the system are divided into three service classes, *class 0*, *class 1*, and *class 2*, where the *class 0* queries are the most important. The three query service classes share data stream *S*. Since there is no data admission control, there is no control loop that uses the query miss ratios to estimate the suitable data admission ratios for each query service class.

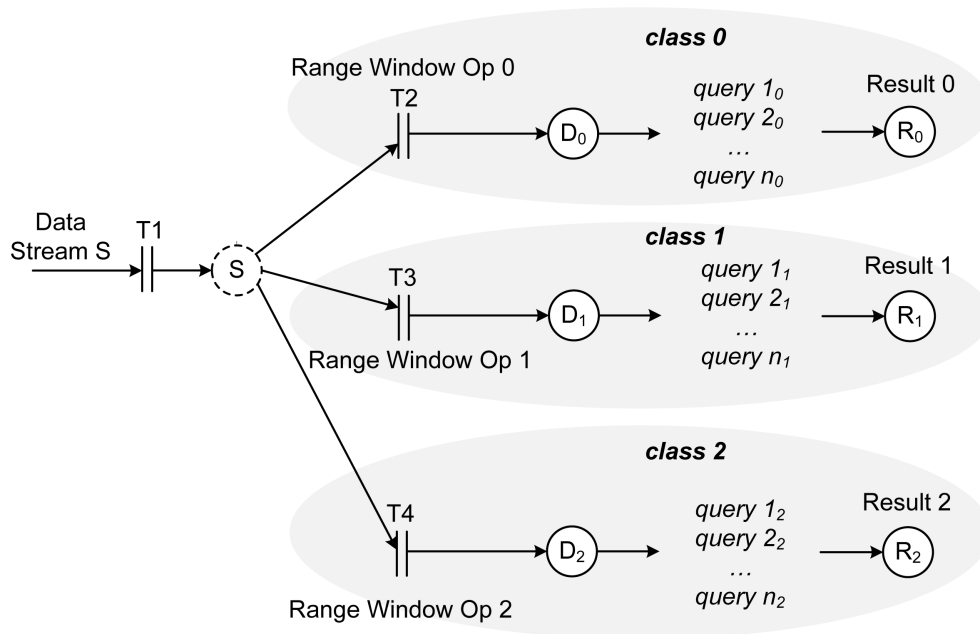


Figure 4.6: MEDAL model of a system with three different service classes, *class 0*, *class 1*, and *class 2*, and no admission controller

4.4.3 Multiple query service classes, single controller

Figure 4.7 shows the MEDAL model of a scheme where there is only one data admission controller and all queries in the system share the same data admission controller. Each service class has its own set of queries. As the queries are executed, the controller receives information about the missed deadline ratios for the three classes and determines whether more or less data tuples should be admitted into the system. The decision of the controller is then propagated to the three service classes.

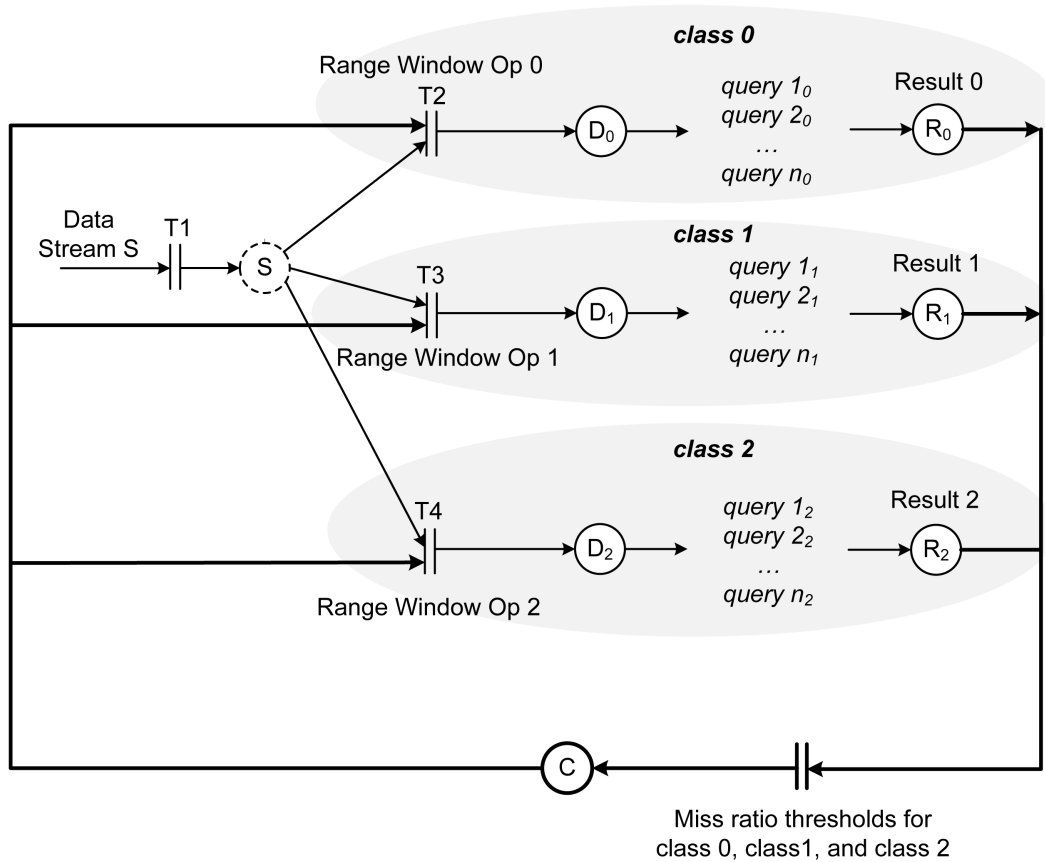


Figure 4.7: MEDAL model of a scenario with three query service classes, *class 0*, *class 1*, and *class 2*, which share a single data admission controller.

4.4.4 Multiple query service classes, multiple controllers

Figure 4.8 shows a different data admission scheme, where each of the service classes in this scheme has its own designated data admission controller. Although all three of the service classes receive data from the same data stream S , the data admission for each service class is determined independently. Each of the three classes, *class 0*, *class 1*, and *class 2*, has its own data admission controller, C_0 , C_1 , and C_2 , respectively. The controllers receive information only about the deadline miss ratio for the queries from their corresponding class.

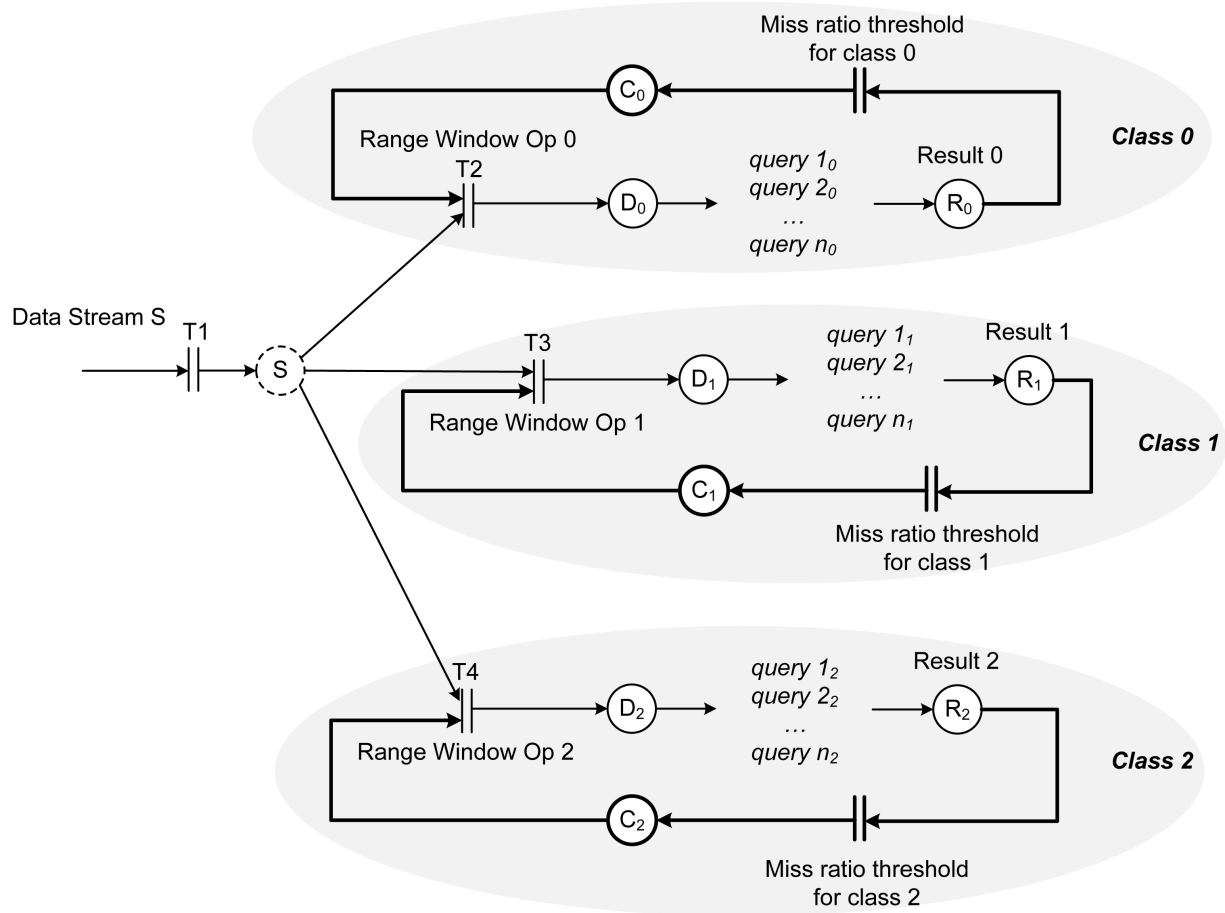


Figure 4.8: MEDAL model of a scenario with three service classes, *class 0*, *class 1*, and *class 2*, where each service class has its own designated controller.

4.5 Summary

In this chapter we show how MEDAL can be used to model and analyze real-time data stream queries, QoS management mechanisms, and the relationships between them. Unlike previous work, where query models and system control logic are designed and analyzed separately, MEDAL allows us to merge these two

components into a single comprehensive system model. The advantage of this combined model is that it can be used not only to predict the workload and estimate the query cost, but also to model and analyze the interactions between the input and output of the query plans and the data control mechanism, which gives us a much better understanding of the system.

We have also designed the MEDAL models for a number of different data stream management configurations, including scenarios where 1) all queries belong to the same query service class, 2) there are three query service classes in the system and no data admission control is applied, 3) there are three query service classes in the system and they share a single data admission controller, and 4) there are three query service classes in the system and each service class has its own designated data admission controller. These models show that MEDAL can specify a variety of stream, QoS management, and data admission configurations.

Part III

Event detection

Chapter 5

Event Detection in Wireless Sensor Networks - Can Fuzzy Values Be Accurate?

EVENT detection is one of the main components in numerous WSN applications. WSNs for military application are deployed to detect the invasion of enemy forces, health monitoring sensor networks are deployed to detect abnormal patient behavior, fire detection sensor networks are deployed to set an alarm if a fire starts somewhere in the monitored area. Regardless of the specific application, the network should be able to detect if particular events of interest, such as fire, have occurred or are about to. However, similar to many other human-recognizable events, the phenomenon *fire* has no real meaning to a sensor node. Therefore, we need suitable techniques that would allow us to describe events in ways that sensor nodes would be able to “understand”. The area of event description and detection in WSNs, however, has not been explored much.

Most previous work on event description in WSNs uses precise, also called *crisp*, values to specify the parameters that characterize an event. For example, we might want to know if the temperature drops below 5 °C or the humidity goes above 46%. However, sensor readings are not always precise. In addition, different sensors, even if located close to each other, often vary in the values they register. Consider an example scenario where we want the air conditioning in a room to be turned on if the temperature goes above 5 °C. Two sensors, *A* and *B*, measure the temperature in the room. The average of their values is used to determine if an action should be taken. At some point, sensor *A* reports 5.1 °C and sensor *B* reports 4.8 °C. The average,

4.95 °C, is below our predefined threshold and the cooling remains off. However, if sensor *B*'s measurement is inaccurate and, therefore, lower than the actual temperature, we have made the wrong decision. The situation becomes even more complex when more than two sensor measurements are involved. This makes determining the precise event thresholds an extremely hard task which has led us to believe that using crisp values to describe WSN events is not the most suitable approach. Fuzzy logic, on the other hand, might be able to address these challenges better than crisp logic.

Fuzzy logic has a number of properties that make it suitable for describing WSN events:

- It can tolerate unreliable and imprecise sensor readings;
- It is much closer to our way of thinking than crisp logic. For example, we think of fire as an event described by high temperature and smoke rather than an event characterized by temperature above 55 °C and smoke obscuration level above 15%;
- Compared to other classification algorithms based on probability theory, fuzzy logic is much more intuitive and easier to use.

A disadvantage of using fuzzy logic is that storing the rule-base might require a significant amount of memory. The number of rules grows exponentially to the number of variables. With n variables each of which can take m values, the number of rules in the rule-base is m^n . Adding spatial and temporal semantics to the decision process further increases the number of rules. Since sensor nodes have limited memory, storing a complete rule-base on every node might not be reasonable. In addition, constantly traversing a large rule-base might considerably slow down the event detection. To address this problem, we have designed a number of techniques that reduce the size of the rule-base. A key property of these techniques is that they do not decrease the event detection accuracy of the system.

The work presented in this chapter has three main contributions:

1. We show that using fuzzy logic results in more accurate event detection than when crisp values are used. We also show that the activity detection accuracy achieved when fuzzy logic is used is comparable to that achieved when using well established classification algorithms, such as Naive Bayes classifiers or decision trees.
2. We incorporate event semantics into the fuzzy logic rule-base to further improve the accuracy of event detection.
3. We have designed techniques that can be used to prevent the exponential growth of the rule-base without compromising the event detection accuracy.

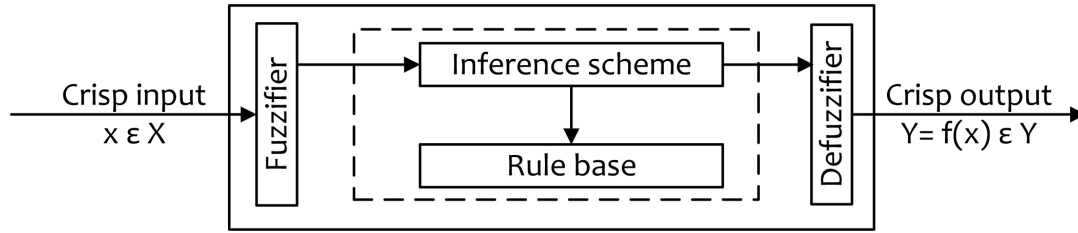


Figure 5.1: A fuzzy logic system contains three main components: a fuzzifier, decision making, which consists of an inference scheme and a rule-base, and a defuzzifier.

5.1 Overview of fuzzy logic

Figure 5.1 shows the structure of a general fuzzy logic system (FLS). The fuzzifier converts the crisp input variables $x \in X$, where X is the set of possible input variables, to fuzzy *linguistic variables* by applying the corresponding membership functions. Zadeh defines linguistic variables as “*variables whose values are not numbers but words or sentences in a natural or artificial language*” [121]. An input variable can be associated with one or more fuzzy sets depending on the calculated membership degrees. For example, a temperature value can be classified as both Low and Medium.

The fuzzified values are processed by *if-then* statements according to a set of predefined rules derived from domain knowledge provided by experts. In this stage the inference scheme maps input fuzzy sets to output fuzzy sets. Finally, the defuzzifier computes a crisp result from the fuzzy sets output by the rules. The crisp output value represents the control actions that should be taken. The above three steps are called fuzzification, decision making, and defuzzification, respectively. We describe each of them in more detail in the following subsections.

5.1.1 Fuzzification

The fuzzifier converts a crisp value into degrees of membership by applying the corresponding membership functions. A membership function determines the certainty with which a crisp value is associated with a specific linguistic value. Figure 5.2 shows an example of a temperature membership function. According to this membership function, a temperature value of -2°C is classified as 20% Freezing and 80% Cold. The membership functions can have different shapes. Some of the most frequently used shapes include triangular, trapezoidal, and Gaussian-shaped. Membership functions are defined by either relying on domain knowledge or through the application of different learning techniques, such as neural networks [122, 123] and genetic algorithms [124].

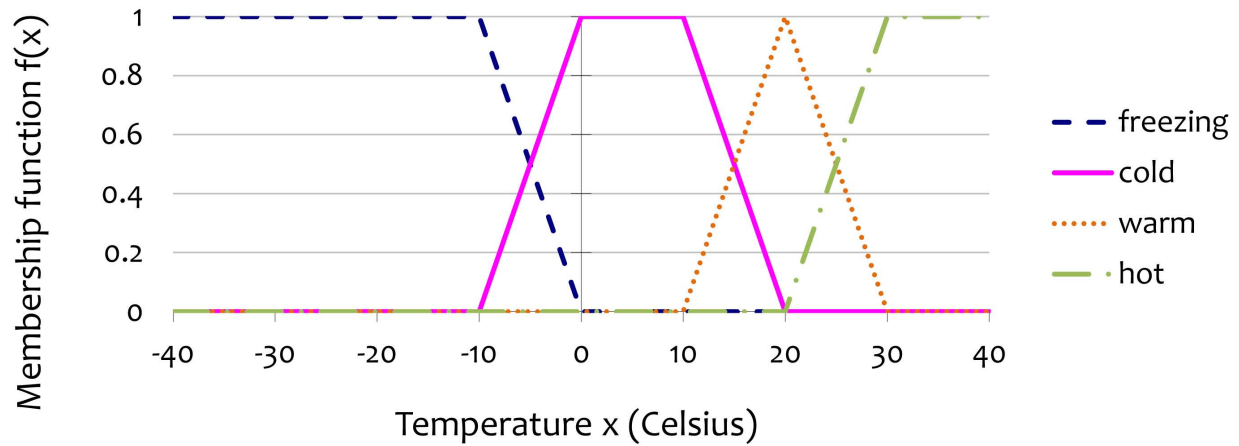


Figure 5.2: Temperature membership function. Using this membership function, a temperature value can be classified as: *Freezing*, *Freezing and Cold*, *Cold*, *Cold and Warm*, *Warm*, *Warm and Hot*, and *Hot*.

5.1.2 Decision making

A rule-base consists of a set of linguistic statements, called rules. These rules are of the form

IF premise, THEN consequent

where the *premise* is composed of fuzzy input variables connected by logical functions (e.g. AND, OR, NOT) and the *consequent* is a fuzzy output variable. The rule-base is usually generated as an exhaustive set of all possible value-combinations for the input linguistic variables that constitute the *premise*. Similarly to how membership functions are defined, the rule-base is derived either based on domain knowledge, or through using machine learning techniques. Consider a t-input 1-output FLS with rules of the form:

$$R^i : \text{IF } x_1 \text{ is } S_1^i \text{ AND } x_2 \text{ is } S_2^i \text{ AND } \dots \text{ AND } x_t \text{ is } S_t^i \text{ THEN } y \text{ is } A^i$$

When input $x' = \{x'_1, x'_2, \dots, x'_t\}$ is applied, the degree of firing of some rule R^i can be computed as:

$$\mu_{S_1^i}(x'_1) * \mu_{S_2^i}(x'_2) * \dots * \mu_{S_t^i}(x'_t) = T_{l=1}^t \mu_{S_l^i}(x'_l)$$

Here μ represents the membership function and both $*$ and T indicate the chosen triangular norm. A triangular norm is a binary operation, such as AND or OR, applied to the fuzzy sets provided by the membership functions [125].

5.1.3 Defuzzification

Executing the rules in the rule-base generates multiple shapes representing the modified membership functions. For example, a set of rules designed to decide the probability that there is a fire may produce the following

result: Low (56%), Medium (31%), and High (13%). Defuzzification is the transformation of this set of percentages into a single crisp value. Based on how they perform this transformation, defuzzifiers are divided into a number of categories. The most commonly used defuzzifiers are *center of gravity*, *center of singleton*, and *maximum methods* [125]:

- The *center of gravity* approach finds the centroid of the shape obtained by superimposing the shapes resulting from applying the rules. The output of the defuzzifier is the x-coordinate of this centroid.
- The defuzzification process can be significantly simplified if the *center of singleton* method is used. With this method, the membership functions for each rule are defuzzified separately. Each membership function is reduced to a singleton which represents the function's center of gravity. The simplification consists in that the singletons can be determined during the design of the system. The center of singleton method is an approximation of the center of gravity method. Although experiments have shown that there are slight differences between these two approaches, in most cases the differences can be neglected [126].
- The class of *maximum methods* determines the output by selecting the membership function with the maximum value. If the maximum is a range, either the lower, upper, or the middle value is taken for the output value depending on the method. Using these methods, the rule with the maximum activity always determines the output value. Applying this approach to the aforementioned fire detection example will produce a decision that there is a Low probability of fire and the other fuzzy values will be automatically ignored. Since the class of maximum methods shows discontinuous output on continuous input, these methods are not considered to be very suitable for use in controllers.

5.2 Event semantics

Sensors are generally believed to be unreliable and imprecise. Therefore, to increase our confidence in the presence of an event somewhere in the monitored area, we often need readings from multiple sensors and/or readings over some period of time. This could be achieved by instrumenting the event description logic with temporal and spatial semantics. We believe that this can significantly decrease the number of false positives. It will also allow us to describe and detect more complex events. To the best of our knowledge, no previous work on applying fuzzy logic to event detection has considered the effects of temporal and spatial semantics on the accuracy of event detection.

Consider, for example, a fire detecting scenario. A sensor network is deployed to monitor a building and trigger an alarm if a fire is detected. There are a number of temperature and smoke sensors in each room, as

Rule #	T_1	ΔT_1	T_2	ΔT_2	S	ΔS	Confidence
1	L	L	L	L	L	L	L
2	L	L	L	L	L	M	L
3	L	L	L	L	L	H	L
4	L	L	L	L	M	L	L
5	L	L	L	L	M	M	L
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
728	H	H	H	H	H	M	H
729	H	H	H	H	H	H	H

Table 5.1: An example fire detection rule-base. The size of the rule-base is determined by the number of linguistic variables and what values they can hold. In this rule-base, there are six linguistic variables, T_1 , ΔT_1 , T_2 , ΔT_2 , S , ΔS , each of which can hold a value of *Low* (L), *Medium* (M), or *High* (H).

well as in the hallways. The floors in the building are monitored separately, and there is a master node on each floor. The rest of the sensor nodes send their readings to the master node on their floor. Based on these readings, the master node determines if there is a fire or not. The fire detection is based not only on the temperature and smoke obscuration readings for a particular moment in time, but also on the rate of change of both the temperature and smoke levels. Therefore, our fire detection logic takes four linguistic variables as input - temperature (T), temperature change (ΔT), smoke obscuration (S), and smoke obscuration change (ΔS). The linguistic values for all four variables can be classified as Low (L), Medium (M), and High (H). The system might be able to achieve higher event detection accuracy if the linguistic variables have higher granularity, i.e. instead of only holding Low, Medium, or High values, they can also hold values such as Very Low, Low-Medium, Medium-High, and Very High. However, the designer of a WSN-based event detection system should use the smallest number of membership sets that can provide high event detection accuracy, while minimizing the size of the rule-base and the corresponding memory consumption.

In order to increase the accuracy of the fire detection scheme, we require that at least two temperature readings and one smoke reading are used to make a decision. Table 5.1 shows an example rule-base for this fire detection scenario. This rule-base, however, introduces a number of concerns which we address in the rest of this section.

5.2.1 Spatial semantics

One of the main goals when designing an event detection system is that the system is accurate and the number of false alarms is small. A way to achieve this is to include readings from multiple sensors in the decision process. For instance, we would be more confident that there is an actual fire if more than one node reports high temperature and smoke readings. If, for example, three sensors from the same room send reports indicating fire, the probability that there is an actual fire in that room is very high. In general, there

is a negative correlation between the distance among the sensors reporting fire and the probability of this report being true. Therefore, we include the concept of location in the event detection logic. We achieve this by augmenting the rules in the rule-base with a linguistic variable that serves as a spatial guard. This variable expresses the application requirements about the distance between the reporting sensors. In our fire detection scenario, we can name this variable *distance* and classify it as Close (C), Distant (D), and Far (F), for example. Incorporating this *distance* variable into the rule-base, however, changes the format of the rules and adds an extra column to the rule-base. Now the format of the rules in Table 5.1 changes to:

IF T_1 is H AND ΔT_1 is H AND T_2 is H AND ΔT_2 is H AND S is H AND ΔS is H AND *distance* is F, THEN Fire is M.

5.2.2 Temporal semantics

To further decrease the number of false alarms, we also need to take into account the temporal properties of the monitored events. The event detection confidence is higher if the sensor readings indicating that a particular event has occurred have been generated within a short period of time of each other. We call the length of this time period *temporal distance* of the readings. The event detection confidence decreases as the temporal distance between the sensor readings increases.

Temporal semantics are especially important for WSNs because of the inherent nature of sensor communication. It is very likely for messages in a WSN to be delayed because of network congestions or routing problems. Consequently, a reliable event detection rule-base should take into consideration the generation times of the sensor readings. To accommodate this, we include another linguistic variable that serves as a temporal guard. This variable, *time*, represents the difference in the generation times of the sensor readings. For example, in our fire detection scenario, *time* could have three semantic values: Short (S), Medium (M), and Long (L). In this way, the information about the time interval within which the sensor readings have been generated is included in the decision process. This will further change the format of the rule-base in Table 5.1 to:

IF T_1 is H AND ΔT_1 is H AND T_2 is H AND ΔT_2 is H AND S is H AND ΔS is H AND *distance* is F AND *time* is M,
THEN Fire is M.

5.3 Decreasing the size of the rule-base

Augmenting the rule-base with temporal and spatial variables increases the number of rules. As mentioned earlier, the size of the rule-base grows exponentially to the number of linguistic variables. In our fire monitoring example, where the only sensor readings we consider are temperature and smoke, the full rule-base has

Rule #	T	ΔT	Confidence
1	L	L	L
2	L	M	L
3	L	H	M
4	M	L	L
5	M	M	M
6	M	H	H
7	H	L	M
8	H	M	H
9	H	H	H

Table 5.2: Rule-base for a temperature sensor. The two linguistic input variables and the fire confidence consequent can be classified as Low (L), Medium (M), or High (H).

$3^8 = 6561$ rules, since we use 8 linguistic variables and each variable can hold 3 different values. In more complicated scenarios that require more than two types of sensors, the number of rules in the fuzzy rule-base could be much higher. Storing such rule-bases might be a challenge for memory constrained sensor nodes. In addition, traversing the full rule-base every time there are new sensor readings will slow down the event detection. To address these concerns, we have designed three techniques to help reduce the number of rules. We demonstrate how these techniques are applied on a relatively small rule-base with a few linguistic variables that can take three values - Low, Medium, and High. However, these techniques can be applied in the same fashion to larger rule-bases that contain more linguistic variables characterized by more complex membership functions.

Although the rule-base reduction techniques alleviate both the storage problem and the rule traversal process, they might have a negative effect on the event detection accuracy. Therefore, maintaining high event detection accuracy was a primary goal when designing the reduction techniques described in this section. We achieve this by carefully modifying the rule-base through merging important rules and removing the rules that do not affect the detection accuracy of the events of interest.

5.3.1 Separating the rule-base

The first technique we use to reduce the size of the rule-base is to separate the rules on a “need to know” basis. Each node stores only the rules corresponding to the types of sensors it has. If, for example, some of the nodes in our fire detection scenario are only equipped with temperature sensors, they do not need to store the whole rule-base. Instead, they store a smaller modified rule-base similar to the one shown in Table 5.2. This rule-base contains only rules with premise linguistic variables based on the values from the temperature sensors. In this way, the event detection logic on each node considers only the rules that are

Rule #	T	ΔT	Confidence
1	L	$\leq M$	L
2	L	H	M
3	M	L	L
4	M	M	M
5	M	H	H
6	H	L	M
7	H	$\geq M$	H

Table 5.3: Reduced rule-base for a temperature sensor. In this rule-base, rules with similar consequents have been combined with the help of the \leq and \geq operators.

relevant to the node's sensor readings. This separation simplifies the decision process and makes the rule-base traversal faster. The rule-base for the smoke sensors can be constructed in a similar way.

5.3.2 Combining rules with similar outcomes

Rules 1 and 2 in Table 5.2 have the same outcome and only differ in the values of ΔT . This observation is also valid for rules 8 and 9. Combining these rule couples could help us further decrease the size of the rule-base. For the rule-base in Table 5.2 applying such an optimization leaves us with 7 rules. The rules, however, have a slightly different syntax. Instead of:

$$R^i : \text{IF } x_1 \text{ is } S_1^i \text{ AND } x_2 \text{ is } S_2^i \text{ AND } \dots \text{ AND } x_t \text{ is } S_t^i \text{ THEN } y \text{ is } A^i$$

some of the rules have the following different form:

$$R^i : \text{IF } x_1 \text{ is } \leq S_1^i \text{ AND } x_2 \text{ is } S_2^i \text{ AND } \dots \text{ AND } x_t \text{ is } \geq S_t^i \text{ THEN } y \text{ is } A^i$$

In the modified rules \leq stands for “in this fuzzy set or in fuzzy sets smaller than it” and \geq stands for “in this fuzzy set or in fuzzy sets greater than it”. Table 5.3 shows the result of applying this reduction technique on the rule-base in Table 5.2.

5.3.3 Incomplete rule-base

A rule-base is considered *complete* if there are rules for every possible combination of the input variables. However, only some of these combinations have outcomes that are important to the event detection system. For example, rules containing variables which do not satisfy the temporal and spatial constraints cannot trigger an alarm. Therefore, the rules with *distance* variable Distant or Far can be removed from the rule-base. This step leaves us with just a third of the original number of rules in the rule-base. Similarly, applying the same approach to the *time* variable and removing the rules with values Medium and Long decreases the rule-base by yet another two thirds.

In addition, if we exclude the rules with consequents that are of no interest to the event detection system, such as rules indicating that the possibility that a fire has occurred is Low, we reduce the size of the rule-base even more. As a result, by lowering the level of completeness of the rule-base, we significantly decrease the number of rules that need to be stored on the sensor nodes. This “trimming” process, however, should be performed very carefully in order to prevent the removal of important consequents. To make sure that the system knows how to proceed if none of the rules in the rule-base has been satisfied, we introduce a *default* rule that is triggered if no other rule has been satisfied.

5.4 Evaluation

We use the FuzzyJ Toolkit for Java [127] to implement the necessary fuzzy logic functionality. To avoid the danger, cost, and non-repeatability of creating fires, we perform trace-based simulations using real fire data publicly available on the National Institute of Standards and Technology (NIST) website [128]. The study they conduct provides sensor measurements from a number of different real fires as well as nuisance scenarios. We have used three of the available real fire scenarios: fire caused by a burning mattress, fire caused by a burning chair, and cooking oil fire. The purpose of the nuisance tests is to study common household nuisance alarm scenarios. We have used two of these tests in our experiments: frying margarine and broiling hamburgers.

5.4.1 Experiments using real fire data

The membership functions for the smoke and temperature input linguistic variables used in the experiments are shown in Figure 5.3. In addition to the temperature and smoke obscuration variables, we also take into consideration the temperature and smoke obscuration difference between two consecutive readings. These two additional variables give us a notion of how fast the temperature and smoke obscuration are changing.

Figure 5.4 shows the membership function for the output fire confidence. This linguistic variable represents the system’s confidence in the presence of fire. For example, if the fire confidence value is higher than 80, we are more than 80% certain that there is a fire. If the fire confidence is smaller than 50, it is more likely that there is no fire.

In the system model we use for our simulations every node decides locally if a fire event has occurred. If it decides that a fire is present, a node forwards its decision to the master node for the house. An alternative system model, where the nodes send a subset of their readings to the master node, and the master node makes a decision, is also possible. In this model, the base node has the aggregated information from all sensors, and might be able to make more accurate decisions. However, because of the increased amount of

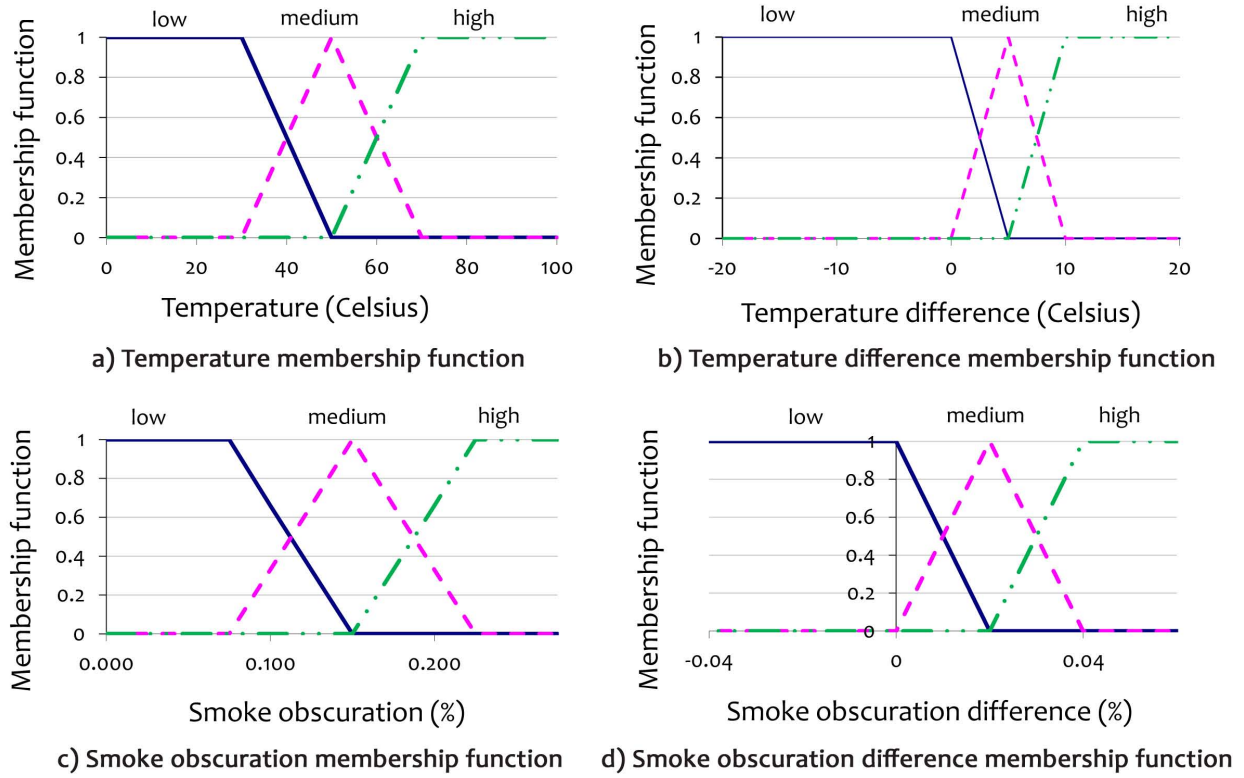


Figure 5.3: Membership functions for the four input linguistic variables: temperature, temperature difference, smoke obscuration, and smoke obscuration difference.

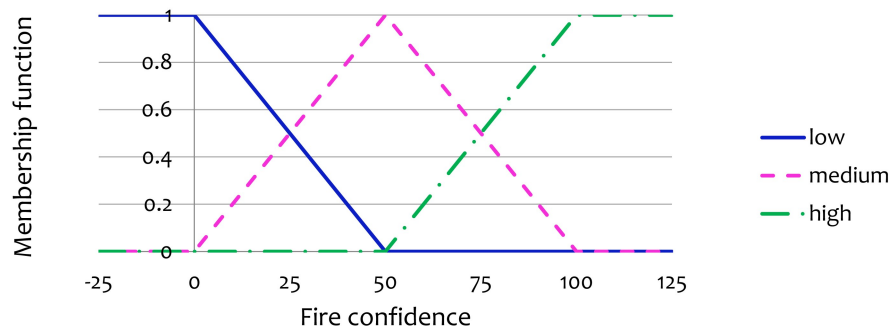


Figure 5.4: Fire confidence membership function.

communication, the lifetime of the network might decrease. Therefore, which model is appropriate depends on the nature of the application and the lifetime requirements of the network.

To provide a baseline for our results, we performed crisp-value experiments with the burning mattress, burning chair, and cooking oil data. The temperature and smoke obscuration thresholds used in the crisp logic experiments are threshold values used in commercial smoke and heat detectors, 55°C and 0.15 m^{-1} , respectively [129, 130]. The membership functions in Figure 5.3 were also built according to these threshold values. We used the commercial crisp thresholds as the border between Low and High, which in our scenario

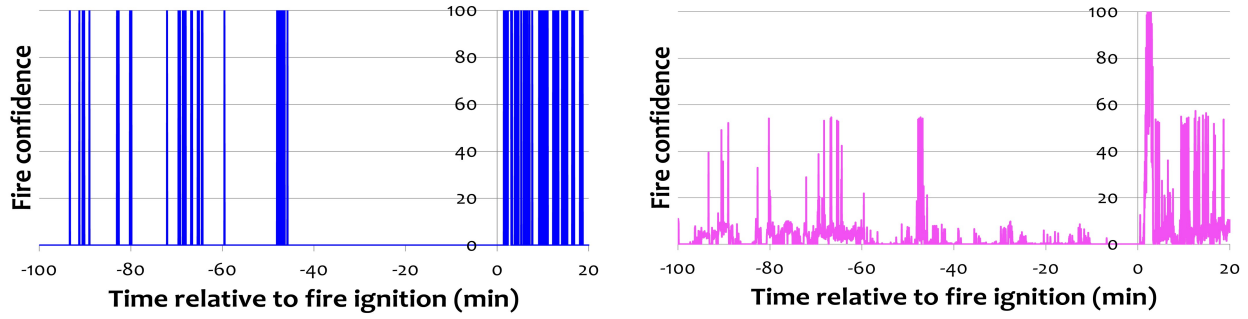


Figure 5.5: Burning mattress simulation: a) crisp value detection b) fuzzy value detection.

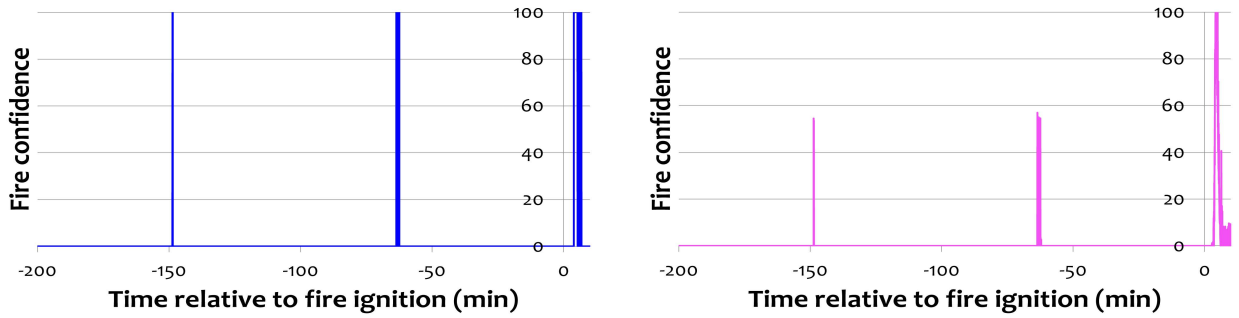


Figure 5.6: Burning chair simulation: a) crisp value detection b) fuzzy value detection.

is classified as 0% Low, 100% Medium, and 0% High for all four linguistic variables. We relied on domain knowledge to determine the remaining details of the membership functions.

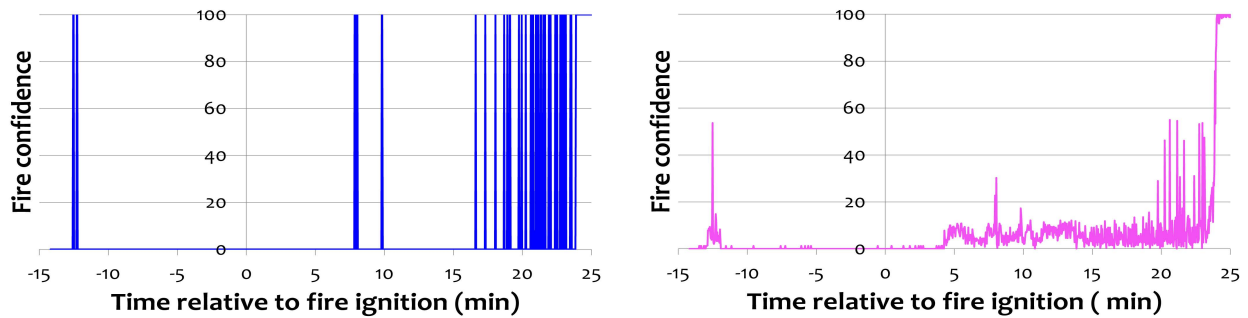


Figure 5.7: Burning oil simulation: a) crisp value detection b) fuzzy value detection.

The results from the crisp-value experiment are shown in Figure 5.5a), Figure 5.6a), and Figure 5.7a). In these and all following figures, the origin of the graph represents the time of fire ignition. As we can see from the three figures, using crisp values resulted in a very large number of false fire detections. In the burning mattress scenario in particular, there were 40 false fire detections in the period prior to the fire ignition, which constitutes about 1.3% of the measurements. This considerable number of false positives significantly affects the efficiency and fidelity of an event detection system. Admittedly, part of these false positives can

be attributed to the aggressive crisp value thresholds. However, if the thresholds are set higher, this could lead to failures in detecting actual fires. In a real fire detection system it is more important to decrease the number of false negatives than that of false positives. Therefore, we have kept the threshold values in compliance with the commercial standards.

What we wanted to investigate with our next set of experiments was whether fuzzy logic can do better in terms of false positives, while still reporting promptly the presence of a fire when one actually occurs. In the first set of fuzzy logic experiments, a node decides if there is a fire based only on its own readings. The readings of neighboring sensor nodes are not considered as inputs to the decision process. The values of the linguistic variables used in the decision process can be classified as Low (L), Medium (M), and High (H), as shown by Figure 5.3 and Figure 5.4. We have used heuristics to build the rule-base for our fire detection experiments. In cases where this is not possible, for example, when more complex events are to be detected, domain experts could be consulted for the definition of the rule-bases. The rule-base for these experiments is shown in Table 5.4. The complete rule-base has 81 rules. Therefore, we show the rule-base after our second reduction technique has been applied.

The results from our first set of fuzzy logic experiments, a burning mattress, a burning chair, and cooking oil fire, are presented in Figure 5.5b), Figure 5.6b), and Figure 5.7b), respectively. As we can see, the fuzzy logic event detection mechanism performs very well. It detects the presence of a fire shortly after the ignition. In addition, unlike the crisp-value fire detection, there are no false positives. All three graphs show fire confidence around 0 before the ignition, except for a number of small peaks when the confidence increases to 54, which is close to 100% Medium. At the same times when the fuzzy-value peaks occur, we can also notice crisp-value peaks but with much higher confidence. The raw sensor data revealed that the peaks were caused by a number of one-second-long reports of increased smoke values. This proves our hypothesis that fuzzy logic is able to accommodate the often imprecise sensor readings. Even in the cases when the nodes erroneously report the presence of smoke, the fuzzy logic mechanism keeps the fire confidence low enough so that a false alarm is not triggered.

Rule	Temperature	Δ Temperature	Smoke	Δ Smoke	Confidence
1	L	L	\leq M	\geq L	L
2	L	L	H	\leq M	L
3	L	L	H	H	M
4	L	\geq M	L	L	L
5	L	H	L	M	L
6	M	L	L	\geq L	L
7	M	L	M	L	L
8	H	L	L	L	L
9	L	M	L	\geq M	M
10	L	M	M	\geq L	M
11	L	M	H	\leq M	M
12	L	H	L	H	M
13	L	H	\geq M	L	M
14	L	H	M	M	M
15	M	L	M	\geq M	M
16	M	L	H	\leq M	M
17	M	M	\leq M	\leq M	M
18	M	M	H	L	M
19	M	H	\leq M	L	M
20	H	\leq M	L	M	M
21	H	\geq M	L	L	M
22	L	M	H	H	H
23	L	H	M	H	H
24	L	H	H	\geq M	H
25	M	L	H	H	H
26	M	M	\geq L	H	H
27	M	M	H	M	H
28	M	H	\geq L	\geq M	H
29	M	H	H	L	H
30	H	\geq L	L	H	H
31	H	\geq L	\geq M	\geq L	H
32	H	H	L	M	H

Table 5.4: Fire detection rule-base for the scenario where a node decides if there is a fire based only on its own sensor readings. The Temperature, Temperature difference, Smoke, and Smoke difference variables take Low (L), Medium (M), and High (H) values.

We also evaluate how including neighbor node values in the decision process affects the detection accuracy. The average of the neighbor values is represented with an additional linguistic variable that we include in the decision rules. In addition, in order to meet the spatial and temporal requirements of the application, we only consider readings 1) received from neighbor nodes that are located close to the current node, and 2) that have been generated within 1 second from the current reading of the node. The results in Figure 5.8, Figure 5.9, and Figure 5.10 show that fire is detected almost as quickly as when the decision process is only based on *own* sensor readings. Although the peak areas are still present, the corresponding fire confidence values are lower when the neighbor readings are included in the decision process. This shows that including the readings of neighbor nodes in the decision process positively affects the detection accuracy.

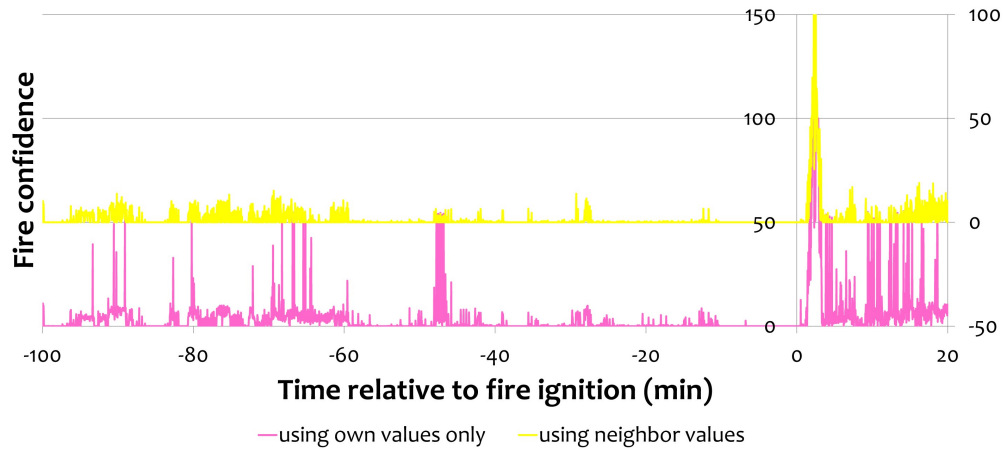


Figure 5.8: Simulating a burning mattress: including neighbor readings in the decision. The results when only own values are used are plotted on the first y-axis. Including the neighbor values is plotted on the second y-axis.

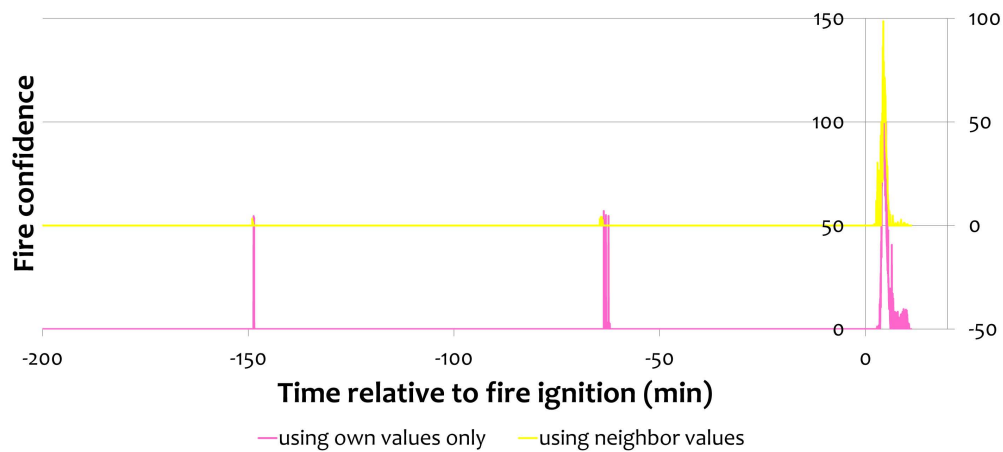


Figure 5.9: Simulating a burning chair: including neighbor readings in the decision. The results when only own values are used are plotted on the first y-axis. Including the neighbor values is plotted on the second y-axis.

Figure 5.10 allows us to make an important observation. In the burning oil scenario, when the fire detection is based on the readings of a single sensor, the system reaches fire confidence of 100 around 23 minutes after the stove has been turned on. However, when the readings of neighbor sensors are considered in the detection process, the maximum fire confidence never exceeds 71, which is approximately 60% Medium and 40% High. This is due to the fact that the neighbor sensors are located further away from the fire and, therefore, their temperature readings have lower values. These results come to show that sensor network designers should be careful when determining the size and radius of a sensor's neighborhood. Although including readings from neighbor sensors improves the event detection accuracy of the system, when these neighbor sensors are located too far from where the event has occurred, this might have a negative effect on

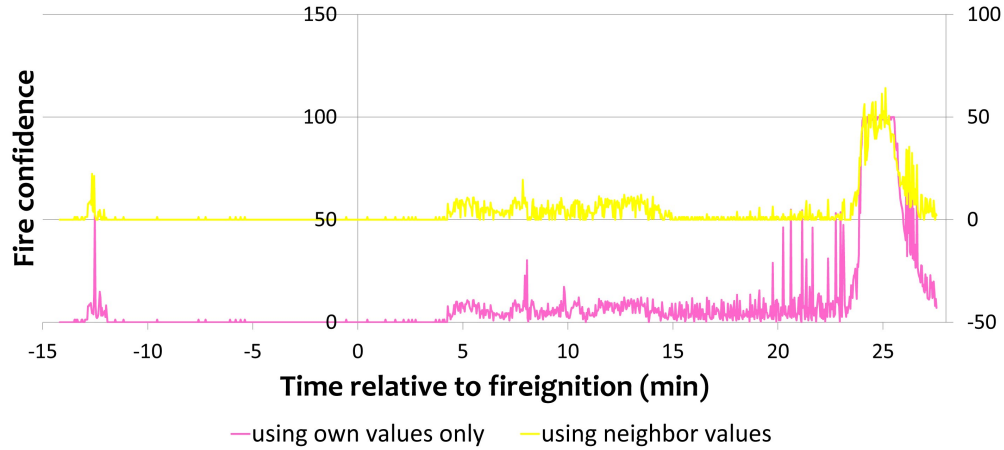


Figure 5.10: Simulating burning oil: including neighbor readings in the decision. The results when only own values are used are plotted on the first y-axis. Including the neighbor values is plotted on the second y-axis.

the detection accuracy.

5.4.2 Experiments using nuisance fire data

The goal of these experiments is to study the behavior of our fuzzy-value fire detection mechanism when it is presented with a nuisance scenario. The Smoke Detector Operability Survey: Report on Finding [131] conducted by the U.S. Consumer Products Safety Commission reported that about 50% of the 1012 participants indicated that they had experienced nuisance alarms, with 80% of those attributed to cooking activities, and an additional 6% citing steam from bathrooms. Dust and tobacco smoke are also mentioned sources. The survey also reported that for the alarms with missing or disconnected batteries, or disconnected AC power, more than one third of respondents indicated that power was removed due to nuisance alarms.

The NIST data provided for the nuisance scenarios differs from that for the actual fires in that the smoke obscuration measurements are not provided. Therefore, we have substituted the two input linguistic variables based on smoke obscuration with two new variables based on aerosol mass concentration. Similarly to the smoke obscuration, the aerosol mass concentration allows us to determine the amount of aerosol particles in the air. The thresholds for the mass concentration and mass concentration difference linguistic variables were chosen based on previous mass concentration alarm research [132]. Figure 5.11 shows the membership functions for the two new linguistic variables.

As with the real fire scenarios, we use crisp-value detection as a baseline. Figure 5.12 shows the results from the crisp-value experiments of frying margarine and grilling hamburgers, respectively. In both scenarios no actual fire occurred. However, as we can see from the figure, the number of false fire detections is high:

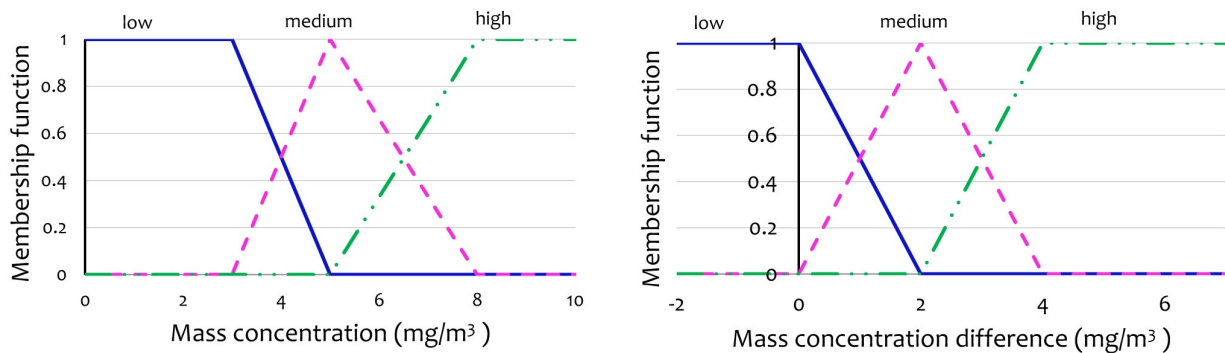


Figure 5.11: Membership functions for the Mass Concentration input linguistic variables.

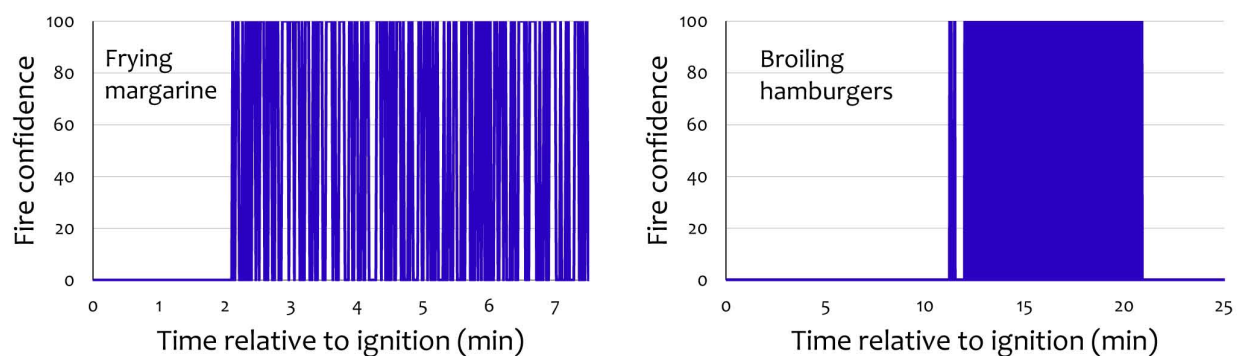


Figure 5.12: Crisp value simulation: a) Frying margarine b) Broiling hamburgers.

172 (34% of all readings) for the frying margarine scenario and 248 (16% of all readings) in the broiling hamburgers scenario.

The results from the fuzzy-value experiments are shown in Figure 5.13. For both scenarios the fire detection confidence follows the same pattern as in the crisp-value experiments. The peaks that are present in Figure 5.13 a) and b) are also present in Figure 5.12 a) and b), respectively. Similarly to the real fire experiments, the difference between the peaks is that in the fuzzy-value scenarios the peaks never reach high confidence. This means that, unlike the cases when crisp values are used, an alarm will not be triggered.

An interesting observation is that in Figure 5.13a) some of the fire confidence peaks reach levels as high as 50%. All of these peaks are grouped around the fifth minute of the experiment. At that time, the aerosol mass concentration increases above $100\text{mg}/\text{m}^3$ with maximum $214\text{mg}/\text{m}^3$. This is the time when the frying caused the highest level of smoke. However, despite these high mass concentration values, the fuzzy logic system manages to determine that no fire is currently present.

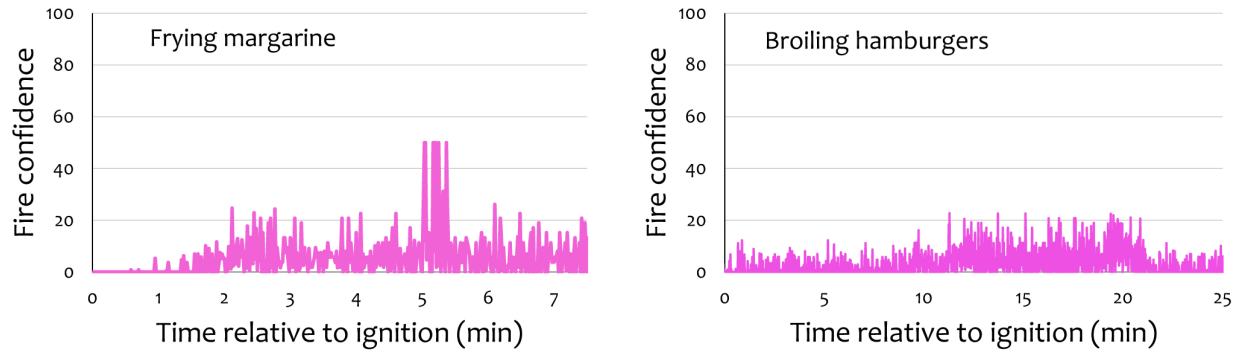


Figure 5.13: Fuzzy value simulation: a) Frying margarine b) Broiling hamburgers.

5.4.3 Analysis

Why does fuzzy logic perform better?

An interesting question is why fuzzy logic is more precise than crisp-value logic. From the considerable decrease in the number of false positives, it appears that fuzzy logic handles the fluctuating sensor readings much better. To understand why this happens we take a closer look at the first false fire detection reported by the crisp-value logic. In the burning mattress scenario this occurs approximately 12 minutes into the experiment. The values that cause the false alarm are: $T = 25.21^{\circ}\text{C}$, $\Delta T = 0^{\circ}\text{C}$, $S = 0.203\%$, and $\Delta S = 0.109\%$. Since the smoke level (S) and the smoke change level (ΔS) are both classified as High, the crisp logic concludes that there must be a fire.

What does the fuzzy logic event detection do differently? According to the membership functions in Figure 5.3, temperature value of 25.21°C is classified as 100% Low; temperature change of 0°C is classified as 100% Low; smoke obscuration level of 0.203% is classified as 33% Medium and 66% High; and smoke obscuration change of 0.119% is classified as 100% High. The decision making process checks which rules from the rule-base are satisfied. These are rules 1 and 3 from the rule-base in Table 5.4. Based on those rules, the defuzzifier reports a fire confidence value of 39.4. This value maps to fire confidence which is 20% Low and 80% Medium. Such level of confidence, however, is not enough to cause the system to report a fire.

This example illustrates why a fuzzy logic event detection system tends to perform better than a crisp one in the presence of short-lasting inaccurate sensor readings, which often occur in WSNs. Fuzzy logic takes into account the certainty with which an event occurs, instead of making binary decisions based on crisp values and fixed thresholds, which improves the accuracy of event detection.

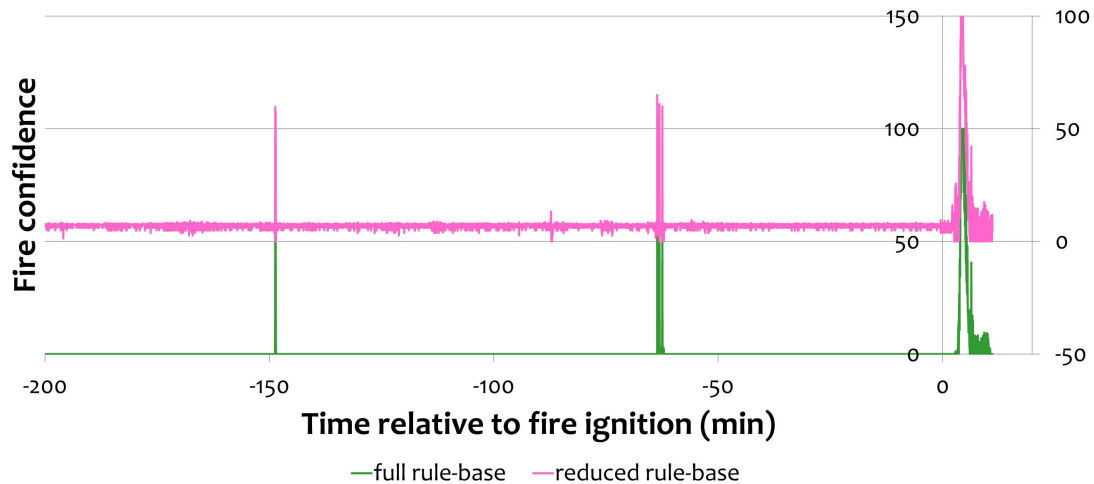


Figure 5.14: Simulating a burning chair with a reduced rule-base. The results when the full rule-base is used are plotted on the first y-axis. Using the reduced rule-base is plotted on the second y-axis.

Decreasing the rule-base

We applied our reduction techniques to the full version of the rule-base shown in Table 5.4. All nodes in the simulation are equipped with both a smoke and a temperature sensor which makes the first technique not applicable. Therefore, we only used the second and third reduction techniques. The rule-base initially has 81 rules. Combining the rules with similar outcomes reduces the number of rules to 32, as shown in Table 5.4. This evaluates to a decrease of 60%. In general, when there are more than two input linguistic variables, applying the second method decreases the rule-base by approximately two thirds. Excluding the rules that result in *Low* fire confidence additionally reduces the size of the rule-base to 25, which is 31% of the original rule-base.

We have compared the behavior of the fire detection system when the full and the reduced rule-bases are used. Figure 5.14 shows the results for the burning chair scenario. The fire confidence is consistently higher when the reduced rule-base is used. However, since this confidence remains *Low*, this does not cause false fire detections.

Detection accuracy

To further understand the behavior of our fuzzy logic approach, we have compared it to two well established classification algorithms: a naive Bayes classifier [133] and a J48 decision tree which is an open source implementation of the C4.5 algorithm [134]. Fuzzy logic is more suitable than these two algorithms for WSN event description since, unlike Bayes classifiers and decision trees where values are considered to be discrete,

	Naive Bayes		J48 Decision Tree		Fuzzy logic	
	number	percent	number	percent	number	percent
Burning chair	105	1.56%	7	0.13%	0	0
Burning mattress	89	2.35%	5	0.13%	0	0

Table 5.5: Number of incorrect classifications by a Naive Bayes classifier and a J48 Tree.

Scenario	Crisp values	Fuzzy values	Plus neighbor readings	Reduced rule-base
Burning chair	236	236	248	236
Burning mattress	103	97	117	97
Cooking oil fire	1431	1431	1443	1431

Table 5.6: Fire detection delay in seconds.

it works with continuous values, which is exactly what the sensor readings are. In addition, specifying the membership functions is simpler and computationally more efficient than building a probability model.

We ran this set of experiments using the Weka data mining tool [135]. The input values to the classification algorithms were the same as the ones used in the fuzzy logic experiments - temperature, temperature difference, smoke obscuration, and smoke obscuration difference. We performed a 10-fold cross validation for both classification algorithms. Table 5.5 shows the number of incorrectly classified instances for the first two fire scenarios, burning mattress and burning chair, as well as what percentage of the total instances was incorrectly classified. Both algorithms produce a number of inaccurate classifications.

Although the percentage of the erroneously classified instances is low, it is higher than the number of misclassifications introduced by fuzzy logic. This could be attributed to the specifics of the datasets we have analyzed; it is possible that fuzzy logic is better suited to process data containing sporadic outliers caused by erroneously high sensor values. However, we expect that in different application scenarios, fuzzy logic will still be able to provide comparable activity detection accuracy to that achieved by Naive Bayes and tree classifiers.

Fire detection delay

Table 5.6 shows the delay incurred by the different fire detection mechanisms. Fire is detected just as fast, and in the burning mattress scenario even faster, when fuzzy values are used. In addition, decreasing the size of the rule-base does not delay the fire detection. We also notice that including the readings of neighbor sensors in the decision process slightly slows down the detection. This is not surprising since not all sensors are located at the same distance from the fire, and, therefore, they start registering abnormal values at different times. Consequently, if a sensor is waiting for its neighbors to also detect the fire, and those neighbors are located further away from the fire source, the detection might be slightly delayed.

5.5 Summary

A disadvantage of the current event detection approaches used in WSNs is that they cannot properly handle the often imprecise sensor readings. We show that fuzzy logic is a powerful and accurate mechanism which can successfully be applied to event detection in WSNs. Compared to using crisp values, fuzzy logic allows the detection algorithm to maintain a high accuracy level despite fluctuations in the sensor values. This helps decrease the number of false positives while still providing fast and accurate event detection. Our experiments support the hypothesis that incorporating the readings of neighbor nodes in the decision process further improves the event detection accuracy.

The evaluation also shows that the rule-base reduction techniques we have developed are efficient and preserve both the correctness and the timeliness of event detection. Using two of these techniques, *combining rules with similar outcomes* and *incomplete rule-base*, reduced the size of our experimental rule-base by more than 70%. Further, compared to two well-established classification algorithms, fuzzy logic provides comparable event detection accuracy.

Part IV

Robustness to node failures

Chapter 6

Run time assurance (RTA)

EMERGING wireless sensor network technologies are applicable to a wide range of mission-critical applications, including fire fighting and emergency response, infrastructure monitoring, military surveillance, and medical applications. These applications must operate reliably and continuously due to the high cost of system failure. However, continuous and reliable operation of WSNs is extremely difficult to guarantee due to hardware degradation and environmental changes, which can cause operating conditions that were impossible for the original system designers to foresee. This is particularly true for applications that operate over long time durations, such as a building monitoring system that must operate for the lifetime of the building. Wireless noise and interference may change dramatically as new wireless technologies are developed and deployed in or near a building, and sensor readings and network topology may change as the occupancy, activities, and equipment in a building evolve over time.

In this chapter, we describe and demonstrate a methodology for *run-time assurance* (RTA), in which we validate at run time that a WSN will function correctly in terms of meeting its high-level application requirements, irrespective of any changes to the operating conditions since it was originally designed and deployed. The basic approach is to use program analysis and compiler techniques to facilitate automated testing of a WSN at run time. The developer specifies the application using a high-level specification, which is compiled into both the code that will execute the application on the WSN, and a set of input/output tests that can be used to verify correct operation of the application. The test inputs are then supplied to the WSN at run time, either periodically or by request. The WSN performs all computations, message passing, and other distributed operations required to produce output values and actions, which are compared to the expected outputs. This testing process produces an end-to-end validation of the essential application logic.

RTA differs from network health monitoring, which detects and reports low-level hardware faults, such as

node or route failures [80, 78]. The end-to-end application-level tests used for RTA have two key advantages over the tests of individual hardware components used for health monitoring: 1) fewer false positives - RTA does not test nodes, logic, or wireless links that are not necessary for correct system operation, and therefore produces fewer maintenance dispatches than health monitoring systems; 2) fewer false negatives - a network health monitoring system will only validate that all nodes are alive and have a route to a base station, but does not test more subtle causes of failure such as topological changes or clock drift. In contrast, the RTA approach tests the ways that an application may fail to meet its high-level requirements because it uses end-to-end tests. Network health monitoring improves system reliability by detecting some types of failures, but stops short of actually validating correct system operation. The goal of RTA, instead, is to provide a positive affirmation of correct application-level operation.

We have implemented a framework for designing and automatically testing WSN applications using the RTA methodology. The developer specifies the application using MEDAL. Our system compiles the model down to TinyOS [136] code that runs on the Telos nodes [137], and tests the defined mappings between sensor input values and system outputs. We use program analysis techniques to identify the minimal set of tests that will cover the essential application logic. This analysis uses new techniques that exploit information about network topology and the redundancy of nodes based on sensing range, and builds on existing techniques to cover all execution paths in the program [138]. Once the minimal set of tests has been identified, our system deploys the TinyOS code onto the network and periodically executes the tests. This implementation serves as a proof of concept of our RTA methodology, which can also be applied more generally to other programming models.

We evaluate our implementation by designing a fire detection system and executing it on a network of 21 TelosB nodes. We artificially introduce failures into the system, including node failures and location errors, and compare the performance of RTA to that of an existing health monitoring solution [78]. Our results indicate that RTA misses 75% fewer system failures and also produces 70% fewer maintenance dispatches than health monitoring. Furthermore, our program analysis techniques reduce the number of tests required such that RTA incurs 33% less messaging overhead than health monitoring. The main contributions of the work presented in this chapter are:

1. A novel RTA methodology that positively affirms correct system operation at run time.
2. A prototype implementation based on a formal sensor network description language.
3. New analysis techniques exploiting network topology and sensing redundancy to reduce the number of necessary tests.

4. A quantitative evaluation of our RTA methodology and implementation, and comparison with an existing network health monitoring system.

6.1 RTA Methodology

The RTA methodology is built around the following three principles:

Run time verification: The RTA principle requires that a system demonstrates at run time that it is able to perform its key services. Currently, testing and debugging techniques are used to fully test the system at design time, and deployment-time validation [139] is used to verify the system during deployment. However, due to the changing environment and the dynamic nature of failures, we argue that even when these techniques have been employed, RTA is still necessary.

Application-level guarantees: Many sensor networks are complicated systems, consisting of numerous components and protocols. Each component may use various fault tolerance, self-healing, or other reliability mechanisms to operate robustly. However, even if one can guarantee that each separate component works correctly, the system may still fail to perform some high-level operations. And a user (such as a fire inspector) is only concerned with whether the system performs the way they want, rather than whether each component works correctly. The goal of the RTA principle is to address this and focus on verifying the application-level services.

Correctness demonstration: There are different ways to demonstrate correctness of services at run time. One option is to monitor system health information and infer system correctness from this information. Such health monitoring techniques are shown to be effective for certain applications with regular traffic [140, 141, 74]. However, many critical events, such as fire or volcano eruptions, are rare. Also, in complex WSN systems, it is hard to determine which states should be monitored and how to infer the correctness of services. Monitoring too many states is inefficient and may cause many false positives, but monitoring too few states could fail to reveal failures. Memento [78] uses a periodic heart beat method to determine if a node is still alive. This approach is not suitable for RTA for two main reasons: 1) node responsiveness does not imply proper functioning of the system and its application semantics; 2) node failure does not imply the failure of the overall system or the application. If we have some level of node redundancy, a small number of node failures may not affect the system application at all. Thus, in general, health monitoring techniques are not sufficient to provide confidence in application-level requirements for WSNs at run time. To address this, we employ testing to verify the proper operation of the application. Using tests allows us to check if the application provides the results we expect regardless of what the state of its components might be.

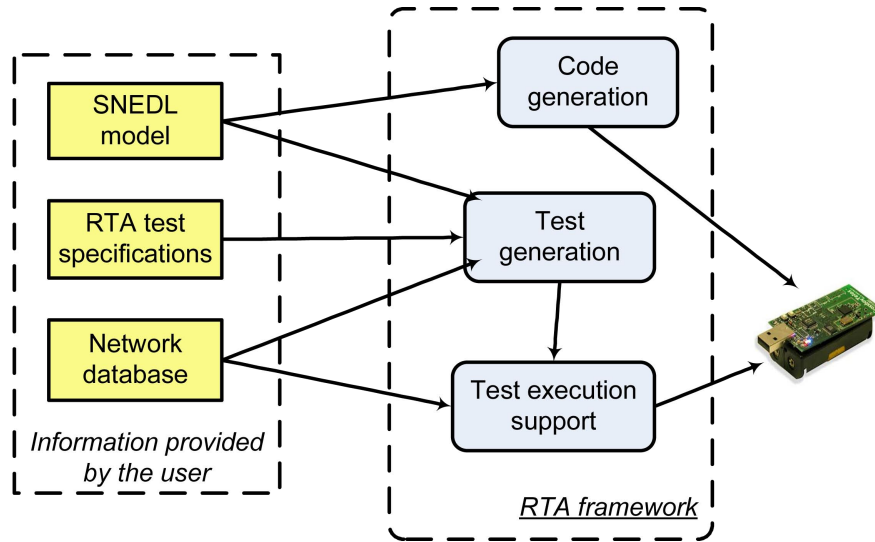


Figure 6.1: The main components of the RTA framework are the automatic code generator, the automatic test generator, and the test execution support.

6.2 Implementation Framework

Figure 6.1 shows how the components of the RTA framework interact with each other. The framework needs three inputs: the MEDAL model of the application logic, the test specification, and the topology of the network. Both the automated code generation and the automated test generation mechanisms need the MEDAL application model. Note that the RTA framework is flexible enough to use other modeling languages, as long as they are able to clearly and unambiguously define the application-level behavior of the system. The test generation mechanism also needs information about what tests the user wants to run (the test specification), and how many and what nodes there are in the regions of the network that will be tested (the network topology). After the code for the nodes has been generated and deployed, and the proper sets of tests have been created, the RTA execution mechanism can start monitoring the application's functionality by running RTA tests on the sensor network.

6.2.1 The MEDAL Programming Model

An advantage of MEDAL is that it provides a different perspective of a WSN system. By representing a WSN application as a MEDAL model we can view it as a flow of tokens from places in the Petri net to other places. There are several reasons why this is useful for the RTA specification. First, this is very suitable for testing purposes and makes running RTA tests extremely straightforward. Second, a token-flow model allows us to easily differentiate between a real event and a test event. Since the test events are specified by test-tokens, all we have to do is mark the test-tokens as such. Then, whenever a transition is triggered or a

place is reached, we can always check the type of the token that caused this to happen and react accordingly. Third, such a model helps the collection of event-logs and makes it easy to define flow-traces in the system.

The MEDAL model of the WSN application is one of the inputs to our RTA framework. The user is asked to write a script to specify the transitions and places of their model as well as the connections among them. We provide a set of predefined transition types which, we believe, covers the majority of logical operations used in a WSN application, such as, greater, less, equal, minimum, maximum, difference between two consecutive values, average, and moving average. If, however, the user wants to specify transitions with more complex functionality, they can do that by writing the necessary code themselves.

6.2.2 Automated Test Generation

For more than a decade automatic test generation has been widely used in software testing. It has improved the quality of testing and reduced the effort and time spent on testing. Despite its many advantages, this approach has not been applied to WSN application testing. One of the main benefits of the RTA framework is that it also provides automatic test generation. The RTA test generator takes three inputs:

1. The MEDAL model of the application. Since the application code is automatically generated based on the MEDAL model, we analyze the model itself.
2. The network topology.
3. The test specification - the user provides information about the events they want tests generated for, the areas of the network they want to test, and the times the test should be run. Consider the following scenario: a user wants to test the network for the occurrence of fire (event *EFire*). They want the tests to be run on the nodes in two rooms (Room1 and Room2) at 7am on the 1st day of each month. The nodes in each room are considered equivalent, i.e. they run the same application and are equipped with the same set of sensors. The specification describing this example scenario is:

```

//equivalence - 'no equivalence' or 'region equivalence'
region equivalence

//Declare the basic elements of the language
Time T1;
Region R1, R2;
Event EFire;

//Define the elements
T1=07:00:00, */1/2012;

R1={Room1};
R2={Room2};
EFire = Fire @ T1;

```

A challenge in testing is to maintain a high degree of coverage while reducing the size of the test suite and thus shortening the testing stage. There are additional reasons, specific to WSN applications, for keeping the test suites small. Extensively testing the network would not only increase the cost of the project but will also significantly reduce the lifetime of the system. In addition, memory constraints prevent us from storing too many test inputs on the nodes. Therefore, it is essential to develop methods that would help decrease the number of tests run by the RTA mechanism.

The number of tests needed to fully test the MEDAL model of an application is m^n , where n is the number of inputs and m is the size of the sensing value range of the sensors. Since there are multiple sensor nodes in the network, the total number of tests would be m^{s*n} where s is the number of sensors. In an example network with 100 sensor nodes, each of which has a temperature sensor with value range 0-100 C and a humidity sensor with value range 0-100 RH, the number of tests for all possible input combinations would be 100^{2*100} , or 10^{400} . Running that many tests on a sensor network is infeasible. To address this problem we utilize three test reduction techniques. The first technique has been widely used for reducing the number of tests for software applications. The second and third ones, however, are novel and also unique to the nature of WSN applications.

Static model analysis

The first test reduction technique performs a static analysis of the MEDAL model. The transitions that fire based on the value of the tokens shape the input-to-output behavior of the model. Therefore, we consider

these transitions to be the *important* transitions in this analysis. To test such a transition, it is enough to identify the “passing” and “non-passing” sets of inputs and then provide values to represent these two sets. For example, to test a transition with a firing rule “the value should be greater than 27”, we need two values - one less or equal to 27, and another one greater than 27. Our static analysis identifies the important transitions and their passing and non-passing sets. Then it chooses a random value to represent each of these sets. This reduction decreases the number of input values to 2^t , where t is the number of important transitions. Therefore, the number of tests for the whole network is decreased to 2^{s*t} , which is significantly smaller than m^{s*n} since the value range of a sensor is typically much larger than 2 and t is comparable with n .

A limitation of this step is that it is tightly coupled with MEDAL. The reduction step could work with another modeling language only if this language is able to precisely identify the application logic and the important parts of the application that define the input-to-output behavior of the application.

Network topology

A unique characteristic of WSNs application is that, unlike other software applications, they are strongly dependent on the topology of the network. For example, two sensors are more likely to influence each other’s decisions if they are neighbors. We take advantage of this feature to further decrease the size of the test suite. We divide the network into *regions*. The size and shape of these regions is very flexible. The default size we use is a room in a building, but a region could also be a group of nodes located close to each other. The RTA test specification contains information about which regions should be tested. When creating the set of tests for a region we only include the nodes within that region. An example application to justify this choice would be detecting an event E in a building with many rooms. A node determines if E has occurred based on its own readings and the readings of its neighbors, where a neighbor is a node in the same room. In this case it is sufficient to test the rooms separately since nodes do not consider the readings of sensors outside of their own room. This approach reduces the number of tests to $R * (2^{sR*t})$, where R is the number of regions in the network and sR is the number of sensors in a region. Since sR is much smaller than the number of nodes in the network, this reduction substantially decreases the number of tests.

Redundancy

Another unique characteristic of WSNs is node redundancy. The assumption that all nodes in a region are redundant helps us reduce the number of tests that are sufficient to test the network even further. If we assume that all nodes in a region are equivalent to each other, we can use the same test inputs to check the behavior of each one of them. Using this assumption we can decrease the number of tests from $R * (2^{sR*t})$ to

$R * (2^t)$. For example, if a network is deployed in a building with 8 rooms ($R = 8$) and there are 5 nodes in each room ($sR = 5$) with 5 important transitions in their application logic ($t = 5$), applying this reduction step decreases the number of tests from 2^{28} to 2^8 .

Often, if the application logic is more complex, the number of tests could remain large even after applying all three reduction techniques. In these cases, designers could choose a test subset that has an acceptable level of coverage. Alternatively, a test schedule could be devised such that tests are chosen on a rotation principle and only a few tests are run each time.

6.2.3 Automated Code Generation

Software integrated development environments (IDE) based on various abstract modeling languages, such as UML and state machines, are widely used for software development in diverse applications [142]. This approach has the potential to reduce or even eliminate the code development phase by providing code generation capabilities. Software development that relies on model-based development and automated code generation from design models could improve both the quality of the resulting system and the efficiency of the development process.

One of the benefits of MEDAL is that it is a formal unambiguous specification language, which renders it suitable for automated code generation. The MEDAL model of an application can be used to automatically generate the code to be executed by the nodes in the network.

Model clarity is one of the key requirements for correct code generations. This is another place where the extensions introduced in Chapter 3 prove to be helpful. If we again consider the explosion application from Chapter 3, there are four types of nodes in the explosion detection system: temperature node, light node, acoustic node, and a cluster-head node. This means that the code generator has to produce four different executables - one for each of the types of nodes. Therefore, the MEDAL model of the application should provide clear indication of the boundaries between the logic for the different nodes.

Figure 6.2 shows how the radio transition in the explosion detection application can be used to determine the boundaries between the different executables. When we perform the cut through the radio transition, the result is four smaller models corresponding to the application logic executed by each of the four types of nodes. Since the radio transition is an abstract element, the code generation mechanism can treat the reoccurring transition *R1* as either an instruction to send messages, when the transition has an input arc, or an instruction to receive messages, when the transition has an output arc. Once the different sub-models corresponding to each of the node types are identified, the code generation mechanism can produce the actual

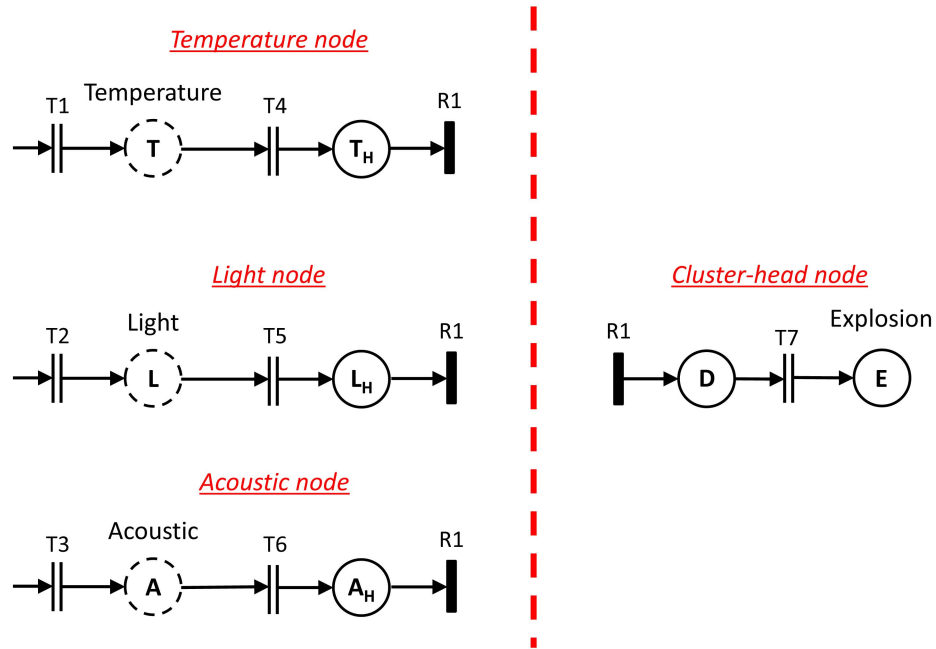


Figure 6.2: The radio transition in a MEDAL model can be used by an automated code generator to determine the boundaries between the different node executables that should be generated.

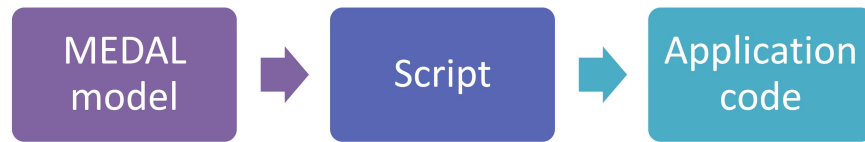


Figure 6.3: The steps of the code generation process for MEDAL. The MEDAL model is translated into a script, which is then used by the code generator to produce the code for the nodes in the system.

code to be run on the network. Similarly, inhibitor arcs help with priority specification and with identifying preconditions for events.

Yafeng Wu developed an automated code generator, which produces TinyOS code based on an application's model [143]. Currently, the input to the code generator is not the MEDAL model itself, but a script that describes the model. This script is used to produce the code for the nodes in the network. Figure 6.3 shows the process of code transformation from a MEDAL model to TinyOS code. The rest of the section describes in more detail the script and the structure of the automatically generated code.

6.2.4 Translating MEDAL into a script

The input to the automated code generator is a script file that designers write to describe the MEDAL model of an application. An example script for the model of the explosion detection application in Figure 3.3 is

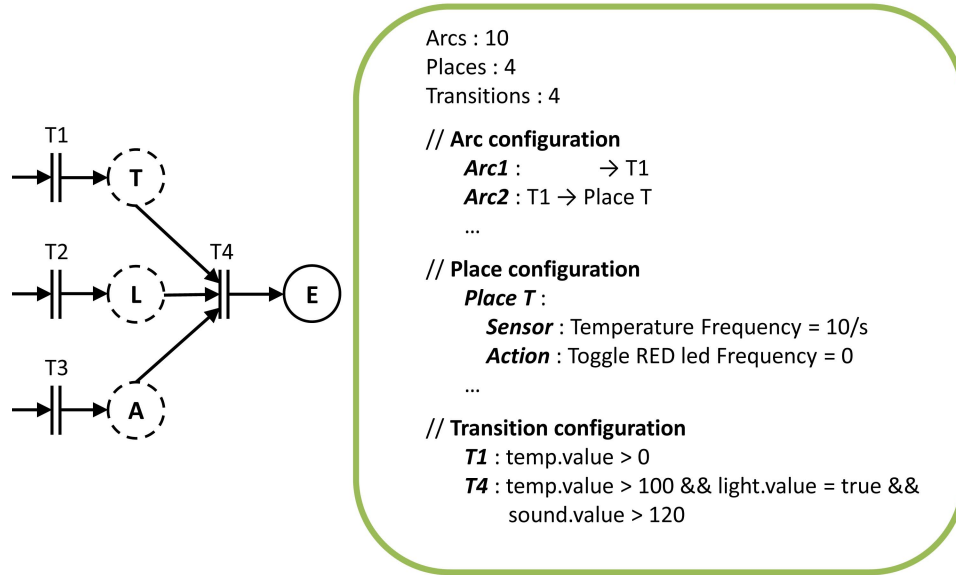


Figure 6.4: A script specifying the MEDAL model of an application describes all arcs, places, and transitions with their types and attributes.

shown in Figure 6.4. The script specifies the arcs, places, and transitions in the model. Places have two attributes: sensors and actions. At the sensor event places, which are places *T*, *L*, and *A*, nodes need to access sensors to get data. The designers have to specify the sensor type and the sampling frequency for these places. There could be multiple operations, called *actions*, that a node has to perform once a particular place has been reached. Although a place can represent a multiple number of actions, it can be associated with only one type of sensor.

6.2.5 Code structure

TinyOS programs are built out of software components. In order to translate the MEDAL model into TinyOS code, we add a MEDAL structure above the TinyOS component model. This structure consists of components that represent places and transitions.

The core of the code structure is the concept of a *token*. A token is defined as an encapsulation of a value plus temporal and spatial information. The code executions is driven, processed, represented, and recorded as tokens passing through places and transitions. Tokens pass through components to transfer information and drive the execution. By recording the traversing history of tokens, we can continuously monitor and collect traces of system execution.

Although the automated code generator currently generates only TinyOS code, it can be adapted to generate code in other languages. For example, instead of components, we can build structures or classes

for the different MEDAL objects, and update the code generator's templates to generate code in the new languages.

6.2.6 Test Execution Support

The test execution support was developed Jingyuan Li. It is responsible for automatically running the RTA tests on the networked sensing devices. Supporting the execution of RTA tests involves the following steps:

1. After system initialization, every node goes through a synchronization process after which it is assigned its current time and location.
2. RTA tests are scheduled or requested via test specifications at the user terminal. RTA test data is generated by the automated test generation mechanism, delivered to the WSN gateway, and then shipped to each node using the test deployment protocol. Each test is scheduled to automatically start and stop at specified times using the test control protocol.
3. When the scheduled test start time is reached, each sensor node involved in testing enters testing mode and starts reading the test inputs that have been provided to it. When test mode is entered, each sensor node stops sampling from the corresponding real sensors to avoid communication conflicts. However, to avoid the interruption of real event monitoring, users can specify that sensor nodes should continue sampling even in test mode.
4. When virtual events are detected by the application, the results are reported to the gateway and then used for test result resolution.

The test execution support completely automates the RTA test process. After the users provide the test specification, the specification is processed by the test generation mechanism which generates the test data. This test data is then delivered to the sensor nodes in the network via a WSN gateway. When it is time to run the scheduled tests, the sensor nodes start reading from their virtual sensors to simulate virtual events. These events are processed by the application logic, and the application layer outcomes are reported back via the WSN gateway.

6.3 Case Study

We present a case study to demonstrate the usability of our RTA methodology. In this scenario, we are required to design and build a fire detection system for a building. The building has seven floors and there are ten rooms per floor. There is at least one node with a smoke sensor and one node with a temperature sensor in each room. The fire detection application uses both smoke and temperature readings to determine the presence of fire. The fire detection algorithm we use is composed of two separate algorithms. The first

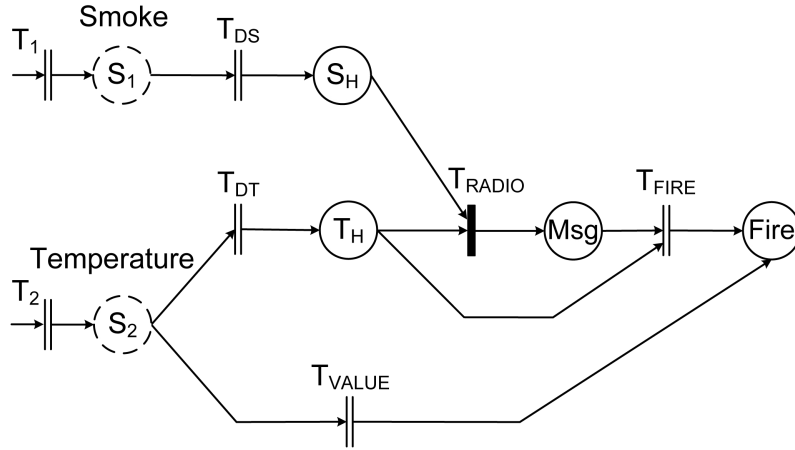


Figure 6.5: A fire detection application uses smoke and temperature readings to determine the presence of fire. The MEDAL model of this application logic concisely and unambiguously depicts the application logic.

monitors the temperature and smoke increase rates and if they both exceed some predefined thresholds, the algorithm reports the presence of fire. The second algorithm reports fire if the temperature value exceeds a predefined threshold. Figure 6.5 shows the MEDAL model describing the logic of the application. The elements of this model are:

- Places S_1 and S_2 represent the two types of sensor events, while S_H , T_H , Msg and $Fire$ stand for, High smoke increase, High temperature increase, Message reception, and Fire, respectively.

- Transitions: Transition T_{DS} is fired if the smoke increase rate goes above a specific predefined value. Firing transition T_{DS} sends the firing value over the radio and raises a local smoke alarm. Transition T_{DT} fires if the temperature increase ratio goes above another predefined threshold. This also causes the application to send a message over the radio and raise a local temperature alarm. Transition T_{FIRE} is fired if fire has been detected. It is enabled only if the local temperature node has detected high temperature increase and at least two messages have been received from neighboring nodes in the same room indicating high temperature and smoke values.

Next, we implement the system. First, we write a new script file according to the application model, and input it into the automated code generator. The automated code generator partitions the whole logic and generates the code for the smoke and temperature sensors.

To automatically generate the test suite, we need the user to provide us with an RTA test specification. The testing specification in Section 6.2.2 could be used for this scenario. Knowing the test specification, the application model, and the topology, we can generate the necessary set of tests to be run by the execution support. These tests need to provide such inputs to places S_1 and S_2 that the presence of fire is simulated. When these tests are run, the application is expected to report the presence of fire based on the virtual

readings from the tests.

At first this case study might seem simplistic. However, several key points must be emphasized. It is important to recognize that this simple model can represent a fire detection system that exists across many floors and uses many sensors. Although we had initially specified that we have seven floors with ten rooms each, these numbers do not confine the application model. The same model could successfully be used to detect fires in a skyscraper with hundreds of rooms. In other words, even though the model is simple, it can represent a large scale system. This case study also uses relatively simple logic for temperature and smoke sensors. However, the power of the underlying Petri net allows us to describe arbitrarily complex logic and control flow. An additional advantage is that the formal model of a complex application is not necessarily too big or complicated since much of the complexity for such systems is encompassed in the logic associated with the transitions.

6.4 Evaluation

To investigate the performance of our RTA methodology, we have implemented a prototype Fire Detection (FD) system that is based on the application model presented in the case study. The system is built on an indoor testbed. The testbed is composed of 21 TelosB nodes, placed in a 7×3 grid on a board. One node is chosen as the base station and the rest are divided into different rooms. The TelosB nodes are equipped only with a light sensor. We have used the light sensors to simulate temperature sensors. We use the fire detection algorithm described in Section 6.3 and the nodes only contain the logic for the temperature sensors. Once a node detects a fire, it sends a report to the base station. For the communication topology we constrain the nodes to only directly communicate with nodes in the same room. For multi-hop communication we use a simple geographic forwarding routing protocol.

6.4.1 Test reduction

We have analytically estimated the number of tests necessary to fully test the MEDAL model for both the prototype FD application and the FD application presented as a case study. Our calculations are shown in Table 6.1 and Table 6.2, respectively. For both tables, the result of applying a particular reduction step is calculated as the number of automatically generated tests when the current reduction and all previous reductions steps have been applied together.

The results in both tables show that the first reduction step, statically analyzing the MEDAL model, provides the highest decrease in the number of tests. However, even after using this reduction, running a hundred billion tests on a sensor network is still unreasonable, if not impossible, considering the limited

rooms	2	4	5	10
nodes in room	10	5	4	2
baseline (no reduction)	$11 * 10^{35}$	$11 * 10^{35}$	$11 * 10^{35}$	$11 * 10^{35}$
static analysis reduction	$11 * 10^{11}$	$11 * 10^{11}$	$11 * 10^{11}$	$11 * 10^{11}$
topology reduction	$2 * 10^7$	4096	1280	160
redundancy reduction	8	16	20	40

Table 6.1: Combining static analysis techniques and knowledge about the network topology and node redundancy reduces the size of the test suite for the FD application by 35 orders of magnitude.

rooms	2	4	5	10
nodes in room	10	5	4	2
baseline (no reduction)	$11 * 10^{91}$	$11 * 10^{91}$	$11 * 10^{91}$	$11 * 10^{91}$
static analysis reduction	$11 * 10^{17}$	$11 * 10^{17}$	$11 * 10^{17}$	$11 * 10^{17}$
topology reduction	$2 * 10^9$	131072	20480	640
redundancy reduction	16	32	40	80

Table 6.2: When all three reduction steps are applied, adding an extra sensor only doubles the number of necessary tests instead of causing an exponential increase.

resources of the sensor nodes. The situation is further exacerbated by the fact that the tests must be run periodically.

Applying our second reduction step, which takes advantage of the topology, additionally decreases the number of tests. The impact of this step is much more noticeable in cases where there are just a few nodes per region. As shown in Table 6.1, in the case where we have 10 sensor nodes per region, the drop in the number of tests is just 10^5 times, while the same step leads to a 10^{10} times reduction in the scenario with 2 nodes per region. Similar results can be seen in Table 6.2.

The decrease in the number of tests after applying the last reduction step is significant as well. Compared to the previous reduction step, here we see the opposite effect: the more sensors there are per region, the higher improvement we get. This is due to the fact that since the nodes in a region are considered to be equivalent, all of them can use the same input test values.

A number of conclusions can be drawn based on the results in Table 6.1 and Table 6.2. First, if no reduction is applied, the number of tests we have to run increases exponentially with the number of sensors needed by the application. However, if all three reduction steps are applied, adding an extra sensor merely doubles the number of necessary tests. Second, it is better to create fewer regions with more nodes than more regions each containing a small number of nodes. Third, all three reduction steps need to be applied in order to generate a small set of tests. We realize that there might be cases where the sensor nodes are not equivalent and the last reduction step cannot be performed. In such situations, to avoid exhausting the network's resources, it might be necessary to either only run a carefully chosen subset of the generated tests or schedule the tests in such a way that all generated tests are run but over a more extended period of time.

The automated test generation mechanism is not designed to handle node mobility. Therefore, changes in the topology might cause the RTA tests to fail even if the system is still able to function correctly. This problem can be alleviated in networks with low node mobility levels where we can rely on the test execution support to automatically detect topology changes and regenerate and redistribute tests when such changes occur. However, this approach is not feasible when the level of node mobility is high.

6.4.2 Robustness to Failure

The goal of RTA is to maintain the accurate operation of a WSN application despite the unreliable and failure-prone underlying infrastructure. To evaluate the robustness of our RTA approach, we have compared it to a health monitoring (HM) system and a pure system with no RTA or health monitoring support. All three fire detection systems have the same application logic. The RTA FD system is generated with the help of our RTA framework. The rooms are tested on a rotation basis and a different room is picked each time a test should be run. The FD system with HM monitoring is implemented following the HM mechanism introduced by Memento [78]. Memento uses heartbeats from neighbor nodes to determine if a node is functional. We assume that when an RTA test fails or the HM system detects a node failure the failed nodes are immediately repaired.

Experimental Results

To demonstrate RTA's efficiency in maintaining application robustness while significantly reducing maintenance cost, we have run multiple experiments under different settings. We use a flashlight to generate the fire events in our testbed. To measure the system's performance under realistic scenarios, the fire event sequences are generated using a Poisson process generator. The room in which a fire occurs is chosen randomly. A random Poisson process is also used to compute the failure sequence and determine which nodes stop executing the application and at what time. During the experiments, the WSN gateway injects node-failures by sending messages to the selected nodes and stopping their execution strictly according to the failure sequence. For each experiment, we run the three systems sequentially with the same fire and failure sequences. A node is periodically chosen to fail until no operational nodes are left and the experiment stops. The robustness of the system is represented by the false negative rate which we define as the ratio between the number of unreported fires and the total number of generated fires. Maintenance cost is measured with the number of repairs, the Mean Time to Repairs (MTTR), i.e. the average time between two consecutive repairs, and the number of messages sent during testing.

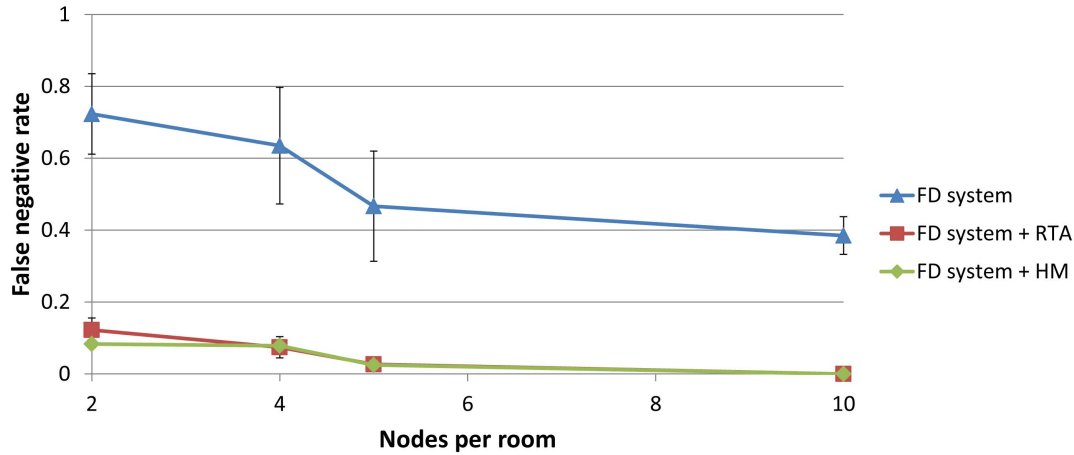


Figure 6.6: RTA and HM achieve similar false negative rates.

In the first set of experiments we measure the robustness and maintenance costs of the three systems. To study the impact of redundancy, we vary the number of nodes per room from 2 to 10. All experiments last 20 minutes and both the RTA and the HM system run a test every minute. The fire event rate is 0.5/s, and the node failure rate is 1/s. Figure 6.6 shows the results of these experiments. Each data point is the average of 5 trials with a 90% confidence interval. The results reveal that, compared to a pure system, both RTA and HM significantly reduce the false negative rates of the FD application. This is expected as both mechanisms can detect node failures and repair nodes immediately. Comparing the RTA system to the HM system shows that both systems demonstrate similar robustness levels. The only visible difference is when there are only 2 nodes per room. Since according to the application logic we need at least two nodes per room, any node failure will lead to a false negative. In this case, the false negative rate of the RTA system is slightly greater than that of the HM system because the RTA system only tests one room at a time while the HM system tests all nodes every time. Increasing the level of redundancy eliminates this difference in the false negative rates. With 5 nodes per room, both systems have a false negative rate close to zero.

We also compare the maintenance cost of the RTA and HM systems. Figure 6.7 shows the number of repairs for both systems. We can see that the RTA system requires fewer repairs than the HM system. The number of repairs required by the HM system is constant since it triggers a repair every time a node fails. On the other hand, the RTA system repairs nodes only when the operational nodes in a room cannot detect the fire. Our results show that with a certain level of redundancy, the RTA mechanism can significantly reduce the number of repairs while still providing high confidence. For example, with 10 nodes per room, the RTA system guarantees no false negatives and requires only a total of 2.5 repairs. On average, the RTA system produces 70% fewer maintenance dispatches. We also measure the MTTR of both systems. Results show

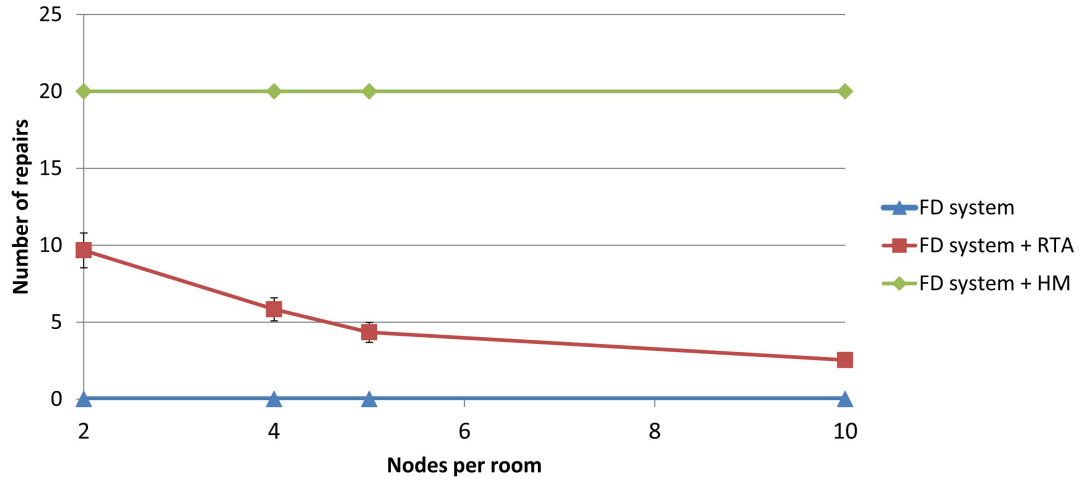


Figure 6.7: RTA requires 50%-70% fewer repairs than HM.

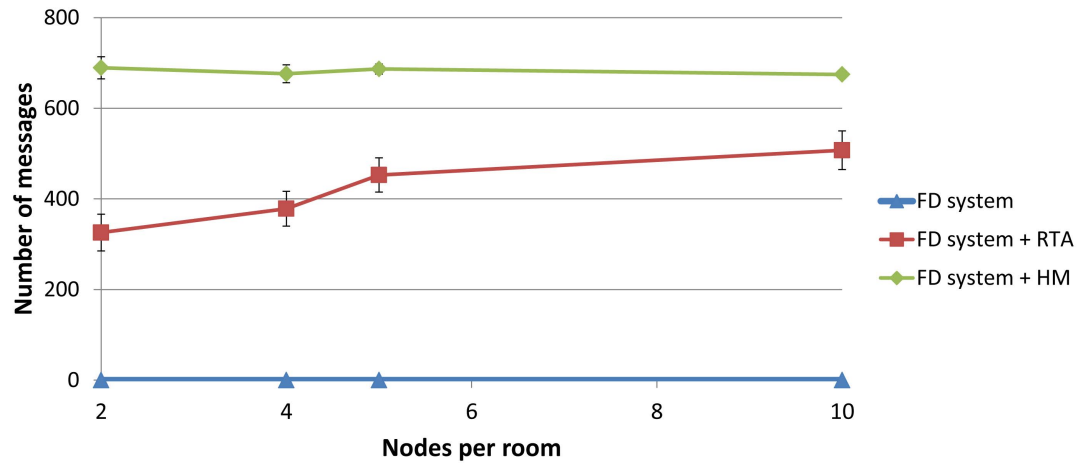


Figure 6.8: On average, RTA uses 33% less messages than HM.

that the RTA system has much longer MTTR than the HM system. For example, with 5 nodes per room, the MTTR of the RTA system is 5.6 seconds, with 1.5 seconds for the HM system.

Another type of maintenance cost is the extra messages incurred by running tests. The base station in the RTA system needs to send messages to trigger tests and nodes are required to report virtual fire events during tests. For the HM system, nodes need to send heartbeats, and the base station should be notified if no heartbeat is received from some node. These extra messages use bandwidth, consume additional energy, and reduce the system lifetime. We have measured the number of the extra messages for both systems and the results are shown in Figure 6.8. We see that the RTA system introduces much less message overhead. The HM system always uses over 650 messages, because nodes keep sending heartbeats. On average, the RTA system uses 33% less messages than the HM system.

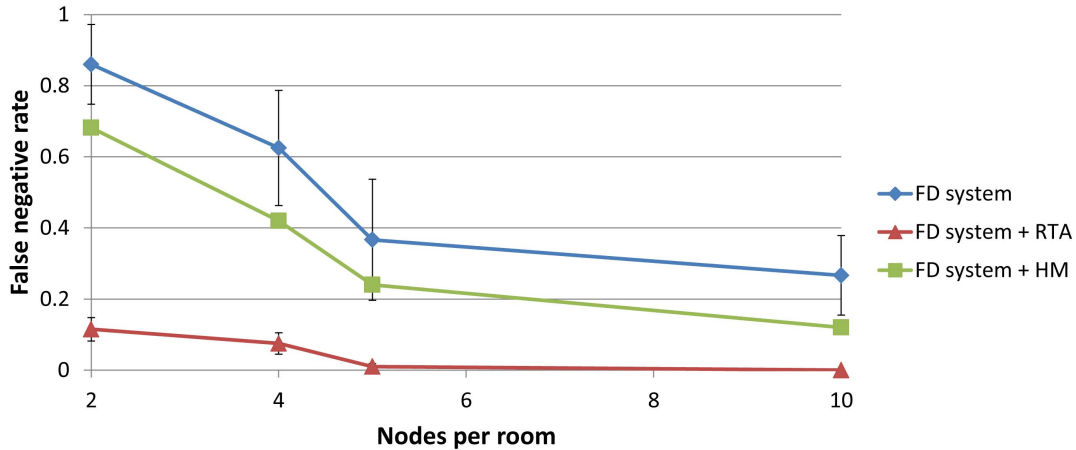


Figure 6.9: When the failures introduced in the system are location errors, the RTA system missed 75% fewer fires than the HM system on average.

Node level failures are lower level failures and the HM mechanism has been specifically designed to detect them. In the next experiment we compare the performance of the two systems when application level failures are introduced. To do this we insert a new type of error, which we call *location error*. A location error can be defined as any unexpected changes in the location of a node from one region/room to another. Such a location change should lead to a failure if it is against the application's requirements. In our scenario, we need to have at least two nodes per room. Therefore, a location change that decreases the number of nodes in a room to less than two should cause an application-level failure.

In this experiment, we change the location of nodes to simulate location errors. We still use a Poisson sequence to generate location errors with the rate of 1/s, and randomly chosen nodes change their locations to adjacent rooms. The false negative rates of our three FD systems are shown in Figure 6.9. We see that the HM system has a high false negative rate and cannot guarantee the system's robustness. This occurs because when a node is moved to an adjacent room, it can still send heartbeats to its neighbors, so the HM system considers it operational. However, since the RTA system tests the fire detection ability of each room, it can catch such errors without using extra mechanisms to explicitly detect location errors. On average, the RTA system misses 75% fewer system failures. When the node redundancy reaches 5 nodes per room, RTA does not incur any false negatives. Based on these results we can conclude that, compared to an HM system, an RTA system can provide better visibility into the application behavior at run time.

Simulation Results

We have used simulation to demonstrate that RTA can significantly reduce the amount of network maintenance. We simulate a scenario in which we have 25 rooms, and each room has N nodes, where N represents the

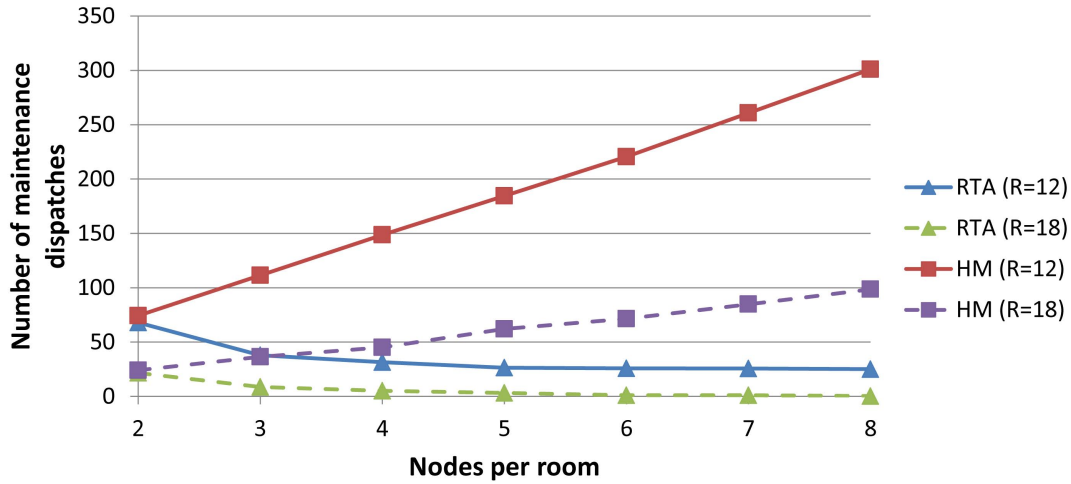


Figure 6.10: RTA reduces the number of maintenance dispatches to only 0.3%-33.9% of HM.

level of node redundancy in that room. We assume that the lifetime of the sensor nodes follows a Poisson distribution. R is the mean lifetime of the sensor nodes in months and it represents the reliability of the nodes. We generate fire events following a Poisson distribution. The expected occurrence interval of these events is three months, and the simulation period is set to two years. Each data point represents the average of five runs.

We make the following two assumptions when comparing the amount of maintenance work required by RTA and HM: 1) If the RTA tests determine that the nodes in a room are not able to detect the presence of fire, a maintenance worker is dispatched to replace the failed nodes in that room; 2) If the HM determines that a node has failed, a maintenance worker is dispatched to replace the failed node.

In order to achieve a fair comparison, we use the same testing overhead (in terms of the number of testing messages sent) for RTA and HM. On average, an RTA test requires K times more messages than HM - in our implementation $K \approx 8$. Therefore, in the simulations RTA runs in $1/8$ the frequency of HM. In this simulation the RTA test frequency is a test per day for each room.

Figure 6.10 shows the number of maintenance dispatches under different levels of node reliability R and redundancy (nodes per room). We observe that when the redundancy is increased RTA makes a significant reduction in the number of maintenance dispatches. With 6 or more nodes per room, the number of RTA maintenance dispatches is less than 10% of the dispatches required by HM. Since HM maintains node-level reliability, the number of maintenance dispatches is proportional to the number of node failures. In contrast, RTA maintains application-level reliability and a maintenance dispatch is only necessary when the application is not able to meet its high-level requirements. Thus, by taking advantage of the node redundancy level, RTA can considerably reduce the total number of the required maintenance dispatches.

System	ROM (bytes)	RAM (bytes)	lines of code
Pure Fire detection system	18142	898	1767
Fire detection system with HM	22976	1280	2590
Fire detection system with RTA	24888	3386	3193

Table 6.3: Compared to HM, RTA has a larger memory footprint.

6.4.3 Overhead

Table 6.3 compares the memory usage and footprint of the three FD systems. We can see that, because of its token-flow programming structure and more powerful test execution facilities, the RTA system uses more program memory and RAM. Also, the code for the RTA system is larger than that for the other two systems. However, since most of this code is automatically generated by the RTA framework, users only need to write the MEDAL model specification script, which was less than 50 lines for the experimental FD system.

6.5 Summary

A major disadvantage of the current reliability and health monitoring techniques used for WSN applications is that they monitor the performance of the low-level components of the system instead of the reliability of the application layer. To the best of our knowledge, this is the first work to address this issue in WSNs and suggest a methodology to help designers and users verify an application's integrity at run time. We have implemented a framework that facilitates the use of our RTA methodology. This framework provides automated code generation, automated test generation, and execution support for the RTA tests. To decrease the vast number of generated test cases, we introduce three test suite reduction techniques, two of which are unique to the nature of WSN applications and take advantage of the network's topology and node redundancy.

We evaluate our implementation on a network of 21 TelosB nodes, and compare performance with an existing network health monitoring solution. Our results indicate that RTA performs much better than health monitoring in identifying application-level failures and almost just as good in identifying low-level failures. In addition to providing application-level verification, RTA misses 75% fewer system failures, produces 70% fewer maintenance dispatches, and incurs 33% less communication overhead than network health monitoring.

Chapter 7

Simultaneous Multi-classifier Activity Recognition Technique (SMART)

ONE of the main appeals of using WSN deployments for smart home applications is the low cost of the devices and the possibility for easy do-it-yourself installation. However, recent studies have found that low-cost sensors suffer from many types of faults [144]. Inexpensive nodes can break and battery-powered nodes lose power. Furthermore, the do-it-yourself installation process leads to a large number of *non-fail-stop* faults in which the sensor does not completely fail. Instead, it continues to report values, but the meaning of these values changes or becomes invalid. For example, sensors were reported to become dislodged over time, or to fall off surfaces only to be re-mounted by the homeowner in the wrong location. Sensors were covered by objects or blocked by an open door or re-arranged furniture. Some sensors mounted on furniture were moved into different rooms. As the number of sensors increases, the number of failed nodes also increases. The authors reported that homes with hundreds of sensors had at least one sensor failure per day on average. The maintenance cost of fixing all such failures is prohibitive, and may negate any cost advantages of inexpensive hardware and installation.

Several techniques have been developed to detect and report fail-stop hardware failures by repeatedly querying the nodes or checking for lost data [78, 80], but these techniques cannot detect non-fail-stop failures because the hardware is still responding and reporting values. Recently, several approaches have been developed to detect non-fail-stop sensor faults by exploiting correlations between neighboring sensors [83, 87]: two data streams that contain shared information about the physical world are typically correlated; if that correlation changes then one of them might be experiencing a non-fail-stop failure. We call this a *bottom up* approach because it uses patterns in the raw sensor data, without knowledge of what the sensor data means.

However, these techniques are designed for homogeneous, periodic, and continuous-valued sensors; it is not straightforward to generalize these approaches to the heterogeneous, binary, and event-triggered sensor suites often used in smart home applications.

In this chapter, we describe a Simultaneous Multi - classifier Activity Recognition Technique (SMART) which uses *top down* application-level semantics to detect, assess, and adapt to sensor failures. SMART **detects** non-fail-stop node failures, which include failures caused by nodes getting stuck at a value, node displacement, or node relocation. The failure detection is performed at run time by using multiple classifier instances that are trained to recognize the same set of activities based on different subsets of sensors. If a sensor fails, it will only affect a subset of the classifiers and the *ratio* of activity detection among the classifiers will change, thereby indicating a possible sensor failure. Once a sensor failure is detected, SMART **adapts** to the failure by excluding the failed node and creating a new classifier ensemble based on the remaining subset of nodes. SMART uses data replay analysis to **assess** whether the failure would have affected activity recognition in the past had the new classifier ensemble been used. If it would have, SMART dispatches a maintenance person to repair the failure. Otherwise no maintenance is necessary. Although we currently focus on activity detection applications, our technique can be generalized to other smart home applications that use similar types of sensors.

We demonstrate the concepts behind SMART in the context of activity recognition, where a classifier is used to identify an activity being performed based on the sensor data generated in the home. The basic insight behind SMART's failure detection is similar to that used by prior work: sensors that monitor the same activities in the physical world typically have correlated values. Instead of relying on blind mathematical correlations among the raw sensor values, however, SMART uses classifiers to identify the degree to which different sensors indicate the same ongoing activity in the home. The power of this technique lies in the ability to use labeled training data sets, application-level feature extraction, or other application-level tools to identify meaningful correlations between sensors for the purpose of sensor failure detection.

We evaluate SMART using publicly available datasets from three homes [145, 146]. We artificially introduce failures in the data, to analyze SMART's ability to detect non-fail-stop node failures and adapt to those failures. The main contributions of this work are:

1. A new approach to detect non-fail-stop failures. Our results indicate that we can identify application-level failures with more than 85% accuracy;
2. A novel failure severity assessment technique which decreases maintenance costs. Our evaluation shows that we reduce the number of maintenance dispatches by 55% on average and increase the mean time to failure (MTTF) of the application 3.2 times;

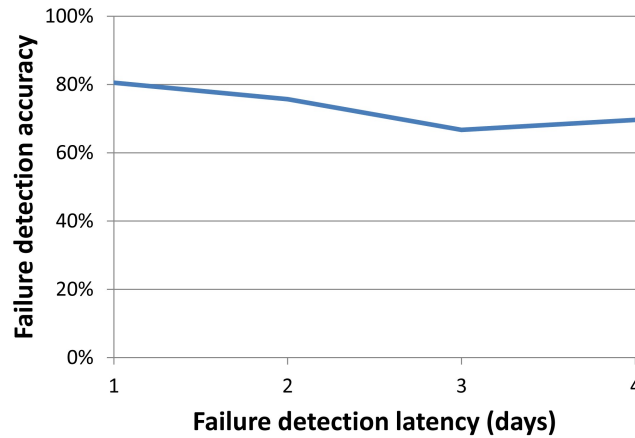


Figure 7.1: Failure detection accuracy for a “movement” non-fail-stop node failure introduced in a smart-home deployment. Correlation-based approaches cannot achieve more than 80% failure detection accuracy even 4 days after the failure has occurred.

3. A multi-classifier approach, which allows the application to use the most suitable classifier at any given time and adapt to node failures by retraining additional classifiers that reflect the current state of the system. Our experiments show that SMART improves the activity recognition accuracy under node failures by more than 15%.

7.1 State of the art

Recently, correlation-based techniques have been developed to identify non-fail-stop node failures in environment-monitoring applications [83]. To understand how accurately correlation-based techniques can detect non-fail-stop failures in event-driven applications, we implemented a correlation-based prototype system. The system is based on Generiwal’s correlation-based framework [83]. However, we had to make a number of adaptations to address the event driven-nature of activity recognition applications:

- The majority of sensors in activity-recognition applications are binary and value-based correlation is not suitable. Therefore, we use temporal correlation, where sensors have higher correlation if they fire within some time of each other;
- Event-driven sensors are idle most of the time and using temporal correlation could result in insignificant correlation values. In order to achieve meaningful correlation values, our correlation technique ignores the periods where no nodes fire.

Figure 7.1 shows the results of our correlation based experiments for detecting a simulated non-fail-stop *movement* failure, where one of the kitchen motion sensor is accidentally moved to point in a different

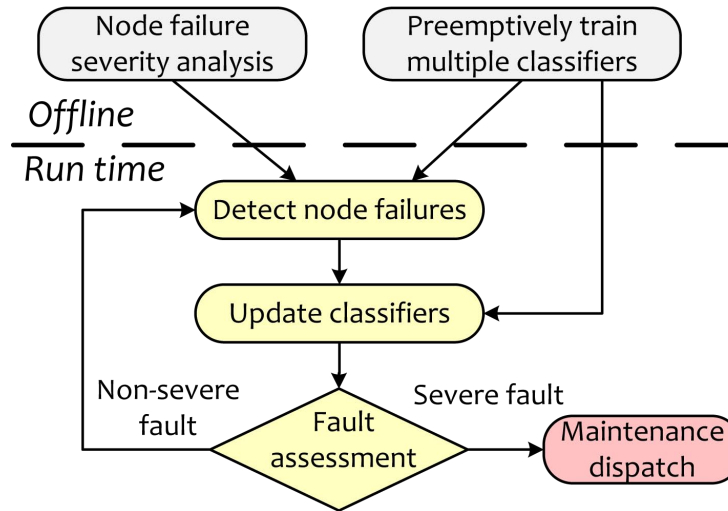


Figure 7.2: SMART has two components: offline and run time. Preliminary offline analysis of node failure severity is performed and multiple classifiers are trained. The results of this analysis together with the classifiers are used at run time to detect node failures and determine if maintenance is needed.

direction. We perform this experiment on a 43-day-long dataset from a two-resident home [145]. The x-axis shows the failure detection latency, which is the number of consecutive days used to collect data after the failure has occurred. To evaluate the failure detection accuracy with a latency of x days, we construct a set of all possible combinations of x consecutive days from the dataset. For each entry in this set, we calculate the correlation values among the sensors for both *no-failure* and *movement-failure* cases. The evaluation of how accurately we can distinguish between the *no-failure* and *movement-failure* cases is performed using a 10-fold cross validation on the J48 decision tree algorithm from the WEKA software package [135].

Our evaluation shows that the correlation-based approach cannot achieve failure detection accuracy higher than 80%. Also, the accuracy decreases as the number of consecutive testing days increases. This shows that bottom-up temporal correlation is not a good approach to detect non-fail-stop sensor failures. The main reason is that temporal correlation between nodes looks at when two nodes fire together irrespective of which activity is being performed. However, sensors are often correlated in a different way based on the activity that is performed. Therefore, a top-down approach, like the one described in this chapter, which evaluates the relations between different sensors per activity, would be able to achieve higher failure detection accuracy.

7.2 Approach

SMART can be divided in two components: offline and run time (Figure 7.2). In the offline stage, our approach trains classifier instances for all possible combinations of node failures by holding those nodes out of the training set. Since classifier training is done offline, the time to train the classifiers does not affect

the system at run time. This stage also performs preliminary failure severity analysis of the effect of sensor failures on the classifiers' performance. This analysis can be rerun in the future if the system behavior evolves over a longer period of time.

At run time SMART chooses an ensemble of preemptively trained classifiers based on the failures that have already been detected. When a new node failure is detected, the system adapts to that failure by updating the classifier ensemble to contain classifiers that are trained for the failure. Since all classifier instances are trained beforehand, the overhead of updating the classifier ensemble at run time is negligible. If the results from the failure severity analysis indicate that the new classifier ensemble can maintain the detection accuracy of the application above the application specified *severity threshold* TH_S , the failure is considered non-severe. We define TH_S as a fraction of the original accuracy of the system when no failures are present. If a failure is not severe, no maintenance is necessary since the application can still meet its high-level requirements. However, if the node failure is severe and the detection accuracy falls below TH_S , one or more sensors need to be repaired. In the rest of this section we describe in more detail: 1) the training and use of the classifier ensemble, 2) the detection of non-fail-stop failures, 3) the node failure severity analysis, which allows us to decrease the number of maintenance dispatches, and 4) how SMART maintains high detection accuracy under failures.

7.2.1 Using multiple simultaneous classifiers

We use the following notation throughout the chapter:

1. $C(Training\ set, Testing\ set)$, where, for a classifier instance C , *Training set* and *Testing set* are the sets of sensors used for training and testing, respectively.
2. $C_{Accuracy}$: accuracy of classifier instance C .
3. C_{NDA} : number of detected activities by classifier instance C .
4. S : the set of all nodes in the system.

Complete node failure analysis requires every possible combination of faulty sensors to be considered. However, the analysis of all $2^{|S|} - 1$ failure sequence traces does not scale to large networks with hundreds of sensors. Based on preliminary analysis, we found that it is enough to initially consider only the $|S|$ single-node failures. Therefore, in addition to the classifier instance $C(S, S)$ trained on all nodes, we also analyze $|S|$ more classifier instances, where $\forall s \in S$ a classifier instance is preemptively trained for the failure of node s , i.e., its training set contains $S - s$ sensors. Figure 7.3 shows how the training sets for the additional $|S|$ classifier instances are identified. The classifier ensemble, containing $|S| + 1$ classifiers, is used to:

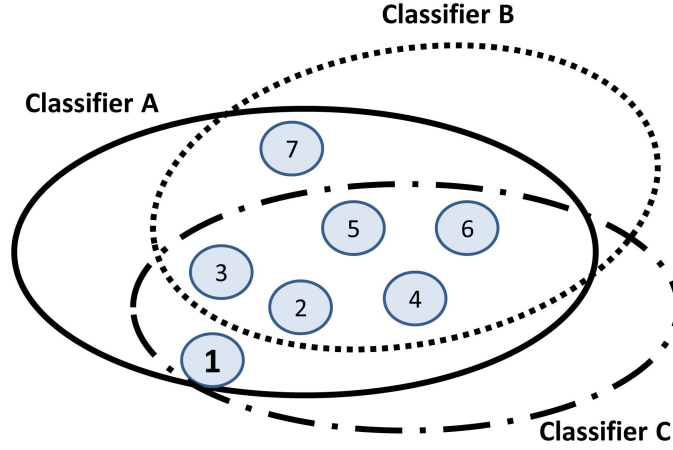


Figure 7.3: Preemptively training classifiers for the occurrence of failures by *holding one sensor out* from the training set allows us to build failure-aware systems. Classifier A is trained on all nodes. Classifiers B and C are trained by holding node 1 and node 7, respectively.

1. Detect the occurrence of failures and identify which nodes have failed by monitoring the relative behavior of the original classifier instance and that of the other $|S|$ instances.
2. Maintain high detection accuracy in the presence of failures.

We denote the classifiers as classifiers $C_0, C_1, \dots, C_{|S|}$ where C_0 is trained on all $|S|$ sensors and C_i is trained on all $|S| - i$.

7.2.2 Failure detection

SMART detects node failures by analyzing the relative performance of the classifiers that had that node in their training set versus those classifiers that did not. For example, when sensor s fails, we expect a change in the relative behavior of classifier $C(S, S - s)$ and that of classifier $C(S - s, S - s)$. Therefore, we calculate the F-score of each of these classifiers with respect to the original classifier C_0 to measure the similarity between their outputs. For example, the F-score for a classifier $C_i(S - s_i, S)$ is calculated as

$$F - score_{C_i} = \frac{2 \times NDA_{both}}{2 \times NDA_{both} + NDA_{C_0} + NDA_{C_i}}$$

where:

1. NDA_{BOTH} is the number of activity instances detected by both classifiers C_0 and C_i , i.e. the number of true positives with respect to classifier C_0 ;
2. NDA_{C_0} is the number of activities detected only by classifier C_0 , i.e. the number of false negatives for classifier C_i with respect to classifier C_0 ;

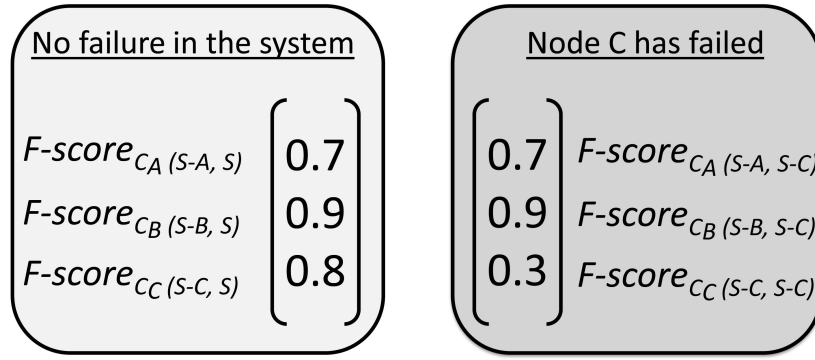


Figure 7.4: Multiple F-score vectors are used to detect when an application-level failure occurs in the system and to identify which node has caused the failure.

3. NDA_{C_i} is the number of activities detected only by classifier C_i , i.e. the number of false positives for classifier C_i with respect to classifier C_0 .

Each of the $|S|$ classifiers has an F-score associated with it, forming an F-score vector:

$$\begin{bmatrix} F - score_{C_1(S-s_1, S)} \\ F - score_{C_2(S-s_2, S)} \\ \vdots \\ F - score_{C_{|S|}(S-s_{|S|}, S)} \end{bmatrix}$$

This vector characterizes the behavior of the system when there are no failures. If a failure occurs, based on the severity of the failure, it might affect some or even all of the values in the F-score vector. Since different failures change the behavior of the classifiers in different ways, our approach tries to identify if a failure has occurred and which node has failed by using another level of classification on the F-score vector. Figure 7.4 illustrates this principle. There are three nodes in the system: A, B, and C. In addition to the classifier instance C_0 trained on all nodes, there are three more classifier instances trained by holding one of the nodes out: C_A , C_B , and C_C . The F-score vector characterizes the relative behavior of C_A , C_B , and C_C to that of C_0 . Since all four classifiers have been trained on a different subset of the nodes, they might not be detecting the exact same number of activities. The first F-score vector represents the state of the system when none of the nodes has failed. When sensor C fails, this affects the system and the relative behavior of the classifiers changes, which is represented by the second F-score vector. C_A , C_B , and C_0 have all been trained with node C. Therefore, their relative ratios will remain similar, since they will be affected in a similar way. On the other hand, classifier C_C was trained by holding node C out and will therefore change its behavior relative to classifier C_0 . By analyzing the changes in the F-score vectors, SMART attempts to both infer that a failure has occurred in the system, and to identify the cause of that failure.

Failure recognition is performed in two steps:

Step 1: Is there a failure or not? We use a *failure detection* classifier (FDC) that is trained to distinguish between non-failure F-score vectors and failure F-score vectors. We use historical data to generate both failure and non-failure F-score vectors and train the FDC. The failure F-score vectors are generated by artificially introducing failures in the historical data though modifying the readings of the “failed” nodes. At run time, the system calculates the relative F-scores for the $|S|$ classifiers and builds a F-score vector. The FDC is used to determine if the F-score vector represents a system that has a failure or not.

Step 2: Which node has failed? Once a node failure has been detected, a second *failure identification* classifier (FIC) is used to determine which of the nodes has failed. The FIC is trained to distinguish between different node failures. Again, this is done with the help of historical data where failures are simulated by modifying the readings of the “failed” nodes.

An important issue to consider is that small fluctuations between F-score vectors might occur even without failures, particularly if residents of a home alter their typical activity patterns. In addition, non-severe failures might cause very small changes to the behavior of the classifiers. Therefore, the FDC, which determines if a failure has occurred or not, is likely to produce a number of false positive and false negative classifications. To improve the accuracy of the failure detection classifier we can increase the size of the historical data used for training. We can also increase the failure detection latency, which is the time used to collect data after the failure has occurred. Section 7.5 discusses the effect of the failure detection latency on the failure detection and identification accuracy of our approach.

7.2.3 Node failure severity analysis

The severity analysis is based on the key insight that node failures are going to impact the application differently based on the available level of redundancy in the system. Consider the scenario in Figure 7.5. The activity we are interested in is *cooking*. There are two sensors in the kitchen but only the star-shaped one is located close to the appliances and the sink. If this sensor fails, the detection accuracy for *cooking* will be impacted significantly. Therefore, this failure is severe and maintenance will be necessary if it occurs. On the other hand, if the oval-shaped sensor fails, this might have a small effect on the accuracy of detecting *cooking*. In that case, the node does not need to be repaired immediately. Also, if the node redundancy in the kitchen had been higher, even the failure of the star-shaped node might have been considered non-severe since it might have barely affected the detection accuracy.

When a failure is detected at run time, we use historical data and apply data replay analysis to evaluate what effect this failure would have had on the application had it occurred in the past. Assume that node s

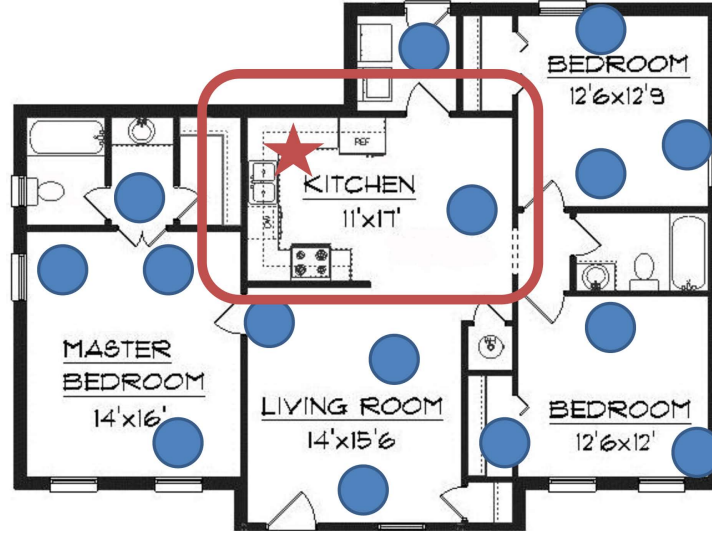


Figure 7.5: The severity of a sensor node failure is very strongly correlated with the level of redundancy in the system. Therefore, because of the low node redundancy in the kitchen, if the star-shaped node fails, the accuracy of detecting *cooking* activities would be severely impacted.

fails, where $s \in S$. We use historical data to estimate $C_{Accuracy}(S, S)$ that would have been reported by the classifier before the failure. After that, we remove all of the readings produced by the failed node s from both the training and the testing set and calculate $C_{Accuracy}(S - s, S - s)$. The difference

$$|C_{Accuracy}(S, S) - C_{Accuracy}(S - s, S - s)|$$

determines the effect of the failure on the application. As mentioned earlier, a failure is considered severe if it decreases the detection accuracy below the severity threshold TH_S , and maintenance is dispatched every time a severe failure is detected. Some applications might consider failures to be severe if they change the number of detected activities by more than 30% of the original number of detected activities. However, for safety-critical applications, a failure could be severe even if it changes the number of detections by 5%. Instead of using a global severity threshold, an application can also specify per-activity severity thresholds. In this way, if some activities are considered more critical than others, they can be assigned stricter severity thresholds.

7.2.4 Maintaining detection accuracy under failures

SMART is able to adapt and maintain high activity recognition accuracy even in the presence of node failures by simultaneously running more than one classifier instance, each of which is trained on a different subset of nodes using the *hold one sensor out* approach. Additionally, SMART simultaneously uses a variety of classifier types, including Naive Bayesian (NB), Hidden Markov Model (HMM), Hidden Semi-Markov Model (HSMM), and decision trees. SMART takes advantage of the fact that classifiers perform differently due to

the ability to extract different types of patterns. As failures are detected at run time, SMART automatically switches to the classifier instance among all types and training sets that performed best on the training data, given the set of nodes known to be functional and failed.

7.3 Experimental setup

We evaluate SMART on 3 houses in 2 publicly available activity recognition datasets. We study one of the CASAS datasets provided by WSU [145]. This dataset contains over 40 days of labeled data from over 60 sensors and it was collected in a two-resident home. The activities are individually labeled per resident. Since one of the residents performs only a small variety of activities, we only consider the activities performed by the other resident. The second dataset [146] contains two single-resident homes: House A and House B. The data from House A was collected in the course of 25 days from 14 sensors. House B contained 27 sensors and the data was collected over 13 days.

Activity recognition for both datasets is performed by dividing each day into fixed-length times slots and the activity performed in each time slot is classified based on the sensor firings within that particular time slot and the time of day. The subjects living in the houses annotated a number of different activities that include *sleeping*, *toileting*, *prepare breakfast*, *prepare dinner*, and *showering*. We use this annotation as the ground truth. The second dataset also contains NB and HMM classifiers for activity recognition. In our experiments we use these classifiers for the evaluation of activities from WSU, House A, and House B. Although we run experiments with only four classifier classes, namely NB and HMM, we are confident that other classifier classes can easily be incorporated into our approach.

In the above datasets, the detection of simple activities, such as *leave house* and *toileting*, relies on a single sensor. When that sensor fails, the corresponding activity can no longer be detected. Such a failure is considered to be severe and maintenance should always be dispatched. In our evaluation we focus on *complex* activities whose detection relies on multiple sensors. This gives us the opportunity to encounter node failures with different levels of severity. There are many complex activities that one might want to monitor, such as cooking, cleaning, studying, exercising, getting dressed, and unpacking groceries. In our experiments we evaluate the detection of the following seven activities:

WSU: *prepare breakfast*, *prepare lunch*, and *prepare dinner*

House A: *prepare breakfast* and *prepare dinner*

House B: *prepare breakfast* and *prepare dinner*

Table 7.1 shows the IDs and location of the nodes participating in the detection of the activities we analyze.

WSU	House A	House B
motion sensor 1	microwave	fridge
motion sensor 2	cups cupboard	plates cupboard
motion sensor 3	fridge	cutlery drawer
door sensor 1	plates cupboard	stove lid
door sensor 2	freezer	sink
door sensor 3	dishwasher	toaster
burner sensor	pans cupboard	microwave
hot water sensor	groceries cupboard	motion sensor

Table 7.1: Sensors participating in the *prepare breakfast*, *prepare lunch*, and *prepare dinner* activities in the WSU house, *prepare breakfast* and *preparing dinner* activities in House A and House B.

The severity threshold for all experiments is set to $TH_S = 0.3$. Empirical analysis we performed with smaller thresholds, such as 0.1 and 0.2, showed that, due to the low level of redundancy in House A and House B, most of the node failures are classified as severe. Therefore, we chose a threshold that allowed us to have a more diverse failure profile for all houses.

The datasets do not contain failure information. All node failures in the experiments were simulated by modifying the values reported by the “failed” node. For the “stuck at” failure we set the value of the failed node to 1. For the misplacement failures, we replaced the data of the failed sensor s with data from a sensor located at s' new position.

We use *F-score* as accuracy measurement for all classifiers:

$$\text{F-score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (7.1)$$

where *Precision* and *Recall* are defined as:

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}} \quad (7.2)$$

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}} \quad (7.3)$$

10-fold cross validation was performed for all experiments analyzing the accuracy of SMART in detecting and identifying node failures. In addition, we keep the number of training days constant for each of the data sets. For example, house A has data for 25 days total. Therefore, for experiments where the number of testing days, i.e. the failure detection latency, was varied from 1 to 6 days, the number of training days was always 19. This was done to avoid bias in the results due to varying size of the training data.

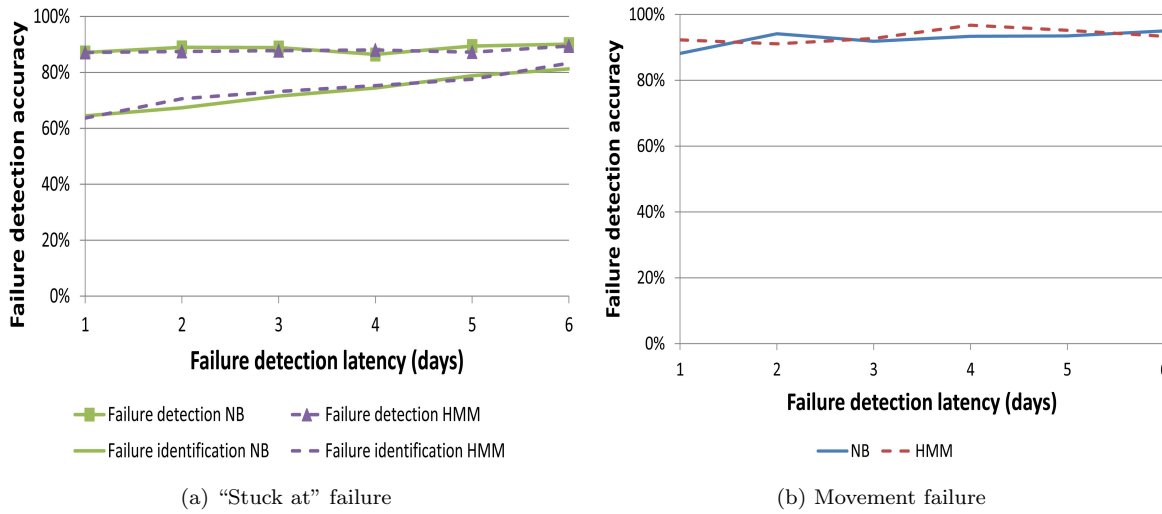


Figure 7.6: Detecting non-fail-stop node failures of the kitchen nodes in the WSU house for both NB and HMM. SMART achieves more than 85% accuracy in detecting a severe “stuck at” non-fail-stop failure and about 80% accuracy in identifying which one of the 8 kitchen sensor nodes has caused the failure. It also achieves more than 95% accuracy in detecting a failure caused by rotating the kitchen motion sensor towards the adjacent living room in the house from the WSU dataset.

7.4 Results

7.4.1 Detecting sensor node failures

We analyze the behavior of SMART when “*stuck at*” and *node movement* failures are introduced in the system. A “stuck at” failure is defined as a series of values that exhibit variance close to zero for a period of time greater than expected. In our experiment we simulated the “stuck at” scenario by changing the readings of the failed sensor so they indicate that the sensor fires at all time slots after the failure.

Figure 7.6a shows SMART’s accuracy in detecting non-fail-stop “stuck-at” failures and identifying which one of the nodes has caused the failure. The results are for the WSU dataset and include failures for all of the 8 kitchen sensors. The x-axis is the failure detection latency, which is the number of days elapsed after the node failure has occurred. Figure 7.6a shows that SMART achieves more than 85% accuracy in detecting that there is a failure in the system. Further, SMART can identify the failed node with more than 80% accuracy. Another observation we make is that, although the accuracy of our approach increases with the number of days elapsed after the failure has occurred, SMART can quickly achieve significant failure detection and identification accuracy. Figure 7.6a shows that SMART’s failure detection accuracy is above 85% from the first day it can identify which one of the 8 kitchen nodes has failed with more than 70% accuracy in only 2 days.

The next experiment evaluates SMART’s behavior in the presence of failures caused by node displacement.

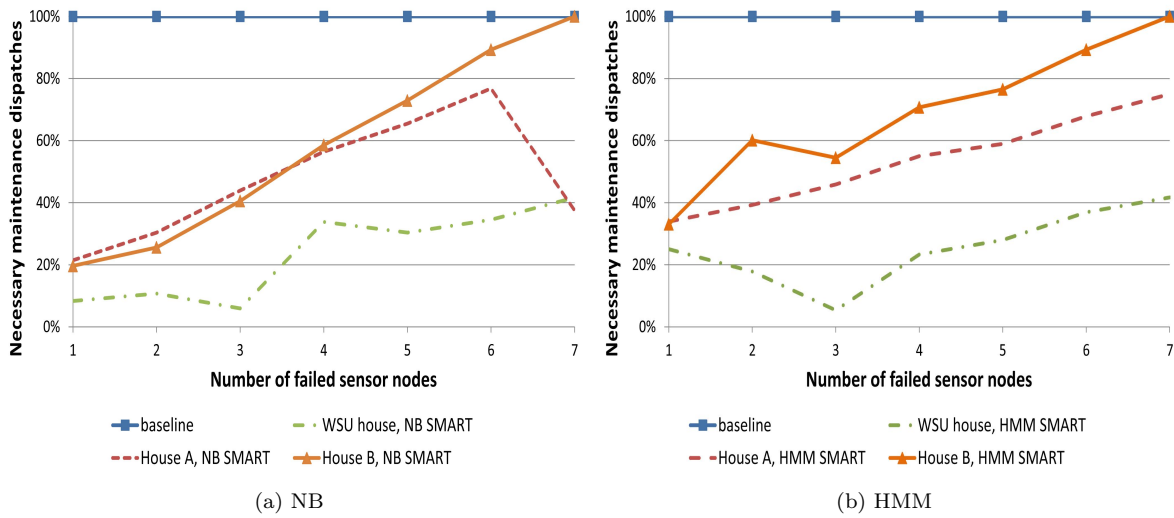


Figure 7.7: Distribution for number of maintenance dispatches for the all kitchen activities the WSU house, House A, and House B. Compared to a baseline where maintenance has to be dispatched every time a node failure occurs, our approach repairs only the severe failures and thus decreases the number of maintenance dispatches by 55%.

Figure 7.6b shows the results for a scenario where the motion sensor in the kitchen of the WSU dataset moved in a way that it can no longer detect movement in the kitchen. Instead, it is pointed towards the living room, which is adjacent to the kitchen. Similar to the “stuck at” failure, SMART is able to detect this displacement failure very accurately. Its accuracy exceeds 90% just one day after the failure has occurred.

7.4.2 Node failure severity assessment

With the baseline repair technique, every time a sensor node fails a repair is necessary and maintenance has to be dispatched. Figure 7.7 shows the effect of our approach on the number of maintenance dispatches. For this experiment we only look at the behavior of the different classifier classes in isolation, i.e. SMART has one classifier class to work with. This allows us to see the differences between the classifier classes. The activities shown in the figure are *prepare breakfast*, *prepare lunch*, and *prepare dinner* from the WSU house and *prepare breakfast* and *prepare dinner* from House A and House B. The figure contains the maintenance distribution over all possible combinations of failures of the sensor nodes used to detect this activity.

From Figure 7.7a we see that when one of the nodes fails we need to perform a repair in less than 20% of the cases on average. This happens because only 1 of the 8 nodes in the WSU home and 3 of the 7(8) nodes in House A and House B, that participate in the detection of the kitchen activities have high influence on the detection accuracy of the activity. Therefore, only a small number of the single-node failures are severe.

Classifier	WSU			House A		House B	
	Breakfast	Lunch	Dinner	Breakfast	Dinner	Breakfast	Dinner
baseline	1	1	1	1	1	1	1
NB SMART	4.5	2.4	6	2.2	5	4.1	2.9
HMM SMART	5.5	2.9	2.4	2.1	2.3	2.3	1.9

Table 7.2: Average MTTF for all five activities. Our approach increases the MTTF 3.2 times on average. The MTTF improvement achieved with each classifier varies with the nature of the activity. SMART chooses the most suitable classifier at any time and thus achieves the highest MTTF.

A maintenance dispatch can be avoided whenever a failure is not severe. When a NB classifier is used, the number of dispatches is decreased by 84% with one sensor failure, 22% with two failed sensors, and so on.

There are other techniques that could fit between the baseline, where a repair is necessary every time a sensor node fails, and SMART on the spectrum of number of necessary maintenance dispatches. Some of these approaches could rely on component redundancy and require a repair only when the number of operational nodes falls below a threshold; or they might consider additional low-level parameters to determine if a repair is necessary or not. However, rather than using low-level semantics, SMART considers the high-level application behavior of the system. Therefore, we believe that SMART will require fewer maintenance dispatches than any approach that analyzes component-level properties.

To evaluate SMART's impact on the MTTF of the application, where MTTF is defined as number of time units after which the detection accuracy falls below TH_S , we compare our approach, where we use an ensemble of classifier instances some of which are preemptively trained for failures, to a baseline, where there is just one classifier instance trained on all nodes. We consider all possible sequence traces of sensor node failures. For each of those traces, we evaluate at what point in time, i.e. after how many node failures, a repair should be performed. Unlike the baseline, our approach determines that the application has failed not when the first node fails, but when the first severe node failure occurs.

Table 7.2 shows the average MTTF values for all three houses. We introduce a new node failure after each time unit. With the baseline approach, since every node failure is also considered to be an application failure, the MTTF of the system is always 1 time unit. The MTTF achieved by different classifiers varies based on the activity. For example, for activity *preparing breakfast* from the WSU house, using HMM results in higher MTTF than when NB is used. However, NB considerably outperforms HMM for the *preparing dinner* activity again from the same house. We attribute this difference to the fact that, based on which activity is being analyzed, the classifiers might rely on a different subset of sensor nodes. In addition, our SMART approach dynamically chooses the best classifier depending on the current state of the system, it maintains the highest application MTTF.

Figure 7.8 shows a more detailed view of the MTTF for *prepare breakfast* from the WSU house. When

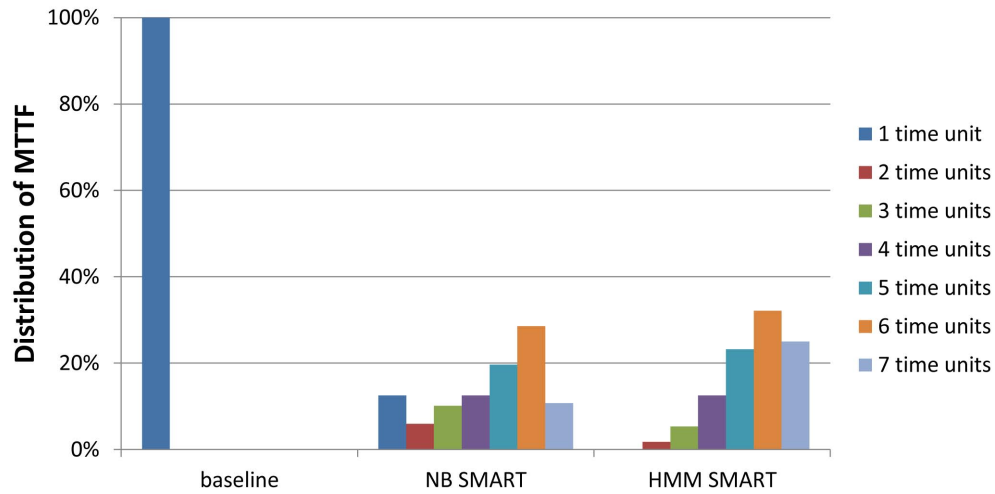


Figure 7.8: Mean time to failure distribution for the WSU house *preparing breakfast*. We assume that a new sensor failure occurs after each time unit. The average MTTF for NB and HMM are 4.5 and 5.5 time units respectively.

SMART uses a NB classifier, the system can sometimes survive 7 failures before it needs maintenance. This happens when all 7 non-severe node failures occur first. In that case, the application's accuracy will remain above the severity threshold until the seventh failure, which has to be severe.

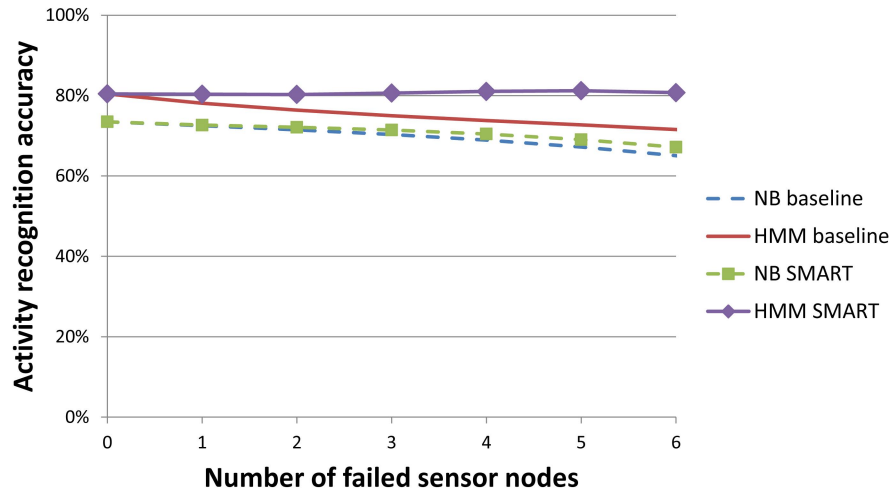


Figure 7.9: Average activity recognition accuracy improvement for the activity *preparing breakfast* in the WSU house. Compared to the baseline NB and HMM classifiers trained with all nodes, the NB and HMM preemptively trained for node failures improves the activity detection accuracy in the presence of failures.

7.4.3 Maintaining high activity recognition accuracy under failures

Compared to a classifier trained on all nodes in the system, our approach achieves higher activity recognition accuracy in the presence of node failures. Figure 7.9 shows the average improvement in the activity recognition accuracy for all possible n -node failures, where n grows between 0 and 6. For this experiment, the baseline NB and HMM classifiers were trained on all nodes in the system. The NB and HMM classifiers that were preemptively trained for failures achieve higher activity recognition accuracy in the presence of failures than their respective baselines. SMART chooses the most accurate classifier from an ensemble of NB and HMM instances, trained for the presence of failures. For the experiment in Figure 7.9, SMART's accuracy overlaps with that of HMM.

The next experiment evaluates the maximum accuracy improvement achieved by the classifiers trained for failures over the classifiers that were trained on all nodes. For example, we compare the accuracy of NB that is preemptively trained by holding nodes out to that of NB trained on all nodes in the system. Figure 7.10 shows the average results for all kitchen activities in the WSU house for NB and HMM. Similarly to the previous graph, we evaluate the average improvement of the activity recognition accuracy for all possible n -node failures, where n grows to up to 6 failures.

Table 7.3 shows the average accuracy improvement for the activities in all three houses when failure-trained classifiers are used. The level of improvement for a classifier class differs based on the activity. For example, the improvement achieved for activity *prepare breakfast* for House A is very low. This is because this activity mainly uses two kitchen sensors and when they fail, training without them does not improve the activity

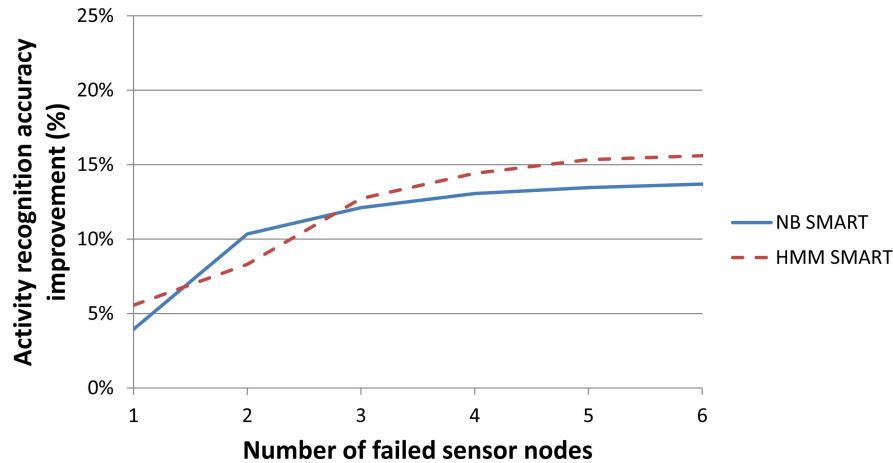


Figure 7.10: Average activity recognition accuracy improvement for all kitchen activities in the WSU house. Using ensembles of classifier instances trained for failures instead of a single instance trained on all nodes significantly improves the activity recognition accuracy under failures.

detection accuracy. Similarly, for activity *prepare dinner* in House B we see very high accuracy improvement, which is because the sensors that are most frequently used by this activity are functionally redundant.

Table 7.3 also shows us that the accuracy improvement for the same activity could vary based on the classifier. This occurs because the different classifiers put different weights on the various sensors they use. SMART takes full advantage of this by choosing the most accurate classifier at any given state of the system.

Classifier	WSU			House A		House B	
	Breakfast	Lunch	Dinner	Breakfast	Dinner	Breakfast	Dinner
NB SMART	5%	16%	11%	0.1%	9%	5%	51%
HMM SMART	17%	6%	15%	1%	14%	19%	47%

Table 7.3: We evaluate how using SMART instead of classifiers trained on all nodes affects the activity recognition accuracy. The average accuracy improvement under the presence for failures the classifiers we analyze is 15%.

7.5 Discussion

We analyze the relationship between the failure detection accuracy of a node and how frequently this node is used for a particular activity. For each node n we define a *node usage ratio* per activity, which is the percentage of instances of that activity where node n was used. The kitchen nodes in the WSU dataset are all frequently used during the execution of the kitchen activities. Therefore, we performed this experiment using House A and considering both *prepare breakfast* and *prepare dinner* activities. Figure 7.11 shows that there is a positive correlation between whether a node fires regularly during the execution of a particular

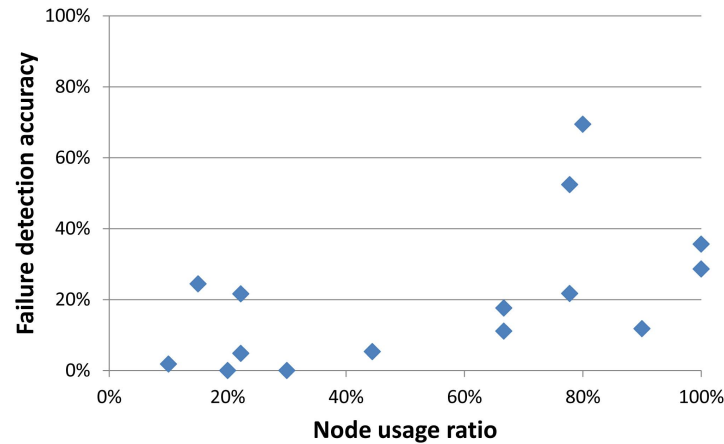


Figure 7.11: Relationship between the failure detection accuracy of a node and how frequently this node is being used for activities *prepare breakfast* and *prepare dinner* for House A. The majority of nodes in this dataset have low usage ratios, which results in low failure detection accuracy even for the nodes with high usage ratio when all nodes are considered by the failure detection mechanism.

activity, i.e. the importance of that node for the activity, and how accurately we can detect the node’s failure. We can see that there are a number of kitchen nodes which rarely fire during activities *prepare breakfast* and *prepare dinner* and the failure detection accuracy for these nodes remains close to 0. On the other hand, the nodes important to these activities, i.e. the nodes that fire frequently, have higher failure detection accuracy.

Including both the important and the non-important nodes in the failure detection process significantly impacts the detection accuracy for the important nodes. Figure 7.12 shows the accuracy of detecting a “stuck at” failure for the important nodes for the kitchen activities in House A. We see that when only the important nodes are considered, the failure detection accuracy increases dramatically.

7.6 Summary

In this chapter we have presented a general failure detection, assessment, and adaptation approach for smart home applications. Even though the datasets used in our evaluation do not directly address fault tolerance or redundancy, our approach still achieves significant gains. SMART decreases the number of maintenance dispatches by 55% and almost triples the MTTF of the application on average. It also maintains sufficient activity recognition accuracy in the presence of failures by dynamically updating the classifiers at run time so they can adapt to the failures that occur. SMART detects all application-level failures at run time with over 85% accuracy. Further, SMART improves the activity recognition accuracy under node failures by more than 15% on average.

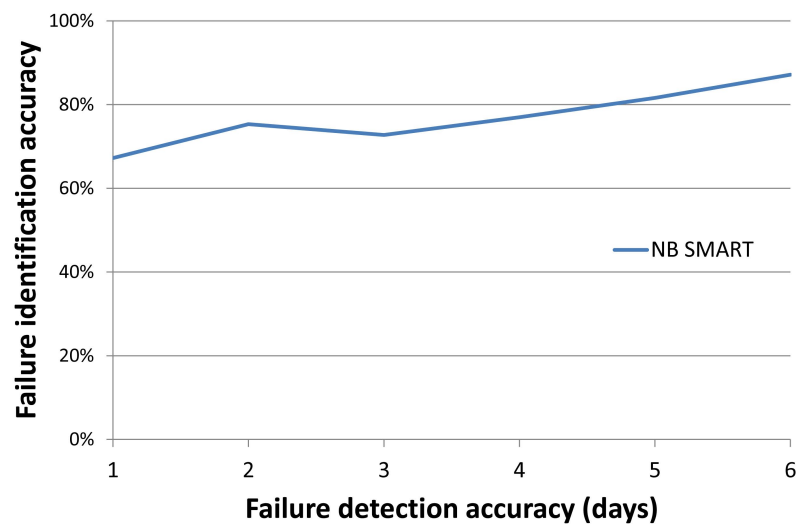


Figure 7.12: Detecting “stuck at” failures for the important nodes participating in activities *prepare breakfast* and *prepare dinner* in House A. The failure detection accuracy for the important nodes for an activity increases when only the failures of these nodes are being considered. SMART achieves about 80% accuracy in identifying which of the important nodes has failed.

Part V

Conclusions and future work

Chapter 8

Conclusions and future work

EVENT detection is a vital component in a wide variety of sensor network applications. The main focus of the research presented in this dissertation was to facilitate the development of robust real time event services for wireless sensor networks. These event services need to be accurate and minimize false alarms, run in real-time, and provide a sufficient level of robustness. To address this we have:

- Developed a formal event specification language, MEDAL, designed specifically for sensor network applications.
- Studied the effect of using fuzzy logic on the accuracy of event detection.
- Designed a number of techniques to maintain sensor network application robustness in the presence of node failures.

In the rest of the chapter we summarize the techniques and results presented in this dissertation. We also discuss possible future work in each of the three areas studied in this dissertation - event specification, event detection, and robustness to node failures.

8.1 Results and contributions

8.1.1 Event specification

In Chapter 3 we have presented MEDAL, a formal event description language we have developed specifically for sensor network applications. MEDAL is an extended Petri net, and it combines features from Timed, Colored, and Stochastic Petri nets. MEDAL has a number of properties, which make it very suitable for describing event-driven WSN applications:

- It addresses fundamental properties of sensor networks such as probabilities, temporal and spatial dependencies, and heterogeneity.
- We have extended MEDAL's syntax to include dedicated radio transitions, inhibitor arcs, and feedback loops. This has allowed MEDAL to model a number of additional key sensor network features, such as communication, actuation, and feedback control.
- MEDAL could help tremendously in the effort to bridge the gap between sensing and event recognition by shifting where event detection takes place. Typically the sensor nodes in the network send their readings to a base station node, which performs the event detection by analyzing the sensor readings. However, MEDAL can help in moving the event detection from the base station to the sensor nodes themselves. This step has a number of advantages, including lower event detection latencies, smaller communication needs, and higher energy efficiency.
- As a system analysis tool, MEDAL can capture the structural, spatial, temporal, and real-time properties of a complex event detection system. Therefore, MEDAL can be used to assist system designers to identify inconsistencies and potential problems with their applications. Further, MEDAL's analysis capabilities can be used to narrow down the origin of failures that occur after the network has been deployed.
- Our experience has shown that MEDAL can model a wide variety of event-driven application. We have also used MEDAL to model three existing WSN applications: volcano monitoring, rural fire detection, and flood detection. As a proof of concept we have also described our experience using MEDAL to model a beef monitoring application. Further, we have evaluated MEDAL's expressiveness by comparing it to other approaches that have been used to describe sensor network event-based applications.

In Chapter 4 we show how MEDAL can also be used to model and analyze real-time data stream queries, QoS management mechanisms, and the relationships between them. Unlike previous work, where query models and system control logic are designed and analyzed separately, MEDAL allows us to merge these two components into a single comprehensive system model. The advantage of this combined model is that it can be used not only to predict the workload and estimate the query cost, but also to model and analyze the interactions between the input and output of the query plans and the data control mechanism, which gives us a much better understanding of the system.

8.1.2 Event detection

A disadvantage of the current event detection approaches used in WSNs is that they cannot properly handle the often imprecise sensor readings. In Chapter 5 we show that fuzzy logic is a powerful and accurate mechanism which can successfully be applied not only to fire detection but to any event detection sensor network application. We have empirically evaluated the effect of using fuzzy logic on the accuracy of event detection. Our results show that:

- Compared to using crisp values, fuzzy logic maintains a high event detection accuracy levels despite fluctuations in the sensor values. Fuzzy logic helps decrease the number of false positives, while still providing fast and accurate event detection.
- Incorporating the readings of neighbor nodes in the decision process further improves the event detection accuracy.
- The size of the rule-base might present a challenge for the resource-constrained nodes in a sensor network. Therefore, we have developed three rule-base reduction techniques to help decrease the memory requirements and the processing time of the rule-base. Our evaluation shows that the rule-base reduction techniques are very effective and preserve both the correctness and the timeliness of event detection. Using two of these techniques, namely, *combining rules with similar outcomes* and *incomplete rule-base*, reduces the size of our experimental rule-base by more than 70%.
- Compared to two well-established classification algorithms, a Naive Bayes classifier and a decision tree, fuzzy logic provides comparable event detection accuracy.

8.1.3 Robustness to node failures

A major disadvantage of the current reliability and health monitoring techniques used for WSN applications is that they monitor the performance of the low-level components of the system instead of the reliability of the application layer. Our work addresses this issue in WSNs in two ways:

Run time assurance

Chapter 6 discusses a run time assurance methodology to help designers and users verify at run time that an application is able to meet its high-level requirements and maintain satisfactory performance even in the presence of node failures. The work presented in this chapter has the following contributions:

- We have implemented an automated test generation framework as part of the run time assurance methodology.

- To decrease the vast number of generated test cases, we have developed three test suite reduction techniques, two of which are unique to the nature of WSN applications and take advantage of the network's topology and node redundancy.
- The evaluation of RTA on a fire detection application shows that RTA performs much better than health monitoring in identifying application-level failures and almost just as good in identifying low-level failures. In addition, RTA incurs 33% less communication overhead and decreases by 70% the amount of maintenance work that needs to be done in order to keep the system operational.

SMART

Chapter 7 presents a general failure detection, assessment, and adaptation approach for smart home applications. This approach, a Simultaneous Multi - classifier Activity Recognition Technique (SMART), uses application-level semantics to detect, assess, and adapt to sensor failures. SMART provides three key features:

1. It detects non-fail-stop faults resulting from e.g., node displacement or node relocation;
2. It assesses the importance of a sensor failure in order to decrease the number of necessary maintenance dispatches;
3. It adapts the system to the failures in order to maintain sufficient activity recognition accuracy.

Our evaluation of SMART on a number of publicly available datasets shows that:

- SMART decreases the number of maintenance dispatches by 55% and almost triples the MTTF of the application on average. It also maintains sufficient activity recognition accuracy in the presence of failures by dynamically updating the classifiers at run time so they can adapt to the failures that occur.
- SMART detects application-level failures at run time with over 85% accuracy.
- SMART improves the activity recognition accuracy under node failures by more than 15% on average.

As long-lived sensor network applications become more common in real homes, the need for fault tolerance and ground truth validation will lead to increased node redundancy. We expect that applying SMART to moderately and highly redundant systems will result in much higher gains.

8.2 Limitations and future work

8.2.1 Event specification

For future work, it would be desirable to develop a GUI-based tool that would be used to build and analyze sensor network applications. This tool will allow application designers to take full advantage of MEDAL and its modeling and analysis capabilities. An additional advantage of this tool is that it could help automate the translation from a formal application model into code even further. Currently, for RTA, an application designer has to manually translate the MEDAL model into a specification script. This operation is still error-prone and might lead to mismatch between the MEDAL model and the script. The graphical MEDAL model designed with the MEDAL tool can be automatically transformed into a specification script. This would make the translation process faster and less error-prone since automating this step will help avoid any errors the designer may introduce while writing the script.

This MEDAL tool could also be used for data-stream applications to perform operator cost analysis and selectivity estimation. It could also help model dependencies between the data stream system and the surrounding environment, which could be extremely useful for context-aware workload prediction.

8.2.2 Event detection

A limitation of our fuzzy logic study was that although we used real fire data, all experiments were done in simulation. Perform experiments on a sensor test bed and real deployments will allow us to better evaluate how using fuzzy logic influences the accuracy and speed of event detection when the decision logic is run on sensor nodes. In addition, it will help study the effect of applying temporal constraints on the accuracy of event detection.

Investigate how fuzzy logic can be integrated into MEDAL is another direction for future work. It will provide sensor network designers with a single comprehensive event modeling system. It will also allow us to evaluate whether using MEDAL and fuzzy logic simultaneously has a positive or a negative effect on the accuracy and performance of the event detection service.

8.2.3 Robustness to node failures

Although the results presented in this area are very promising, this is just the foundation for the future work on detecting, assessing, and adapting to node failures in sensor network home applications. Our approaches and evaluation have a few limitations that can be addressed:

- Both RTA and SMART have been evaluated on small datasets or on a test bed. In the future, RTA and SMART can be evaluated on a deployment designed with better fault tolerance features. SMART can also be integrated with the design of an activity recognition system and evaluate how providing appropriate node redundancy at design time can further aid fault tolerance, reduce dispatches, and improve accuracy. In addition, SMART can be applied to smart home applications other than activity recognition, and analyze its accuracy and applicability.

- Further, the effect of integrating out approaches with health-monitoring and trust-based techniques can be investigated. We expect that this will expand the types of failures that are being detected and improve the accuracy of failure detection. In addition, SMART cannot accurately detect the failures of sensors that are not frequently used in any of the activities. Since the failure of these sensors will cause minimal changes in the behavior of the classifiers, SMART is likely to attribute these changes to natural activity pattern fluctuations rather than a failure. However, SMART can be combined with state of the art health-monitoring systems, which can help accurately detect fail-stop failures experienced by the rarely used nodes.

Bibliography

- [1] Geoffrey Werner-Allen, Konrad Lorincz, Matt Welsh, Omar Marcillo, Jeff Johnson, Mario Ruiz, and Jonathan Lees. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10:18–25, March 2006.
- [2] Jaime Lloret, Miguel Garcia, Diana Bri, and Sandra Sendra. A wireless sensor network deployment for rural and forest fire detection and verification. *Sensors*, 9(11):8722–8747, 2009.
- [3] Elizabeth A. Basha, Sai Ravela, and Daniela Rus. Model-based monitoring for early warning flood detection. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 295–308, New York, NY, USA, 2008. ACM.
- [4] Shuoqi Li, Sang H. Son, and John A. Stankovic. Event detection services using data service middleware in distributed sensor networks. In *Proceedings of the 2nd international conference on Information processing in sensor networks*, IPSN'03, pages 502–517, Berlin, Heidelberg, 2003. Springer-Verlag.
- [5] Wai Fu Fung, David Sun, and Johannes Gehrke. COUGAR: the network is the database. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 621–621, New York, NY, USA, 2002. ACM.
- [6] Ramesh Govindan, Joseph Hellerstein, Wei Hong, Samuel Madden, Michael Franklin, and Scott Shenker. The sensor network as a database. Technical Report 02-771, Computer Science Department, University of Southern California, 2002.
- [7] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 491–502, New York, NY, USA, 2003. ACM.
- [8] Michael Worboys. Event-oriented approaches to geographic phenomena. *International Journal of Geographical Information Science*, 2008.
- [9] Fulvio Babich and Lia Deotto. Formal methods for specification and analysis of communication protocols. *Communications Surveys Tutorials, IEEE*, 4(1):2–20, quarter 2002.
- [10] J. Ellsberger, D. Hogrefe, and A. Sarma. *SDL: Formal Object-oriented Language for Communicating Systems*. Prentice-Hall, Inc., 1997.
- [11] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1990.
- [12] H. El-Gendy and H. Baraka. Transformation of Lotos specifications to Estelle specifications. In *Proceedings of the 2nd IEEE Symposium on Computers and Communications (ISCC '97)*, ISCC '97, pages 215–221, Washington, DC, USA, 1997. IEEE Computer Society.
- [13] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1-2):134–152, 1997.
- [14] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Proceedings of the DI-MACS/SYCON workshop on Hybrid systems III : verification and control: verification and control*, pages 208–219, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

- [15] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. In *CAV 1997*, pages 460–463, London, UK, 1997. Springer-Verlag.
- [16] U. Dayal, A. P. Buchmann, and D. R. McCarthy. Rules are objects too: A knowledge model for an active, object-oriented databasesystem. In *Lecture notes in computer science on Advances in object-oriented database systems*, pages 129–143, New York, NY, USA, 1988. Springer-Verlag New York, Inc.
- [17] Alejandro P. Buchmann, Jürgen Zimmermann, José A. Blakeley, and David L. Wells. Building an integrated active OODBMS: Requirements, architecture, and design decisions. In *Proceedings of the Eleventh International Conference on Data Engineering, ICDE '95*, pages 117–128, Washington, DC, USA, 1995. IEEE Computer Society.
- [18] Sharma Chakravarthy, V. Krishnaprasad, Eman Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94*, pages 606–617, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.
- [19] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Event specification in an active object-oriented database. In *Proceedings of the 1992 ACM SIGMOD international conference on Management of data, SIGMOD '92*, pages 81–90, New York, NY, USA, 1992. ACM.
- [20] C. Liebig, M. Cilia, and A. Buchmann. Event composition in time-dependent distributed systems. In *Proceedings of the Fourth IECIS International Conference on Cooperative Information Systems, COOPIS '99*, pages 70–, Washington, DC, USA, 1999. IEEE Computer Society.
- [21] Scarlet Schwiderski. *Monitoring the behaviour of distributed systems*. PhD thesis, University of Cambridge, 1996.
- [22] S. Yang and S. Chakravarthy. Formal semantics of composite events for distributed environments. In *15th International Conference on Data Engineering*, pages 400–407, 1999.
- [23] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Universidad de Bonn, 1962.
- [24] Matthias Jantzen and Rüdiger Valk. Formal properties of place/transition nets. In *Proceedings of the Advanced Course on General Net Theory of Processes and Systems: Net Theory and Applications*, pages 165–212, London, UK, 1980. Springer-Verlag.
- [25] H.J. Genrich and K. Lautenbach. System modelling with high-level Petri nets. In *Theoretical Computer Science*, volume 13, pages 109–136. Elsevier, 1981.
- [26] K. Jensen. *Colored Petri Nets and the Invariant-Method*. Theoretical Computer Science, 1981.
- [27] Lars M. Kristensen, Soren Christensen, and Kurt Jensen. The practitioners guide to coloured Petri nets. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(2):98–132, 1998.
- [28] F. DiCesare, G. Harhalakis, J.M. Proth, M. Silva, and F. B. Vernadat. *Practice of Petri Nets in Manufacturing*. Chapman and Hall, London, 1993.
- [29] M.R. Tremblay and M.R. Cutkosky. Using sensor fusion and contextual information to perform event detection during a phase-based manipulation task. In *Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 3, pages 262 –267 vol.3, aug 1995.
- [30] Binjia Jiao, Sang H. Son, and John. Stankovic. GEM: Generic event service middleware for wireless sensor networks. *International Conference on Networked Sensing Systems (INSS)*, 2005.
- [31] Yuan Wei, Sang H. Son, and John A. Stankovic. Rtstream: Real-time query processing for data streams. In *Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, ISORC '06*, pages 141–150, Washington, DC, USA, 2006. IEEE Computer Society.

- [32] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.
- [33] Daniel Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12:120–139, 2003.
- [34] Emmanuel Tapia, Stephen Intille, and Kent Larson. Activity recognition in the home using simple and ubiquitous sensors. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 158–175. Springer Berlin / Heidelberg, 2004.
- [35] Christopher Wren and Emmanuel Tapia. Toward scalable activity recognition for sensor networks. In Mike Hazas, John Krumm, and Thomas Strang, editors, *Location- and Context-Awareness*, volume 3987 of *Lecture Notes in Computer Science*, pages 168–185. Springer Berlin / Heidelberg, 2006.
- [36] Paul Castro, Patrick Chiu, Ted Kremenek, and Richard R. Muntz. A probabilistic room location service for wireless networked environments. pages 18–34, 2001.
- [37] Marco Duarte and Yu-Hen Hu. Distance based decision fusion in a distributed wireless sensor network. pages 392–404, 2003.
- [38] Thomas M. Chen and Varadharajan Venkataramanan. Dempster-shafer theory for intrusion detection in ad hoc networks. *IEEE Internet Computing*, 9:35–41, November 2005.
- [39] Huadong Wu, Mel Siegel, Rainer Stiefelhagen, and Jie Yang. Sensor fusion using dempster-shafer theory. *Proceedings of the 19th IEEE Instrumentation and Measurement Technology Conference, 2002. IMTC/2002*, pages 7–12, 2002.
- [40] R. Murphy. Dempster-shafer theory for sensor fusion in autonomous mobilerobots. *Robotics and Automation, IEEE Transactions on*, pages 197–206, 1998.
- [41] A. Wood, G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, and J. Stankovic. Alarm-net: Wireless sensor networks for assisted-living and residential monitoring. Technical Report CS-2006-13, Computer Science Department, University of Virginia, 2006.
- [42] Dimitrios Lymberopoulos, Abhijit S. Ogale, Andreas Savvides, and Yiannis Aloimonos. A sensory grammar for inferring behaviors in sensor networks. In *Proceedings of the 5th international conference on Information processing in sensor networks*, IPSN '06, pages 251–259, New York, NY, USA, 2006. ACM.
- [43] H. Ghasemzadeh, J. Barnes, E. Guenterberg, and R. Jafari. A phonological expression for physical movement monitoring in body sensor networks. *5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pages 58–68, 2008.
- [44] Oliver Amft, Martin Kusserow, and Gerhard Trster. Probabilistic parsing of dietary activity events. 13:242–247, 2007.
- [45] Indranil Gupta, Denis Riordan, and Srinivas Sampalli. Cluster-head election using fuzzy logic for wireless sensor networks. In *Proceedings of the 3rd Annual Communication Networks and Services Research Conference*, pages 255–260, Washington, DC, USA, 2005. IEEE Computer Society.
- [46] JM Kim, SH Park, YJ Han, and TM Chung. CHEF: Cluster head election mechanism using fuzzy logic in wireless sensor networks. *10th International Conference on Advanced Communication Technology*, pages 645–659, 2008.
- [47] Hae Young Lee and Tae Ho Cho. Fuzzy logic based key disseminating in ubiquitous sensor networks. In *10th International Conference on Advanced Communication Technology*, page 958=962, 2008.

- [48] Byung Hee Kim, Hae Young Lee, and Tae Ho Cho. Fuzzy key dissemination limiting method for the dynamic filtering-based sensor networks. In *Proceedings of the intelligent computing 3rd international conference on Advanced intelligent computing theories and applications*, ICIC'07, pages 263–272, Berlin, Heidelberg, 2007. Springer-Verlag.
- [49] B. Lazzerini, F. Marcelloni, M. Vecchio, S. Croce, and E. Monaldi. A fuzzy approach to data aggregation to reduce power consumption in wireless sensor networks. *Annual meeting of the North American Fuzzy Information Processing Society NAFIPS*, pages 436–441, 2006.
- [50] Jin Myoung Kim and Tae Ho Cho. Routing path generation for reliable transmission in sensor networks using ga with fuzzy logic based fitness function. pages 637–648, 2007.
- [51] Shu-Yin Chiang and Jing-Long Wang. Routing analysis using fuzzy logic systems in wireless sensor networks. pages 966–973, 2008.
- [52] Qingchun Ren and Qilian Liang. Fuzzy logic-optimized secure media access control (fsmac) protocol wireless sensor networks. *Proceedings of the 2005 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety*, pages 37–43, 2005.
- [53] Saad A. Munir, Yu Wen Bin, Ren Biao, and Ma Jian. Fuzzy logic based congestion estimation for qos in wireless sensor network. *Wireless Communications and Networking Conference*, pages 4336–4341, 2007.
- [54] F. Xia, W. Zhao, Y. Sun, and Y.-C. Tian. Fuzzy logic control based qos management in wireless sensor/actuator networks. *Sensors*, 7:3179–3191, 2007.
- [55] Q. Liang and L. Wang. Event detection in wireless sensor networks using fuzzy logic system. In *Proceedings of the 2005 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety*, pages 52–55, 2005.
- [56] Mihai Marin-Perianu and Paul Havinga. D-FLER: a distributed fuzzy logic engine for rule-based wireless sensor networks. In *Proceedings of the 4th international conference on Ubiquitous computing systems*, UCS'07, pages 86–101, Berlin, Heidelberg, 2007. Springer-Verlag.
- [57] Nguyet T. M. Nguyen and Mary Lou Soffa. Program representations for testing wireless sensor network applications. In *Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting*, DOSTA '07, pages 20–26, New York, NY, USA, 2007. ACM.
- [58] John Regehr. Random testing of interrupt-driven software. In *Proceedings of the 5th ACM international conference on Embedded software*, EMSOFT '05, pages 290–298, New York, NY, USA, 2005. ACM.
- [59] Zhifeng Lai, S. C. Cheung, and W. K. Chan. Inter-context control-flow and data-flow test adequacy criteria for nesC applications. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 94–104, New York, NY, USA, 2008. ACM.
- [60] Thomas Clouqueur, Kewal K. Saluja, and Parameswaran Ramanathan. Fault tolerance in collaborative sensor networks for target detection. volume 53, pages 320–333, Washington, DC, USA, March 2004. IEEE Computer Society.
- [61] Lilia Paradis and Qi Han. A survey of fault management in wireless sensor networks. volume 15, pages 171–190, New York, NY, USA, June 2007. Plenum Press.
- [62] Linnyer Beatrys Ruiz, Isabela G. Siqueira, Leonardo B. e Oliveira, Hao Chi Wong, José Marcos S. Nogueira, and Antonio A. F. Loureiro. Fault management in event-driven wireless sensor networks. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '04, pages 149–156, New York, NY, USA, 2004. ACM.

- [63] Mengjie Yu, H. Mokhtar, and M. Merabti. Fault management in wireless sensor networks. *IEEE Wireless Communications*, 14(6):13–19, december 2007.
- [64] Y. Zhao, R. Govindan, and D. Estrin. Residual energy scan for monitoring sensor networks. In *IEEE Wireless Communications and Networking Conference*, pages 356–362, March 2002.
- [65] Chieh-Yih Wan, Shane B. Eisenman, and Andrew T. Campbell. CODA: congestion detection and avoidance in sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 266–279, New York, NY, USA, 2003. ACM.
- [66] IBM Autonomic Computing, 2008. <http://www.research.ibm.com/autonomic/>.
- [67] Tian He, Sudha Krishnamurthy, John A. Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Jonathan Hui, and Bruce Krogh. Energy-efficient surveillance system using wireless sensor networks. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, MobiSys '04, pages 270–283, New York, NY, USA, 2004. ACM.
- [68] Hongwei Zhang and Anish Arora. GS3: scalable self-configuration and self-healing in wireless networks. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, PODC '02, pages 58–67, New York, NY, USA, 2002. ACM.
- [69] Kamin Whitehouse, Gilman Tolle, Jay Taneja, Cory Sharp, Sukun Kim, Jaein Jeong, Jonathan Hui, Prabal Dutta, and David Culler. Marionette: using RPC for interactive development and debugging of wireless embedded networks. In *Proceedings of the 5th international conference on Information processing in sensor networks*, IPSN '06, pages 416–423, New York, NY, USA, 2006. ACM.
- [70] Jing Yang, Mary Lou Soffa, Leo Selavo, and Kamin Whitehouse. Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 189–203, New York, NY, USA, 2007. ACM.
- [71] Tamim Sookoor, Timothy Hnat, Pieter Hooimeijer, Westley Weimer, and Kamin Whitehouse. Macrodebugging: global views of distributed program execution. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 141–154, New York, NY, USA, 2009. ACM.
- [72] Mohammad Maifi Hasan Khan, Liqian Luo, Chengdu Huang, and Tarek Abdelzaher. SNTS: sensor network troubleshooting suite. In *Proceedings of the 3rd IEEE international conference on Distributed computing in sensor systems*, DCOSS'07, pages 142–157, Berlin, Heidelberg, 2007. Springer-Verlag.
- [73] Mohammad Maifi Hasan Khan, Hieu Khac Le, Hossein Ahmadi, Tarek F. Abdelzaher, and Jiawei Han. Dustminer: troubleshooting interactive complexity bugs in sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 99–112, New York, NY, USA, 2008. ACM.
- [74] Bor-Rong Chen, Geoffrey Peterson, Geoff Mainland, and Matt Welsh. Livenet: Using passive monitoring to reconstruct sensor network dynamics. In *Proceedings of the 4th IEEE international conference on Distributed Computing in Sensor Systems*, DCOSS '08, pages 79–98, Berlin, Heidelberg, 2008. Springer-Verlag.
- [75] Douglas Herbert, Vinaitheerthan Sundaram, Yung-Hsiang Lu, Saurabh Bagchi, and Zhiyuan Li. Adaptive correctness monitoring for wireless sensor networks using hierarchical distributed run-time invariant checking. volume 2, New York, NY, USA, September 2007. ACM.
- [76] Mohammad Maifi Khan, Tarek Abdelzaher, and Kamal Kant Gupta. Towards diagnostic simulation in sensor networks. In *Proceedings of the 4th IEEE international conference on Distributed Computing in Sensor Systems*, DCOSS '08, pages 252–265, Berlin, Heidelberg, 2008. Springer-Verlag.
- [77] L. Luo, T. He, G. Zhou, L. Gu, T. Abdelzaher, and J. A. Stankovic. Achieving repeatability of asynchronous events in wireless sensor networks with envirolog. In *Infocom 2006*, pages 1–14, April 2006.

- [78] S. Rost and H. Balakrishnan. Memento: A health monitoring system for wireless sensor networks. In *3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, pages 575–584, 2006.
- [79] L. Ruiz, J. Nogueira, and A. Loureiro. MANNA: A management architecture for wireless sensor networks. *IEEE Communications Magazine*, 41:116–125, February 2003.
- [80] Nithya Ramanathan, Kevin Chang, Rahul Kapur, Lewis Girod, Eddie Kohler, and Deborah Estrin. Sympathy for the sensor network debugger. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 255–267, New York, NY, USA, 2005. ACM.
- [81] Kevin Ni, Nithya Ramanathan, Mohamed Nabil Hajj Chehade, Laura Balzano, Sheela Nair, Sadaf Zahedi, Eddie Kohler, Greg Pottie, Mark Hansen, and Mani Srivastava. Sensor network data fault types. *ACM Trans. Sen. Netw.*, 5:25:1–25:29, June 2009.
- [82] Nithya Ramanathan, Thomas Schoellhammer, Eddie Kohler, Kamin Whitehouse, Thomas Harmon, and Deborah Estrin. Suelo: human-assisted sensing for exploratory soil monitoring studies. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 197–210, New York, NY, USA, 2009. ACM.
- [83] Saurabh Ganeriwal, Laura K. Balzano, and Mani B. Srivastava. Reputation-based framework for high integrity sensor networks. *ACM Trans. Sen. Netw.*, 4:15:1–15:37, June 2008.
- [84] D. Partridge and W. B. Yates. Engineering multiversion neural-net systems. *Neural Comput.*, 8:869–893, May 1996.
- [85] Giorgio Giacinto and Fabio Roli. Dynamic classifier selection. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, pages 177–189, London, UK, 2000. Springer-Verlag.
- [86] Fabio Roli, Giorgio Giacinto, and Gianni Vernazza. Methods for designing multiple classifier systems. In *Proceedings of the Second International Workshop on Multiple Classifier Systems*, MCS '01, pages 78–87, London, UK, 2001. Springer-Verlag.
- [87] H. Sagha, J. del R Millan, and R. Chavarriaga. Detecting and rectifying anomalies in body sensor networks. In *International Conference on Body Sensor Networks (BSN)*, pages 162–167, may 2011.
- [88] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30:122–173, March 2005.
- [89] Michael Franklin. Declarative interfaces to sensor networks. Presentation at NSF Sensor Workshop, 2004.
- [90] A. Ohta and K. Tsuji. Concurrent systems technology. Turing machine equivalence of time asymmetric choice nets. *IEICE Transactions on Fundamentals in Electronics, Communications and Computer Science*, E83-A(11):2278–2281, 2000.
- [91] Krasimira Kapitanova and Sang H. Son. Medal: a compact event description and analysis language for wireless sensor networks. pages 117–120, 2009.
- [92] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, apr 1989.
- [93] Shan Lin, Jingbin Zhang, Gang Zhou, Lin Gu, John A. Stankovic, and Tian He. ATPC: adaptive transmission power control for wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 223–236, New York, NY, USA, 2006. ACM.

- [94] Tian He, John A. Stankovic, Chenyang Lu, and Tarek Abdelzaher. Speed: A stateless protocol for real-time communication in sensor networks. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ICDCS '03, pages 46–, Washington, DC, USA, 2003. IEEE Computer Society.
- [95] Jiacun Wang. *Timed Petri nets: Theory and application*. Kluwer Academic Publishers, Boston, 1998.
- [96] Michel Hack. *Decidability questions for Petri Nets*. PhD thesis, Massachusetts Institute of Technology.
- [97] David Chu, Lucian Popa, Arsalan Tavakoli, Joseph M. Hellerstein, Philip Levis, Scott Shenker, and Ion Stoica. The design and implementation of a declarative sensor network system. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 175–188, New York, NY, USA, 2007. ACM.
- [98] Raghu Ramakrishnan and Jeffrey D. Ullman. A survey of deductive database systems. *The journal of logic programming*, 23:125–149, May 1995.
- [99] Amol Bakshi, Viktor K. Prasanna, Jim Reich, and Daniel Larner. The abstract task graph: a methodology for architecture-independent programming of networked sensor systems. In *Proceedings of the 2005 workshop on End-to-end, sense-and-respond systems, applications and services*, EESR '05, pages 19–24, Berkeley, CA, USA, 2005. USENIX Association.
- [100] Ramakrishna Gummadi, Omprakash Gnawali, and Ramesh Govindan. Macro-programming wireless sensor networks using Kairos. In *Distributed Computing in Sensor Systems*, Lecture Notes in Computer Science. 2005.
- [101] Timothy W. Hnat, Tamim I. Sookoor, Pieter Hooimeijer, Westley Weimer, and Kamin Whitehouse. Macrolab: a vector-based macroprogramming framework for cyber-physical systems. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 225–238, New York, NY, USA, 2008. ACM.
- [102] Ryan Newton and Matt Welsh. Region streams: functional macroprogramming for sensor networks. In *Proceedings of the 1st international workshop on Data management for sensor networks: in conjunction with VLDB 2004*, DMSN '04, pages 78–87, New York, NY, USA, 2004. ACM.
- [103] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Longman Publishing Co., Inc., 2002.
- [104] Leslie Lamport and Lawrence C. Paulson. Should your specification language be typed. *ACM Trans. Program. Lang. Syst.*, 21:502–526, May 1999.
- [105] Jerzy Martyna. Linking simulation with formal verification and modeling of wireless sensor network in tla+. In *Computer Networks*, volume 79 of *Communications in Computer and Information Science*, pages 131–140. Springer-Verlag Berlin Heidelberg, 2010.
- [106] Asad Awan, Ahmed Sameh, Suresh Jagannathan, and Ananth Grama. Building verifiable sensing applications through temporal logic specification. In *Proceedings of the 7th international conference on Computational Science, Part I: ICCS 2007*, ICCS '07, pages 1205–1212, Berlin, Heidelberg, 2007. Springer-Verlag.
- [107] Matt Welsh and Geoff Mainland. Programming sensor networks using abstract regions. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, pages 3–3, Berkeley, CA, USA, 2004. USENIX Association.
- [108] J. F. Gracey, David S. Collins, and Robert J. Huey. *Meat Hygiene*. Saunders Ltd., 1999.
- [109] R. Nullmeyer, D. Stella, G. Montijo, and St. Harden. Human factors in air force flight mishaps: Implications for change. 2005.

- [110] Ann Blandford and B. L. William Wong. Situation awareness in emergency medical dispatch. *International Journal of Human-Computer Studies*, 61:421–452, October 2004.
- [111] J. C. Gorman, N. J. Cooke, and J. L. Winner. Measuring team situation awareness in decentralized command and control environments. *Ergonomics*, 49:1312–1325, 2006.
- [112] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Rec.*, 32:5–14, June 2003.
- [113] Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Monitoring streams: a new class of data management applications. In *Proceedings of the 28th international conference on Very Large Data Bases, VLDB '02*, pages 215–226. VLDB Endowment, 2002.
- [114] D. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E Ryvkina, N Tatbul, Y. Xing, and S Zdonik. The design of the Borealis stream processing engine. In *Conference on Innovative Data Systems Research (CIDR)*, 2005.
- [115] Lewis Girod, Kyle Jamieson, Yuan Mei, Ryan Newton, Stanislav Rost, Arvind Thiagarajan, Hari Balakrishnan, and Samuel Madden. Wavescope: a signal-oriented data stream management system. In *Proceedings of the 4th international conference on Embedded networked sensor systems, SenSys '06*, pages 421–422, New York, NY, USA, 2006. ACM.
- [116] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.
- [117] Chenyang Lu, John A. Stankovic, Sang H. Son, and Gang Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms*. *Real-Time Systems*, 23:85–126, July 2002.
- [118] Chenyang Lu, Ying Lu, Tarek F. Abdelzaher, John A. Stankovic, and Sang Hyuk Son. Feedback control architecture and design methodology for service delay guarantees in web servers. *IEEE Trans. Parallel Distrib. Syst.*, 17:1014–1027, September 2006.
- [119] Lennart Ljung. *System identification (2nd ed.): theory for the user*. Prentice Hall PTR, 1999.
- [120] E. Lee and L. Markus. *Foundations of optimal control theory*. New York: Wiley, 1967.
- [121] L. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *28th IEEE Transactions on Systems, Man, and Cybernetics*, 1:28 – 44, 1973.
- [122] Shin-ichi Horikawa, Takeshi Furuhashi, and Yoshiki Uchikawa. On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm. *Neural Networks, IEEE Transactions on*, 3(5):801–806, 1992.
- [123] Jyh-Shing R. Jang. Self-learning fuzzy controllers based on temporal backpropagation. *Neural Networks, IEEE Transactions on*, 3(5):714 –723, 1992.
- [124] Ahmet Arslan and Mehmet Kaya. Determination of fuzzy logic membership functions using genetic algorithms. *Fuzzy Sets and Systems*, 118(2):297 – 306, 2001.
- [125] George J. Klir and Bo Yuan. *Fuzzy sets and fuzzy logic: theory and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.
- [126] Ch. Schmid. Course on dynamics of multidisciplinary and controlled systems. <http://www.atp.ruhr-uni-bochum.de/rti/syscontrol/main.html>, 2005.
- [127] NRC FuzzyJ Toolkit. <http://www.csie.ntu.edu.tw/sylee/courses/FuzzyJ/Docs/>.
- [128] Building and fire research laboratory. <http://smokealarm.nist.gov/>.

- [129] WS4916 Series Wireless Smoke Detector. <http://www.alarmsuperstore.com/dsc/WS4916Installation.pdf>.
- [130] Justin Geiman. Evaluation of smoke detector response estimation methods. Master's thesis, University of Maryland, College Park, 2003.
- [131] Charles Smith. Smoke detector operability survey: Report on findings. U.S. Consumer Product Safety Commission, 1994.
- [132] Charles D. Litton. Laboratory evaluation of smoke detectors for use in underground mines. *Fire Safety Journal*, 44(3):387 – 393, 2009.
- [133] David D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *ECML*, 1998.
- [134] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.
- [135] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. In *SIGKDD Explorations*, volume 11, 2009.
- [136] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse¹, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. TinyOS: An operating system for sensor networks. In *Ambient Intelligence*, 2005.
- [137] CrossBow. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf.
- [138] A. Kolawa and D. Huizinga. *Automated Defect Prevention: Best Practices in Software Management*. Wiley-IEEE Computer Society Press, 2007.
- [139] H. Liu, L. Selavo, and J. Stankovic. Seedtv: deployment-time validation for wireless sensor networks. In *Proceedings of the 4th workshop on Embedded networked sensors*, EmNets '07, pages 23–27, New York, NY, USA, 2007. ACM.
- [140] Matthias Dyer, Jan Beutel, Thomas Kalt, Patrice Oehen, Lothar Thiele, Kevin Martin, and Philipp Blum. Deployment support network a toolkit for the development of wsns. In *Proceedings of the 4th European conference on Wireless sensor networks*, EWSN'07, pages 195–211, Berlin, Heidelberg, 2007. Springer-Verlag.
- [141] Matthias Ringwald, Kay Römer, and Andrea Vitaletti. Passive inspection of sensor networks. In *Proceedings of the 3rd IEEE international conference on Distributed computing in sensor systems*, DCOSS'07, pages 205–222, Berlin, Heidelberg, 2007. Springer-Verlag.
- [142] David Gluch and Andrew Kornecki. Automated code generation for safety-related applications: A case study. *Computer Science*, 8:37–48, 2007.
- [143] Yafeng Wu, Krasimira Kapitanova, Jingyuan Li, John A. Stankovic, Sang H. Son, and Kamin Whitehouse. Run time assurance of application-level requirements in wireless sensor networks. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 197–208, New York, NY, USA, 2010. ACM.
- [144] Timothy W. Hnat, Vijay Srinivasan, Jiakang Lu, Tamim I. Sookoor, Raymond Dawson, John Stankovic, and Kamin Whitehouse. The hitchhiker's guide to successful residential sensing deployments. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 232–245, New York, NY, USA, 2011. ACM.
- [145] D. Cook and M. Schmitter-Edgecombe. Assessing the quality of activities in a smart environment. *Methods of Information in Medicine*, 48:480–485, 2009.

- [146] Tim van Kasteren, Athanasios Noulas, Gwenn Englebienne, and Ben Kröse. Accurate activity recognition in a home setting. In *Proceedings of the 10th international conference on Ubiquitous computing*, UbiComp '08, pages 1–9, New York, NY, USA, 2008. ACM.
- [147] Cougar. Cornell Database Group-Cougar. <http://www.cs.cornell.edu/bigreddata/cougar/>, 2010.
- [148] Chris Olston, Jing Jiang, and Jennifer Widom. Adaptive filters for continuous queries over distributed data streams. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 563–574, New York, NY, USA, 2003. ACM.
- [149] Bin Liu, Yali Zhu, Mariana Jbantova, Bradley Momberger, and Elke A. Rundensteiner. A dynamically adaptive distributed system for processing complex continuous queries. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 1338–1341. VLDB Endowment, 2005.
- [150] Yi-Cheng Tu, Song Liu, Sunil Prabhakar, and Bin Yao. Load shedding in stream databases: a control-based approach. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 787–798. VLDB Endowment, 2006.
- [151] Yuan Wei, Vibha Prasad, and Sang H. Son. Qos management of real-time data stream queries in distributed environments. In *Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, ISORC '07, pages 241–248, Washington, DC, USA, 2007. IEEE Computer Society.
- [152] Peter J. Haas, Jeffrey F. Naughton, and Arun N. Swami. On the relative cost of sampling for join selectivity estimation. In *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '94, pages 14–24, New York, NY, USA, 1994. ACM.
- [153] Viswanath Poosala and Yannis E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 486–495, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
- [154] Douglas Comer. The ubiquitous B-Tree. *ACM Computing Surveys*, 11:121–137, June 1979.
- [155] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.
- [156] D. Barbara, W. DuMouchel, C. Faloutsos, P. Hass, J. Hellerstein, Y. Ioannidis, H. Jagadish, T. Johnson, R. Ng, V. Poosala, K. Ross, and K. Sevcik. The New Jersey data reduction report. Technical report, Bulletin of the Technical Committee on Data Engineering, 1997.
- [157] Surajit Chaudhuri, Gautam Das, Mayur Datar, Rajeev Motwani, and Vivek R. Narasayya. Overcoming limitations of sampling for aggregation queries. In *Proceedings of the 17th International Conference on Data Engineering*, pages 534–542, Washington, DC, USA, 2001. IEEE Computer Society.
- [158] Annita N. Wilschut and Peter M. G. Apers. Dataflow query execution in a parallel main-memory environment. In *Proceedings of the first international conference on Parallel and distributed information systems*, PDIS '91, pages 68–77, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.
- [159] Wei Hong and Michael Stonebraker. Optimization of parallel query execution plans in xprs. volume 1, pages 9–32. Springer Netherlands, 1993. 10.1007/BF01277518.
- [160] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *Conference on Innovative Data Systems Research (CIDR)*, pages 245–256, 2003.
- [161] Usa Sammapun, Insup Lee, and Oleg Sokolsky. Rt-mac: Runtime monitoring and checking of quantitative and probabilistic properties. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '05, pages 147–153, Washington, DC, USA, 2005. IEEE Computer Society.

- [162] Usa Sammapun, Insup Lee, Oleg Sokolsky, and John Regehr. Statistical runtime checking of probabilistic properties. In *Proceedings of the 7th international conference on Runtime verification*, RV'07, pages 164–175. Springer-Verlag, Berlin, Heidelberg, 2007.
- [163] T. L. M. van Kasteren, G. Englebienne, and B. J. A. Kröse. Activity recognition using semi-Markov models on real world smart home datasets. *Journal of Ambient Intelligence and Smart Environments*, 2:311–325, August 2010.
- [164] Beth Logan, Jennifer Healey, Matthai Philipose, Emmanuel Munguia Tapia, and Stephen Intille. A long-term evaluation of sensing modalities for activity recognition. In *Proceedings of the 9th international conference on Ubiquitous computing*, UbiComp '07, pages 483–500, Berlin, Heidelberg, 2007. Springer-Verlag.
- [165] Matthai Philipose, Kenneth Fishkin, Mike Perkowitz, Donald Patterson, Dieter Fox, Henry Kautz, and Dirk Hahnel. Inferring activities from interactions with objects. *IEEE Pervasive Computing*, 3, 2004.
- [166] Emmanuel Munguia Tapia, Stephen Intille, and Kent Larson. Activity recognition in the home using simple and ubiquitous sensors. In *Pervasive Computing*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004.
- [167] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT*, pages 23–37, London, UK, UK, 1995. Springer-Verlag.